بسم الله الرحمن الرحیم

گزارش پروژه نهایی درس هوش محاسباتی

موضوع:

Deep Convolutional Network (DCN)

شبکه پیچشی عمیق

استاد مربوطه:جناب آقای تورانی

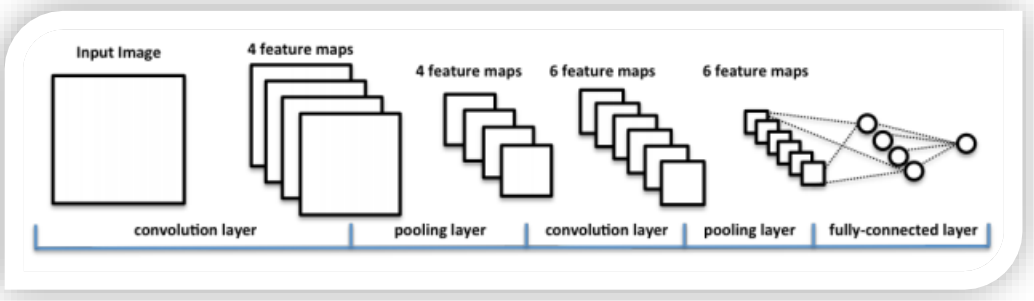
اعضای گروه 07 :

سیده ذکیه اسحاقی:960122680033

کوثر غلامعلی زاده:960122680041

شبکه عصبی پیچشی

شبکه عصبی پیچشی یک الگوریتم یادگیری عمیق است که تصویر ورودی را دریافت می‌کند و به هر یک از اشیا و جنبه‌های موجود در تصویر میزان اهمیت (وزن‌های قابل یادگیری و بایاس) تخصیص می‌دهد و قادر به متمایزسازی آن‌ها از یکدیگر است. در الگوریتم ConvNet در مقایسه با دیگر الگوریتم‌های دسته‌بندی به پیش‌پردازش (Pre Processing) کمتری نیاز است.



در سطح اول لایه کانولوشن قرار دارد که با استفاده از kernel های متنوع میتواند ویژگی های جدیدی را از تصویر استخراج کند. به دنبال آن عملیات pooling انجام می شود که وظیفه آن کاهش ابعاد و تعداد پارامترهای شبکه است. خروجی این لایه بعد از تبدیل به بردار یک بعدی به لایه شبکه اتصال کامل ارسال می شود، در این لایه از الگوریتم های رایج شبکه های عصبی استفاده می شود. بلاک convolution و pooling می تواند به دفعات تکرار شود و شبکه ای عمیق تر ساخته شود. تعداد لایه های Fully connected نیز توسط کاربر تعیین می شود.

مجموعه داده

مجموعه داده ها به سه دسته تقسیم می شوند:

مجموعه ی Training Data برای هدایت پروسه آموزش به کار گرفته می شود ، برای به روز کردن وزن های شبکه عصبی به هنگام آموزش.

مجموعه ی Validation Dataبرای مونیتور کردن (نظارت کردن)کیفیت مدل شبکه عصبی از سیستم به هنگام فرآیند یادگیری و تعیین شرط توقف یادگیری برای پروسه ی training استفاده می شود.

مجموعه ی Test Data مستقلا برای تعیین کیفیت نهایی شبکه ی آموزش دیده شده از لحاظ دقت و قابلیت های تعمیم سازی Generalizationسیستم اصلی استفاده می شود.

در این پروژه ما دیتاست مورد نظر را از سایت Kaggle که یک سایت معتبر در زمینه داده است دانلود کردیم که لینک آن را در زیر آورده ایم:

https://www.kaggle.com/ashishjangra27/gender-recognition-200k-images-celeba

این دیتاست،نسخه ای از دیتاست Celeba است که تصاویر سلبریتی های جهان در آن است.طراحان این دیتاست لیبل های جنسیت عکس‌ها را تهیه کرده و به عنوان یک دیتاست برا تشخیص جنسیت ارائه داده اند.

