به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین اول

نام و نام خانوادگی	مهسا راستی – مینا سیدی
شماره دانشجویی	810398084 - 810198393
تاریخ ارسال گزارش	1401.09.18

فهرست

شکل‌ها

جدول‌ها

پاسخ ۱. آشنایی با یادگیری انتقالی Transfer Learning

.1.1

سرطان پوست یکی از شایعترین نوع سرطان‌ها است که ابتدا به کمک دید بصری متخصص و سپس با کمک آنالیز موسکوپی انجام می‌شود. تشخیص نوع و طبقه بندی سلول‌های سرطانی در مدت زمان طولانی با بررسی متوالی تغییر رنگدانه‌های پوستی صورت می‌پذیرد. با کمک شبکه عصبی VGG19، میتوان دقت طبقه‌بندی سلول‌های سرطانی پوست را ارتقا داد. در این مقاله دو نوع سلول سرطانی پوست Dermatofibroma و Basal Cell Carcinoma و سلول غیر سرطانی Keratosis-like Lesions طبقه‌بندی می شود.

.1.2

معماری شبکه VGG19، از بلوک Convlotionی تشکیل شده که با MaxPool به عنوان لایه ادغام به همدیگر متصل میشوند. لایه کانولوشنی نقشه ویژگی‌های ورودی را استخراج میکند و لایه ادغام اطلاعات تولید شده توسط لایه کانولوشنی را کاهش میدهد. لایه‌ها در این شبکه به صورت Fully Connected است یعنی وزن ها را روی ورودی های تولید توسط تحلیل ویژگی اعمال میکند. در هر لایه از شبکه، چند تبدیل Convlotion انجام شده و خروجی این لایه به عنوان ورودی لایه بعدی، مورد استفاده قرار می‌گیرد. درنهایت خروجی از فرم خطی 4096تایی به فرم خطی 1000 تایی تبدیل می شود. لایه هایی که به دیتای ورودی اولیه نزدیکتر هستند، تعداد Convlotion کمتری روی آنها اعمال میشود در نتیجه پهن‌تر هستند اما لایه‌های دورتر تبدیل Convlotion بیشتری روی آنها اعمال شده و در نتیجه عمیق‌تر هستند. این شبکه به صورت هرمی دیده شده و ابعاد عکسی رفته‌رفته کاهش می‌یابد. در این مقاله، بر روی خروجی تابع Softmax اعمال شده و فرم خطی 1000 تایی به فرم خطی 3 تایی(تعداد خروجی های BCC، KL و DF) تبدیل می شود. استفاده از خروجی لایه قبلی به عنوان ورودی لایه بعدی یکی از خصوصیات مثبت این شبکه است. البته همین کار باعث افزایش مدت زمان Train شبکه شده و از نظر زمانی بهینه نخواهد بود چرا که تعداد داده های بیشتری به عنوان داده Train وارد شبکه خواهد شد. همچنین با تغییر این شبکه، مدت زمان اجرای آن به شدت کندتر می‌شود و میتوان گفت نمیتوان آن را به شبکه مورد نظر تعلیم داد.

پیش پردازش‌هایی که بر روی داده های شبکه به منظور افزایش دیتاست و ارتقا دقت، صحت و ... انجام می شود، برش عکس، تغییر مقیاس(تبدیل Affine)، تغییر روشنایی، تغییر کنتراست، دوران افقی، دوران عمودی و تغییر ابعاد عکس به 64*64 است. همه پارامترهای این پیش پردازش به صورت تصادفی انتخاب می‌شود.

.1.3

این شبکه بر روی مجموعه داده های ImageNet آموزش دیده است که این داده ها را به 1000 دسته تقسیم میکند. سپس با اعمال تغییرات کوچکی، این شبکه را برای داده های تصاویر پوستی، اجرا میکنیم. شبکه VGG19 تغییر یافته، تصاویر پوست را به سه دسته سلول سرطانی پوست Dermatofibroma و Basal Cell Carcinoma و سلول غیر سرطانی Keratosis-like Lesions تقسیم میکند. در مقاله تنها این 3 حالت تعریف شده و تمامی تصاویر ورودی در یکی از این سه دسته قرار می گیرند. در صورتی که تصویر ورودی متعلق به هیچ یکی از این دسته ها نباشد، می توان یا دسته چهارمی به شبکه اضافه نمود و یا عکس را متعلق به دسته ای بدانیم که بالاترین احتمال را در پیش‌بینی دارد.

.1.4

ابتدا فایل API را از روی صفحه کاربری Kaggle دانلود کرده و در Colab آپلود میکنیم.