این دیتاست دارای ۲۰۰ هزار عکس است که ما ۵۰ هزار عدد از عکس‌های آن را استفاده کردیم.چون هدف ما آموزش شبکه عصبی ساخته شده است به طوری بتواند حداقل دقت ۹۰ درصد را در مرحله تست بگیرد و با توجه به اندازه شبکه ۵۰۰۰۰ داده کافی است.

پس ۲۵۰۰۰ هزار عکس مرد و ۲۵۰۰۰ عکس زن از از این دیتاست را انتخاب می کنیم.

❖ توضیحات کد:

- کتابخانه های torch , matplotlib.pyplot , numpy و os را import می کنیم.

```
import torch
from tqdm import tqdm
from PIL import Image
from torch import optim
from torch.nn import Sequential,Linear,Sigmoid,BCELoss,MSELoss,Tanh,Co
nv2d,MaxPool2d,ReLU,Flatten,Dropout
import  numpy as np
from torchsummary import summary
import os
import matplotlib.pyplot as plt
```

- Data Set مورد نظر از سایت kaggle دانلود می کنیم و سپس با دستور unzip فایل را از حالت فشرده خارج می کنیم

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

!kaggle datasets download -d ashishjangra27/gender-recognition-200k-
images-celeba
!unzip -x /content/gender-recognition-200k-images-celeba.zip
```

- توسط تابع `listdir` کتابخانه `os` لیست اسم های`female`در مسیر زیر را بدست می آوریم و 2500 عکس را با سایز 64 در 64 میخوانیم.(علت انتخاب سایز 64 بهبود سرعت `train` بود به طوری که کیفیت تصاویر هم مطلوب باشد) ، سپس عکس را به آرایه `numpy` و `shape (3, 64, 64)` تبدیل می کنیم و در نهایت به آرایه `f` اضافه می کنیم.

```
files=os.listdir("/content/Dataset/Train/Female")
f=[]
for file in files[:25000]:
    img=Image.open("/content/Dataset/Train/Female/"+file).resize((64,6
4))
```

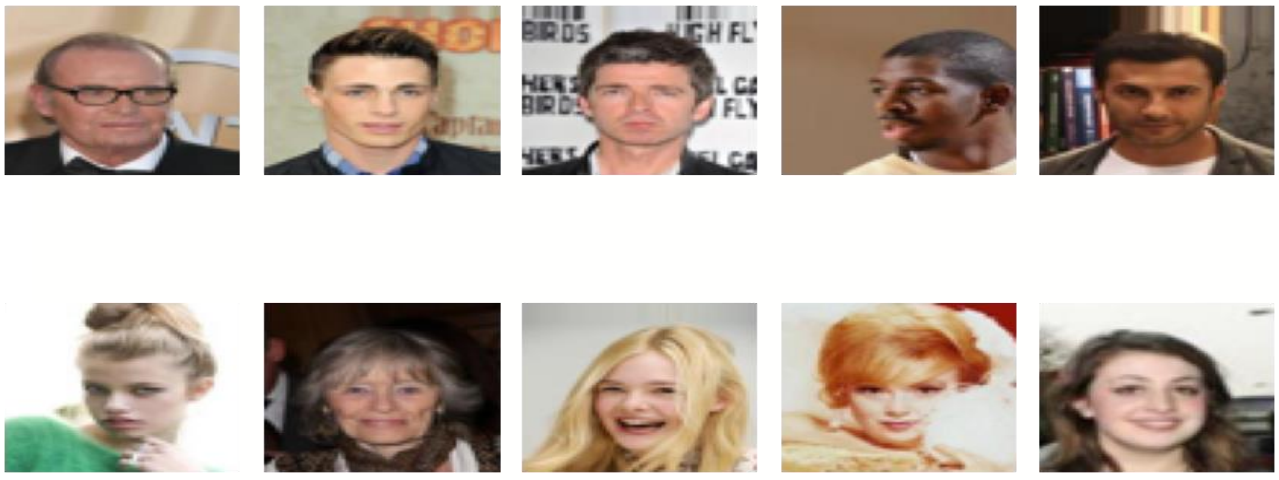
```
    img=np.array(img).reshape((3,64,64))
    f.append(img)
```

- تمام مراحل را برای داده های male نیز تکرار می کنیم.