شکل 1. وارد کردن فایل API سایت Kaggle به Colab

سپس یک Directory در Colab برای انتقال داده ها از Kaggle به Colab ایجاد میکنیم.

```
!rm -r ~/.kaggle #Remove Old Kaggle Directory
!mkdir ~/.kaggle #Create Kaggle Directory
!mv ./kaggle.json ~/.kaggle/ #Move kaggle.json in Directory
!chmod 600 ~/.kaggle/kaggle.json #Allocate the Required Permission for File.
!kaggle datasets list #Show List of Dataset
```

```
ref                                                             title
--------------------------------------------------------------  ----------------------------------------------
meirnizri/covid19-dataset                                       COVID-19 Dataset
madhurpant/world-deaths-and-causes-1990-2019                    World Deaths and Causes (1990 - 2019)
thedevastator/jobs-dataset-from-glassdoor                       Salary Prediction
thedevastator/how-much-sleep-do-americans-really-get            How Much Sleep Do Americans Really Get?
theakhilb/layoffs-data-2022                                     Layoffs Dataset 2022
bwandowando/daily-spotify-top-50-of-60-countries                Spotify Daily Top 50 Of 59 countries + Global
akshaydattatraykhare/data-for-admission-in-the-university       Data for Admission in the University
aguado/bike-rental-data-set-uci                                 Bike Rental Data Set - UCI
dbarteaux99/stable-diffusion-1-5                                Stable Diffusion 1.5 (normal and EMAonly) with
whenamancodes/predict-diabities                                 Predict Diabetes
prosperchuks/health-dataset                                     Diabetes, Hypertension and Stroke Prediction
mvieira101/global-cost-of-living                                Global Cost of Living
thedevastator/cancer-patients-and-air-pollution-a-new-link      Lung Cancer Prediction
thedevastator/nike-usa-products-prices-descriptions-and-custom  Nike Products: Prices, Descriptions, Reviews
piterfm/fifa-football-world-cup                                 FIFA Football World Cup
notshrirang/spotify-million-song-dataset                        Spotify Million Song Dataset
akshaydattatraykhare/diabetes-dataset                           Diabetes Dataset
catherinerasgaitis/mxmh-survey-results                          Music & Mental Health Survey Results
malayvyas/usa-gdp-dataset-19612021                              USA GDP Growth Dataset 1961-2021
sulphatet/daily-weather-data-40-years                           daily weather data: 40 years
```

شکل 2. ساخت Directory در Colab

دیتاست های مقاله را از سایت Kaggle دانلود می‌کنیم(حجم داده ها بسیار زیاد است به همین دلیل از دانلود و سپس آپلود مستقیم داده ها خودداری می‌شود).

```
[3]  1  !kaggle datasets download -d umangjpatel/ham10000-imagenet-style-dataset

Downloading ham10000-imagenet-style-dataset.zip to /content
100% 2.58G/2.58G [01:28<00:00, 33.5MB/s]
100% 2.58G/2.58G [01:28<00:00, 31.5MB/s]
```

شکل 3. دانلود دیتاست های مقاله از Kaggle

دیتاهای دانلود شده در یک فایل zip. به صورت فولدرهای جداگانه قرار دارند که آنها را unzip کرده و از حالت فشرده خارج می‌کنیم.

```
1  !unzip ham10000-imagenet-style-dataset.zip

inflating: nv/ISIC_0032176.jpg
inflating: nv/ISIC_0032177.jpg
inflating: nv/ISIC_0032178.jpg
inflating: nv/ISIC_0032180.jpg
inflating: nv/ISIC_0032181.jpg
inflating: nv/ISIC_0032183.jpg
inflating: nv/ISIC_0032184.jpg
inflating: nv/ISIC_0032186.jpg
inflating: nv/ISIC_0032188.jpg
inflating: nv/ISIC_0032189.jpg
```

شکل 4. Unzip کردن داده ها

.1.5

کتابخانه های مورد استفاده را فراخوانی می‌کنیم.

```
!pip install torchmetrics

import torch
import numpy as np
import torch.nn as nn
import matplotlib.pyplot as plt
import torch.nn.functional as F
from torch.utils.data import DataLoader, Dataset
from torchmetrics.classification import BinaryF1Score
from torchmetrics.classification import BinaryAccuracy
from torchmetrics.classification import BinaryPrecision
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting torchmetrics
  Downloading torchmetrics-0.11.0-py3-none-any.whl (512 kB)
     |████████████████████████████████| 512 kB 6.2 MB/s
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.8/dist-packages (from torchmetrics) (4.1.1)
Requirement already satisfied: numpy>=1.17.2 in /usr/local/lib/python3.8/dist-packages (from torchmetrics) (1.21.6)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from torchmetrics) (21.3)
Requirement already satisfied: torch>=1.8.1 in /usr/local/lib/python3.8/dist-packages (from torchmetrics) (1.12.1+cu113)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist-packages (from packaging->torchmetrics)
Installing collected packages: torchmetrics
Successfully installed torchmetrics-0.11.0
```

شکل 5. فراخوانی کتابخانه‌ها

یک Device برای اجرای برنامه در آن محیط ایجاد می‌کنیم.