```
m=[]
files=os.listdir("/content/Dataset/Train/Male")
for file in files[:25000]:
    img=Image.open("/content/Dataset/Train/Male/"+file).resize((64,64)
)
    img=np.array(img).reshape((3,64,64))
    m.append(img)
```

- نمایش پنج عکس از دسته female و پنج عکس از دسته male

```
tmp=m[:5]+f[:5]
plt.figure(figsize=(15,10))
for j in range(1,11):
  plt.subplot(2,5,j)
  plt.axis("off")
  plt.imshow(tmp[j-1].reshape((64,64,3)))

plt.subplots_adjust(hspace=0, wspace=0.1)
plt.show()
```



- دسته بندی داده به سه دسته test و train, validaton

60درصد از کل داده مربوط به دسته male را به train و 20در صد را برای validation و 20 درصد باقی مانده را برای فاز test درنظر می گیریم.

و هر کدام را به ترتیب در x_train و x_val و x_test نگه می داریم

و در y_train و y_val و y_test با همان ترتیب به ازای تمام داده های male،[0] و به ازای تمام داده های female [1]می نویسیم.

```
ix_train=int(len(m)*0.6)
ix_val=int(len(m)*0.2)
x_train=m[:ix_train]
x_val=m[ix_train:ix_train+ix_val]
x_test=m[ix_train+ix_val:]
y_train=[[0] for i in range(ix_train)]
y_val=[[0] for i in range(ix_val)]
y_test=[[0] for i in range(len(m)-ix_val-ix_train)]


ix_train=int(len(f)*0.6)
ix_val=int(len(f)*0.2)
x_train+=f[:ix_train]
x_val+=f[ix_train:ix_train+ix_val]
x_test+=f[ix_train+ix_val:]
y_train+=[[1] for i in range(ix_train)]
y_val+=[[1] for i in range(ix_val)]
y_test+=[[1] for i in range(len(m)-ix_val-ix_train)]
```

❖ transform

- داده ها را ابتدا به فرمت tensor و سپس به float تبدیل می کنیم.

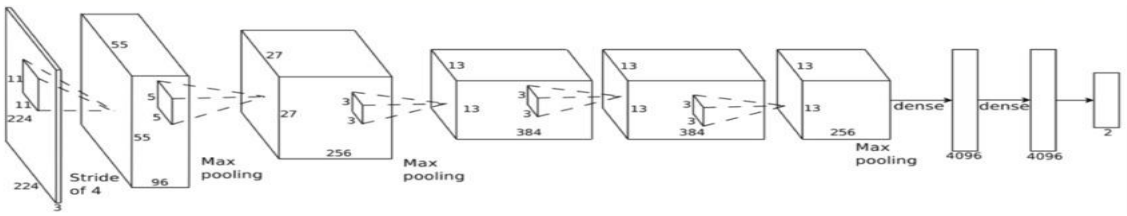
```
x_train=torch.tensor(x_train).float()
y_train=torch.tensor(y_train).float()
x_test=torch.tensor(x_test).float()
y_test=torch.tensor(y_test).float()
x_val=torch.tensor(x_val).float()
y_val=torch.tensor(y_val).float()
```

❖ scaling

- سپس طبق مطالعه ی نمونه های مختلف در اینترنت ، مقادیر را به 255 تقسیم کردیم تا محدوده پیکسل ها بین0تا 1 باشد و یادگیری شبکه سریع تر باشد.

```
x_train=x_train/255
x_val=x_val/255
x_test=x_test/255
```

❖ Model



- معماری شبکه را طبق تصویر زیر پیاده سازی کردیم با این تفاوت که لایه های کانوولشن بیشتری در نظر گرفتیم و ابعاد عکس را 64 در 64تنظیم کردیم و از فیلتر maxpool2Dبرای محاسبه ماکزیمم تعدادی پیکسل جهت کاهش سایز تصویر استفاده کردیم

  به دلیل مواجه شدن با مسئله over fit در ابتدا کار ، ملزم به استفاده ازdropout شدیم تا به کمک آن مشکل را حل کنیم .

```
model=Sequential(Conv2d(3,16,(3,3),stride=1,padding=1),ReLU(),MaxPool2d(kernel_size=(2
,2),stride=2),
                 Conv2d(16,32,(3,3),stride=1,padding=1),ReLU(),MaxPool2d(kernel_size=(2
,2),stride=2),
                 Conv2d(32,64,(3,3),stride=1,padding=1),ReLU(),MaxPool2d(kernel_size=(2
,2),stride=2),
                 Flatten(),
                 Dropout(0.5),
                 Linear(4096,128),ReLU(),Dropout(0.5),
                 Linear(128,1),Sigmoid())
model.to("cuda")

summary(model,input_size=(3,64,64))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 64, 64]             448
              ReLU-2          [-1, 16, 64, 64]               0
         MaxPool2d-3          [-1, 16, 32, 32]               0
            Conv2d-4          [-1, 32, 32, 32]           4,640
              ReLU-5          [-1, 32, 32, 32]               0
         MaxPool2d-6          [-1, 32, 16, 16]               0
            Conv2d-7          [-1, 64, 16, 16]          18,496
              ReLU-8          [-1, 64, 16, 16]               0
         MaxPool2d-9            [-1, 64, 8, 8]               0
           Flatten-10                 [-1, 4096]               0
           Dropout-11                 [-1, 4096]               0
            Linear-12                  [-1, 128]         524,416
             ReLU-13                  [-1, 128]               0
           Dropout-14                  [-1, 128]               0
            Linear-15                    [-1, 1]             129
           Sigmoid-16                    [-1, 1]               0
================================================================
Total params: 548,129
```

```
Trainable params: 548,129
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 2.03
Params size (MB): 2.09
Estimated Total Size (MB): 4.17
```

- طبق آموزش از Adam optimizer استفاده کردیم که پارامتر و مدل را می گیرد و وزن هارا آپدیت می کند.

```
optimizer = optim.Adam(model.parameters())
```

- تعداد ایپاک و بچ سایز را به ترتیب 64 و 50 در نظر گرفتیم.

```
BATCH_SIZE=64
EPOCHS=50
```

❖ Training

- برای classify به صورت باینری از BCELoss استفاده کردیم و توسط DataLoader داده ها را به اندازه بچ سایز ایندکس می دهد.

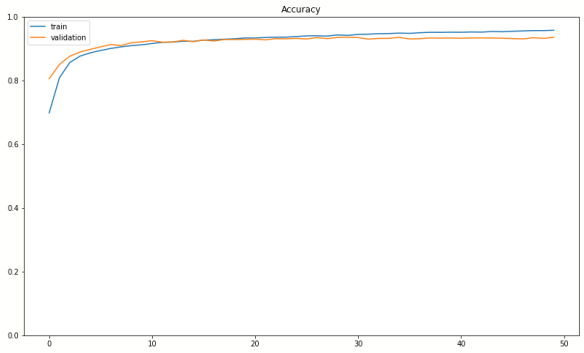
حالا باید مدل را آموزش دهیم.در پایان هر ایپاک دقت مدل را اندازه می گیریم تا بدانیم مدل در حال یادگیری است یا خیر.همچنین validation را هم حساب می کنیم تا بتوانیم رفتار مدل روی داده غیر آموزشی را هم بدانیم و accuracyداده ی بهترین وزن را ذخیره کنیم .

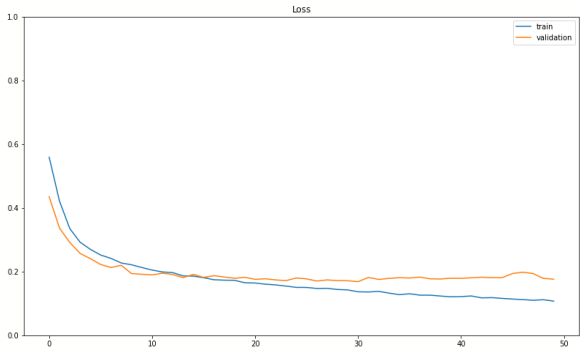
```
mse=BCELoss()
rn=np.arange(len(x_train))
dataloader=torch.utils.data.DataLoader(rn, BATCH_SIZE, True)
dataloader_val=torch.utils.data.DataLoader(np.arange(len(x_val)), 1000
, True)
total_losses_train=[]
total_losses_val=[]
total_accuracy_train=[]
total_accuracy_val=[]
maxi=-1
for epoch in tqdm(range(EPOCHS)):
  losses=[]
  accuracy_train=[]
  get_batch=iter(dataloader)
  for index in get_batch:
    batch_x=x_train[index].to("cuda")
    batch_y=y_train[index].to("cuda")
    pred=model(batch_x)
    #####update########
    model.train()##switch to trian
    optimizer.zero_grad()
    loss=mse(pred,batch_y)# loss
```