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

شکل 6. ایجاد کردن Device

شبکه VGG19 را از کتابخانه torchvision دانلود کرده و وارد برنامه می‌کنیم.

```
VGG19 = torchvision.models.vgg19_bn(weights=True)
VGG19
```

```
/usr/local/lib/python3.8/dist-packages/torchvision/models/_utils.py:223: UserWarning: Argumen
  warnings.warn(msg)
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (7): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (9): ReLU(inplace=True)
    (10): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (12): ReLU(inplace=True)
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (14): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (16): ReLU(inplace=True)
```

شکل 7. دانلود VGG19

یک کلاس به نام My_Data می‌سازیم که در آن preprecoss های گفته شده در قسمت **1.2**. انجام شده و همچنین تصویر به torch تبدیل شده و کلاسی که به آن تعلق دارد، به صورت one-hot در خروجی ظاهر می‌شود.

```
class My_Data():
  def __init__(self, x):
    self.x = x
    preprocess = transforms.Compose([
        transforms.ToTensor(),
        transforms.RandomCrop(size=(64, 64)),
        transforms.RandomAffine(0),
        # transforms.ColorJitter(brightness=0, contrast=0, saturation=0, hue=0),
        transforms.RandomAutocontrast(p=0.5),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomVerticalFlip(p=0.5),
        transforms.Resize([64, 64])
    ])
    self.preprocess = preprocess

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

  def __getitem__(self, index):
    image = Image.open(self.x[index][0])
    image = self.preprocess(image)
    image = image.convert("RGB")

    label = torch.zeros((3, 1))
    label[self.x[index][1]] = 1
    return image, label
```

شکل 8. کلاس My_Data

با توجه به توضیحات داده شده در مقاله، مدل را تغییر می‌دهیم.

```
model=nn.Sequential(
    VGG19,
    nn.ReLU(),
    nn.Linear(1000,1000),
    nn.ReLU(),
    nn.Linear(1000,3)
)
```

شکل 9. تغییر شبکه

Optimizer، Loss Function، Accuracy، Precision و F1Score را تعریف می‌کنیم.

```
optimizer = torch.optim.Adam(model.parameters(), lr = 0.01)
loss_function = nn.CrossEntropyLoss()

acc = MulticlassAccuracy(num_classes=3)
pre = MulticlassPrecision(num_classes=3)
f1 = MulticlassF1Score(num_classes=3)
softmax = nn.Softmax(dim = 1)
```

شکل 10. معرفی Optimizer، Loss Function، Accuracy، Precision و F1Score

مدل و توابع معرفی شده در قسمت قبلی را وارد Device می‌کنیم.

```
[52] model.to(device)
     loss_function.to(device)
     acc.to(device)
     pre.to(device)
     f1.to(device)
```

شکل 11. وارد کردن مدل و توابع به Device

80% تصاویر از دسته‌های BCC، BKL و DF را به داده‌های Train و 20٪ باقی‌مانده را به داده‌های Test تقسیم میکنیم. در نهایت آنها را در train_list و test_list دخیره می‌کنیم.

```
bcc_List, bkl_List, df_List = [], [], []

bcc_path = os.listdir('/content/bcc')
for image_path in bcc_path:
  bcc_List.append(['/content/bcc' + '/' + image_path, 0])
train_list1, test_list1 = train_test_split(bcc_List, test_size = 0.2, random_state=10)


bkl_path = os.listdir('/content/bkl')
for image_path in bkl_path:
  bkl_List.append(['/content/bkl' + '/' + image_path, 1])
train_list2, test_list2 = train_test_split(bkl_List, test_size = 0.2, random_state=10)

df_path = os.listdir('/content/df')
for image_path in df_path:
  df_List.append(['/content/df' + '/' + image_path, 2])
train_list3, test_list3 = train_test_split(df_List, test_size = 0.2, random_state=10)

train_list = train_list1+train_list2+train_list3
test_list = test_list1+test_list2+test_list3
```

شکل 12. ساخت train_list و test_list

با DataLoader، داده‌های Train , Test را به صورت خوشه‌ای وارد مدل می‌کنیم.

```
train_data = My_Data(train_list)
test_data = My_Data(test_list)

train = DataLoader(train_data, batch_size=50, shuffle=True)
test = DataLoader(test_data, batch_size=50, shuffle=False)
```

شکل 13. ساخت DataLoader

مدل ساخته شده را با داده‌های Train و Test آموزش می‌دهیم و اپکی که بیشترین مقدار دقت را دارد، به عنوان بهترین مدل ذخیره می‌کنیم.

```
epochs_number = 100
acc_list, train_loss_list, test_loss_list = [], [], []

for epoch in range(epochs_number):
  accuracy, precision, f1score, train_loss, test_loss = [], [], [], [], []
  model.train()
  for data, pred in train:
    data = data.to(device)
    pred = pred.to(device)
    pred = pred.squeeze(2)
    optimizer.zero_grad()
    output = model(data)
    loss = loss_function(output, torch.tensor(pred).to(device))
    loss.backward()
    train_loss.append(loss.item())
    optimizer.step()
```

```
  with torch.no_grad():
    model.eval()
    for data, pred in test:
      data = data.to(device)
      pred = pred.to(device)
      pred = pred.squeeze(2)
      outputs = model(data)
      outputs = softmax(outputs)
      los = loss_function(outputs, pred)
      test_loss.append(los.item())
      accuracy.append(acc(outputs.cpu(), pred.cpu()))
      precision.append(pre(outputs.cpu(), pred.cpu()))
      f1score.append(f1(outputs.cpu(), pred.cpu()))

    train_loss_list.append(sum(trian_loss)/len(train_loss))
    test_loss_list.append(sum(test_loss)/len(test_loss))
    acc_list.append(sum(accuracy)/len(accuracy)*100)

    if(accuracy[-1]>=max(accuracy)):
      print(f'in epoch {epoch}'+' Model Saved!')
      torch.save(model, 'saved_model.pth')

    print(f'epoch {epoch} : acc {sum(accuracy)/len(accuracy)*100}   pre {sum(pre
```

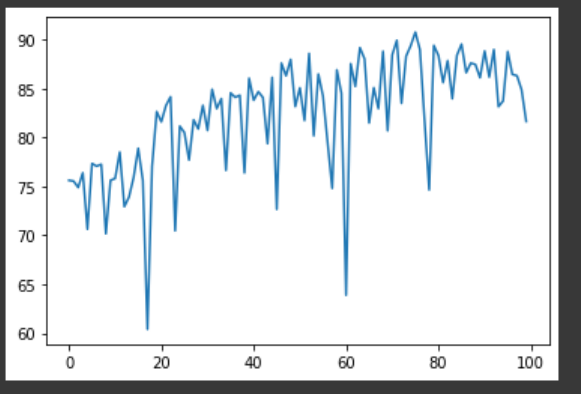
```
<ipython-input-14-40034fc4f719>:14: UserWarning: To copy construct from a tensor, it is recommended to use sourceTensor.clone().detach() or sourceTensor.clone(
  loss = loss_function(output,torch.tensor(pred).to(device))
Model Saved in epoch 0!
epoch 0 : acc 75.6190414428711   pre 63.42857360839844   f1 63.42857360839844   train 1.0982084657464708   test 0.9450398938096452
epoch 1 : acc 75.54036712646484   pre 64.79553985595703   f1 60.343414306640625   train 0.8175467721053532   test 0.9276191762515477
epoch 2 : acc 74.8944091796875   pre 65.45880889892578   f1 56.773250579833984   train 0.8182756326028279   test 0.943466126918792 7
Model Saved in epoch 3!
epoch 3 : acc 76.38095092773438   pre 64.92865753173828   f1 62.64634704589844   train 0.8380569304738726   test 0.9519747069903782
epoch 4 : acc 70.633544921875   pre 56.278438568115234   f1 54.763404846191406   train 0.8028660693338939   test 0.9727574842316764
Model Saved in epoch 5!
epoch 5 : acc 77.33333587646484   pre 65.88819885253906   f1 65.20598602294922   train 0.8224628652845111   test 0.9538033860070365
epoch 6 : acc 77.0641860961914   pre 63.869049072265625   f1 53.1077995300293   train 0.7992291471787861   test 0.9279389296259198
epoch 7 : acc 77.2546615600586   pre 70.96833801269531   f1 60.246498107910156   train 0.7984216191938945   test 0.9581560237067086
epoch 8 : acc 70.1697769165039   pre 58.785282135009766   f1 41.67434310913086   train 0.811725409675587   test 1.0112671766962325
epoch 9 : acc 75.6190414428711   pre 63.42857360839844   f1 63.42857360839844   train 0.811332996402468   test 0.9079610194478717
epoch 10 : acc 75.80951690673828   pre 63.42857360839844   f1 63.42857360839844   train 0.7770963822092328   test 0.9148112790925162
Model Saved in epoch 11!
epoch 11 : acc 78.50103759765625   pre 71.39798736572266   f1 58.83539962768555   train 0.7931636380297797   test 0.9030659454209464
epoch 12 : acc 72.9358139038086   pre 59.761192321777344   f1 58.07523727416992   train 0.7682236305304936   test 0.9525981971195766
epoch 13 : acc 73.91304016113281   pre 64.57142639160156   f1 42.63470458984375   train 0.7737845778465271   test 0.9685061659131732
epoch 14 : acc 75.90475463867188   pre 63.41107940673828   f1 63.275611877441406   train 0.7715565030063901   test 0.8676281486238752
Model Saved in epoch 15!
epoch 15 : acc 78.88612365722656   pre 69.718994140625   f1 67.26285552978516   train 0.7456410356930324   test 0.8931910991668701
epoch 16 : acc 75.6190414428711   pre 63.42857360839844   f1 63.42857360839844   train 0.7325732452528817   test 0.9184185607092721
epoch 17 : acc 60.41408157348633   pre 40.62112045288086   f1 40.62112045288086   train 0.7252102004630225   test 1.0446434702192033
```

```
epoch 75 : acc 90.74948120117188   pre 87.6654052734375   f1 85.45773315429688   train 0.4126169867813587   test 0.7315561856542315
epoch 76 : acc 88.9482421875   pre 86.49898529052734   f1 82.3153076171875   train 0.39293893639530453   test 0.7496345639228821
epoch 77 : acc 81.66046142578125   pre 72.37030029296875   f1 71.29452514648438   train 0.4117927551269531   test 0.8205459884234837
epoch 78 : acc 74.64181518554688   pre 61.230499267578125   f1 59.911094665527344   train 0.408707575074264   test 0.8964886239596775
epoch 79 : acc 89.39131164550781   pre 88.18834686279297   f1 82.88591766357422   train 0.4130161702632904   test 0.7558851497513908
epoch 80 : acc 88.3685302734375   pre 85.9630126953125   f1 80.74555206298828   train 0.405778137168713 97   test 0.766101496560233
epoch 81 : acc 85.60248565673828   pre 81.00900268554688   f1 77.45431518554688   train 0.41701467441661016   test 0.7756380098206657
epoch 82 : acc 87.83851623535156   pre 85.45266723632812   f1 80.23365020751953   train 0.39651436678000856   test 0.7766582454953875
epoch 83 : acc 83.95446014404297   pre 76.38288116455078   f1 74.9913330078125   train 0.4166855865291187   test 0.7908538154193333
epoch 84 : acc 88.38923645019531   pre 84.08458709716797   f1 81.77558135986328   train 0.40669940784573555   test 0.7366312657083783
epoch 85 : acc 89.5238037109375   pre 85.2235107421875   f1 83.82877349853516   train 0.42305052067552296   test 0.7276836974280221
epoch 86 : acc 86.61283874511719   pre 82.56488800048828   f1 78.61624908447266   train 0.3966745547950268   test 0.7915010792868478
epoch 87 : acc 87.58592224121094   pre 84.38473510742188   f1 80.0304946899414   train 0.39514797925949097   test 0.7642858879906791
epoch 88 : acc 87.45342254638672   pre 84.45124816894531   f1 79.92094421386719   train 0.37858930070485386   test 0.7714911784444537
epoch 89 : acc 86.10352325439453   pre 82.84231567382812   f1 76.9734878540039   train 0.391816503235272   test 0.8044196452413287
epoch 90 : acc 88.81988525390625   pre 84.63146209716797   f1 82.49110412597656   train 0.3954284137913159   test 0.7409692917551313
epoch 91 : acc 86.16563415527344   pre 80.72830200195312   f1 78.51549530029297   train 0.3787208949880941   test 0.775859705039433
epoch 92 : acc 88.96894836425781   pre 83.90946960449219   f1 83.22908020019531   train 0.38034127546208246   test 0.7216806326593671
epoch 93 : acc 83.14286041259766   pre 75.67900848388672   f1 73.23467254638672   train 0.3650415763258934   test 0.8145332251276288
epoch 94 : acc 83.7267074584961   pre 76.0827865600586   f1 74.8980484008789   train 0.36141741648316383   test 0.787240326404571 5
epoch 95 : acc 88.77018737792969   pre 83.39838409423828   f1 82.8162841796875   train 0.37224431708455086   test 0.7196304202079773
epoch 96 : acc 86.43892669677734   pre 80.72955322265625   f1 79.21249389648438   train 0.3848756349512509   test 0.7694009287016732
epoch 97 : acc 86.30227661132812   pre 80.61145782470703   f1 78.63939666748047   train 0.365973031946591   test 0.7793828248977661
epoch 98 : acc 84.86956787109375   pre 77.82611846923828   f1 76.65171813964844   train 0.4079419821500778   test 0.8123618364334106
epoch 99 : acc 81.63975524902344   pre 72.95004272460938   f1 72.10244750976562   train 0.3673451031957354   test 0.8270502856799534
```

شکل 14. آموزش شبکه

نمودار مقدار دقت در 100 اپک را رسم می‌کنیم. مشاهده می‌شود با صرف نظر از برخی اپک‌ها، میزان دقت با افزایش اپک، افزایش می‌یابد.

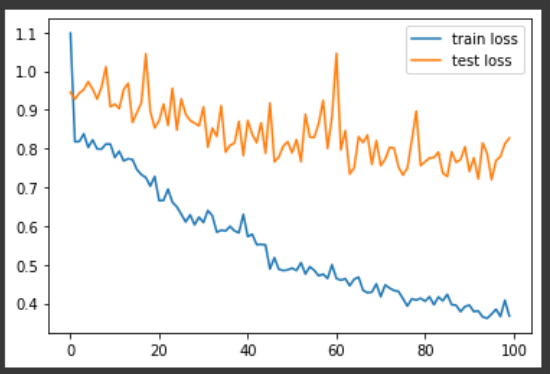
```
import matplotlib.pyplot as plt
plt.plot(acc)
plt.show()
```



شکل 15. نمودار دقت

نمودار train loss و test loss را هم رسم می‌کنیم. می‌بینیم train loss با افزایش اپک کاهش می‌یابد.

```
plt.plot(loss_train)
plt.plot(loss_test)
plt.legend(['train loss', 'test loss'])
plt.show()
```



شکل 16. نمودار train loss و test loss

بهترین مدل ذخیره شده را بر روی داده‌های Test اعمال می‌کنیم و Accuracy، Precision، F1Score و Test Loss را محاسبه می‌کنیم.

```
model.eval()

datas = torch.Tensor().to('cuda')
preds = torch.Tensor().to('cuda')
acc_list, pre_list, f1_list, test_loss_list = [], [], [], []

with torch.no_grad():
  for data, pred in test:
    data = data.to(device)
    pred = pred.to(device)
    pred = pred.squeeze(2)
    outputs = model(data)
    outputs = softmax(outputs)
    los = loss_function(outputs, pred)
    test_loss_list.append(los.item())
    acc_list.append(acc(outputs.cpu(), pred.cpu()))
    pre_list.append(pre(outputs.cpu(), pred.cpu()))
    f1_list.append(f1(outputs.cpu(), pred.cpu()))
    datas = torch.cat((datas, pred))
    preds = torch.cat((preds, outputs))

  print(f'accuracy : {sum(acc_list)/len(acc_list)}')
  print(f'precision : {sum(pre_list)/len(pre_list)}')
  print(f'f1score : {sum(f1_list)/len(f1_list)}')
  print(f'test_loss : {sum(test_loss_list)/len(test_loss_list)}')
```

شکل 17. اعمال بهترین مدل روی داده تست

جدول 1. **Accuracy**، **Precision**، **F1Score** و **Test Loss** بهترین مدل

Accuracy	91.12%
Precision	87.69%
F1score	86.24%
Test Loss	74.69%

برای محاسبه ماتریس طبقه‌بندی، لیستی متشکل از اعداد 0، 1 و 2 می‌سازیم که در واقعیت و در پیشبینی توسط مدل، هر یک از داده‌های تست متعلق به کدام دسته هستند.

```
true = []
for i in range(len(preds)):
  true.append(torch.argmax(preds[i]).item())
print(true)

prediction = []
for i in range(len(datas)):
  prediction.append(torch.argmax(datas[i]).item())
print(prediction)
```

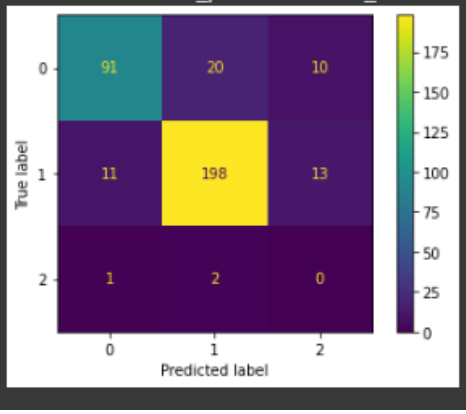
```
[0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

شکل 18. اعمال بهترین مدل روی داده تست

سپس ماتریس طبقه بندی را رسم می‌کنیم.

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
con_mat = confusion_matrix(true, prediction)
ConfusionMatrixDisplay(con_mat).plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f4b7061e040>
```



شکل 19. نمودار ماتریس طبقه‌بندی

پاسخ ۲ – آشنایی با تشخیص چهره مسدود شده

1.2.1.6. خلاصه ساختار شبکه

آموزش با شبکه های CNN، PSPNet و +DeepLabv3 با پشتیبانی ResNet-101 از پیش آموزش داده شده و یک Vision Transformer، SegFormer با پشتیبانی MIT-B5 از پیش آموزش داده شده انجام شده است. مدل ترین شده با دو دیتاست تست شده است.

تمامی آزمایش ها با MMSegmentation انجام شده است. تمامی مدل ها برروی 4 جی پی یوی Tesla V100 ترین شده اند. سایز تصویر ورودی 512×512، batch size 8 و تعداد iteration ها 30k بوده است. Optimizer برای PSPNet و +DeepLabv3، SGD با learning rate 0.01، momentum 0.9 و decay rate 0.0005 بوده است. Optimizer برای SegFormer، AdamW با learning rate 6e-05 و weight decay rate 0.01 بوده است. ارزیابی هر 400 iteration یکبار برروی دیتاست RealOcc صورت میگرفته است. data augmentation هایی شامل random horizontal flips،random rotation of 30 degrees و photometric distortion اعمال شده است. Loss function برای همه آزمایش ها Binary cross-entropy loss با online hard example mining (OHEM) بوده است. Performance مدل ها با مقایسه mIoU ها (the mean Intersection over Union) روی تمام کلاس های دیتاست ارزیابی، ارزیابی شده است.

2.1.7-2. تفاوت Occlusion ها

میان Occlusion ها از لحاظ میزان رزولوشن، بزرگی و رنگ تفاوت وجود دارد ولی همه آن ها در کلاس Background دسته بندی میشوند.

2.1.8-3. کلاس بندی کردن داده ها

دو روش جنریت کردن دیتاهای occluded شده معرفی شد:

1. NatOcc: برای تولید صورت های مسدود شده ترکیبی طبیعی
2. RandOcc: یک روش معمولی تولید دیتا مسدود شده ترکیبی

همچنین برای استفاده گسترده از CelebAMask-HQ به صورت دستی annotation mask ها تصحیح شده و کتگوری های جدید(occluded and non-occluded) تهیه شدند. برای تسهیل

کردن ارزیابی مدل، دو دیتاست از چهره های مسدود شده از دنیای واقعی با annotated mask های دستی، RealOcc (aligned and cropped) و RealOcc-Wild (in the wild) شرکت داده شدند.

همچنین دیتاها شامل دو کلاس کلی میشوند. Background و Face.

Class	Definition
Face	The skin of the head includes eyes, nose, mouth but excluding ears.
Face (Grey Area)	Tattoo, shadow, moustache, and beard overlapped with face, as well as skin of the bald person are considered as part of the face.
Background (Occlusions)	Non-face area and any objects such as sunglasses, shirt, hair, microphone, hands that are physically covering (overlap) the face. Words from magazines or copyright labels fall into this category as well.
Background (Grey Area)	Transparent/Translucent glasses

شکل 20. تعاریف کلاس ها در دیتاست

.1.9

.1.10 2-4. شبکه مناسب برای وقتی که intensity چهره ها با occlusion های مصنوعی متفاوت باشند.

شبکه های Feature-based Methods پیشنهاد میشود. وقتی تفاوت ها بین کلاس ها زیاد است، خیلی نیازی نیست که آن ها را پارتیشن بندی هم بکنیم.

1.11. 2-5. مقایسه کارایی PSPNet و DeepLab

	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 21. مقایسه کارایی PSPNet و DeepLab

این دو شبکه کارایی نزدیک به هم دارند. البته بر روی سه دیتاست اصلی، ترکیب های مختلف دیتاست نتیجه های متفاوتی دارند.

برای هر سه دیتاست شبکه DeepLab روی دیتاهای ترکیبی نتیجه بهتری میدهد و برای داده های غیرترکیبی PSPNet نتیجه بهتری داده است.

پاسخ ۳ – تشخیص بلادرنگ اشیا

1.12. 3-1. شخصی سازی

ابتدا مدل YOLOv6 را از گیت هاب کلون میکنیم.

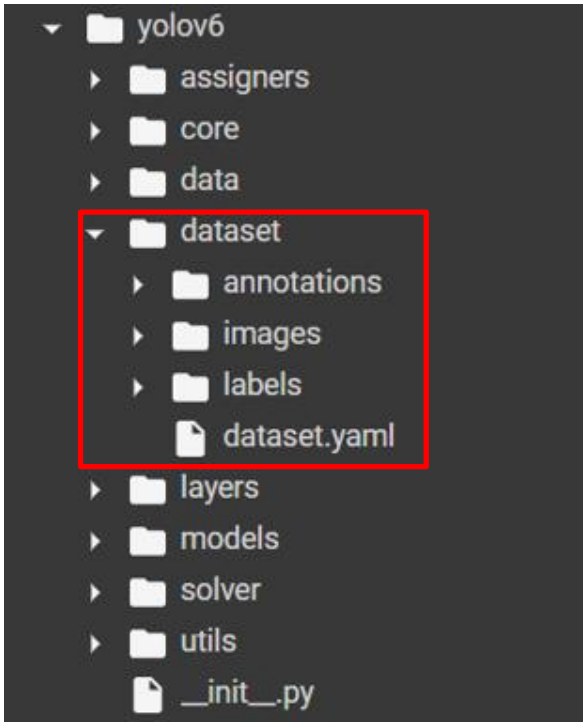
```
!git clone https://github.com/meituan/YOLOv6
```

شکل 22. کلون کردن YOLOv6

برای شخصی سازی باید سه مجموعه داده train- test- evaluation داشته باشیم و برای هرکدام هم سه مجموعه label داشته باشیم.

داده های خود را در دایرکتوری/YOLOv6/yolov6 قرار میدهیم.

```
!unzip /content/YoloV6_Chess.zip
```



شکل 23. قرار دادن دیتاها در دایرکتوری صحیح

سپس به دایرکتوری YOLOv6 میرویم. فایل requirements.txt را اینستال کرده تا تمام کتابخانه های مورد نیاز نصب شوند. سپس مراحل آموزش را روی gpu میبریم.

```
cd YOLOv6
```

```
/content/YOLOv6
```

```
!pip install -r requirements.txt
```

```
import torch
torch.cuda.is_available()
torch.cuda.get_device_name(0)
```

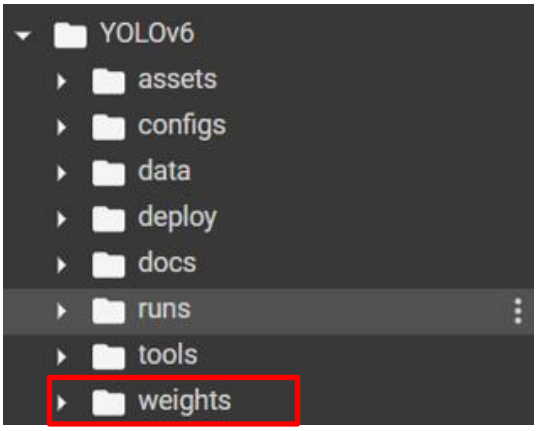
```
'Tesla T4'
```

شکل 24. نصب کتابخانه های مورد نیاز

کد زیر را برای تولید فایل yolov6s.pt ران میکنیم. سپس فولدر جدیدی به نام weights ساخته و آن را در این فولدر قرار میدهیم.

```
torch.hub.download_url_to_file('https://github.com/meituan/YOLOv6/releases/download/0.2.0/yolov6s.pt', 'yolov6s.pt')
```

```
100%  36.3M/36.3M [00:02<00:00, 35.5MB/s]
```



شکل 25. تولید فایل yolov6s.pt

فایل dataset.yaml را میسازیم.

```
train: /content/YOLOv6/yolov6/dataset/images/train
val: /content/YOLOv6/yolov6/dataset/images/valid
test: /content/YOLOv6/yolov6/dataset/images/test

is_coco: False

nc: 13
names: ['bishop', '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 26. dataset.yaml

کد زیر را برای ترین کردن به میزان 10 ایپاک ران میکنیم.

```
!python tools/train.py --batch 16 --conf configs/yolov6s_finetune.py --data-path /content/YOLOv6/yolov6/dataset/dataset.yaml
```

```
--device 0 --epochs 10 --eval-interval 2
```

```
     Epoch  iou_loss  dfl_loss  cls_loss
       9/9    0.4416         0     1.222: 100% 38/38 [00:18<00:00,  2.03it/s]
Inferencing model in train datasets.: 100% 2/2 [00:03<00:00,  1.85s/it]

Evaluating speed.

Evaluating mAP by pycocotools.
Saving runs/train/exp7/predictions.json...
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
Loading and preparing results...
DONE (t=0.19s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.53s).
Accumulating evaluation results...
DONE (t=0.20s).
```

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.232
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.338
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.303
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.374
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.236
 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 ] = 0.245
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.678
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.727
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.749
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.721
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Results saved to runs/train/exp7
Epoch: 9 | mAP@0.5: 0.33756701334247724 | mAP@0.50:0.95: 0.23225509787150994

Training completed in 0.072 hours.
```

شکل 27. نتایج ترین

کد زیر را برای مرحله evaluation ران میکنیم.

```
!python tools/eval.py --data /content/YOLOv6/yolov6/dataset/dataset.yaml  --weights /content/YOLOv6/runs/train/exp7/weights/best_ckpt.pt
```

```
Evaluating mAP by pycocotools.
Saving runs/val/exp7/predictions.json...
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
Loading and preparing results...
DONE (t=0.11s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.51s).
Accumulating evaluation results...
DONE (t=0.19s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.233
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 0.339
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.304
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.374
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.236
 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 ] = 0.242
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.679
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.728
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.749
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.721
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
Results saved to runs/val/exp7
```

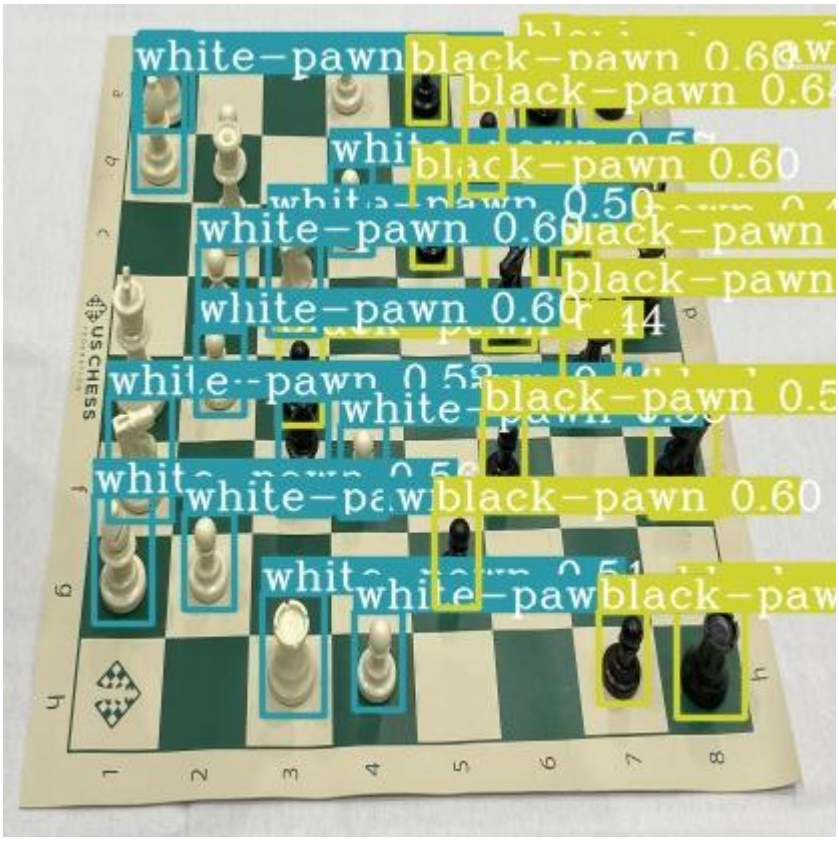
شکل 28. نتایج ارزیابی

با ران کردن کد زیر که infer.py است، فایل های سگمنت شده مهره های شطرنج همراه با برچسب های دقت تولید میشود و در مسیر `runs/inference/exp` ذخیره میشود. برای داده های تست، ترین و ارزیابی این کار را انجام میدهیم تا تمام فایل های سگمنت شده تصاویر تولید شوند.

```
!python tools/infer.py --weights /content/YOLOv6/runs/train/exp6/weights/best_ckpt.pt --source /content/YOLOv6/yolov6/dataset/images/test
```

```
--yaml /content/YOLOv6/yolov6/dataset/dataset.yaml --device 0
```

شکل 29. کد اینفرنس



شکل 30. تصویر نمونه سگمنت شده 1

شکل 31. تصویر نمونه سگمنت شده