```
    loss.backward() #gradient
    optimizer.step() #
    losses.append(loss.item())

    ####computing accuracy of batch
    pred=pred>0.5
    target=batch_y
    correct=torch.sum(pred==target).item()
    accuracy=correct/len(batch_y)
    accuracy_train.append(accuracy)

  ####validation
  model.eval()###swtich to evaluation
  get_batch=iter(dataloader_val)
  loss_val=[]
  accuracy_val=[]
  for i in get_batch:
    pred=model(x_val[i].to("cuda"))
    loss_val.append(mse(pred,y_val[i].to("cuda")).item())
    pred=pred>0.5
    target=y_val[i].to("cuda")
    correct=torch.sum(pred==target).item()
    accuracy_val.append(correct/len(y_val[i]))
  accuracy_val=np.mean(accuracy_val)
  loss_val=np.mean(loss_val)
  if accuracy_val>maxi:
    maxi=accuracy_val
    torch.save(model,"model.pth")


  tqdm.write("Epoch"+str(epoch)+"  loss:"+str(np.mean(losses))+"  accu
racy_train:"+str(np.mean(accuracy_train))+"  loss_val:"+str(loss_val)+
"  val_accuracy:"+str(accuracy_val))
  total_losses_train.append(np.mean(losses))
  total_accuracy_train.append(np.mean(accuracy_train))
  total_losses_val.append(loss_val)
  total_accuracy_val.append(accuracy_val)
```

- **نمایش نمودار**

```
plt.figure(figsize=(30,8))
plt.subplot(1,2,1)
plt.ylim([0,1])
plt.plot(range(EPOCHS),total_accuracy_train,label="train")
plt.plot(range(EPOCHS),total_accuracy_val,label="validation")
plt.legend()
plt.title("Accuracy")

plt.subplot(1,2,2)
plt.ylim([0,1])
plt.plot(range(EPOCHS),total_losses_train,label="train")
plt.plot(range(EPOCHS),total_losses_val,label="validation")
plt.legend()
plt.title("Loss")

plt.show()
```





❖ Evaluating

- حال نوبت به ارزیابی مدل است تا مطمئن شویم مدل به درستی یاد گرفته ، داده را آموزشی را حفظ نکرده و اصطلاحا overfitرخ نداده است.پس بهترین وزن ذخیره شده را load کرده و داده تست را تخمین میزنیم و جواب واقعی آن مقایسه می کنیم

```
model=torch.load("model.pth")
dataloader_test=torch.utils.data.DataLoader(np.arange(len(x_test)), 10
00, True)
model.eval()
get_batch=iter(dataloader_test)
loss_test=[]
accuracy_test=[]
for i in get_batch:
    pred=model(x_test[i].to("cuda"))
    loss_test.append(mse(pred,y_test[i].to("cuda")).item())
    pred=pred>0.5
    target=y_test[i].to("cuda")
    correct=torch.sum(pred==target).item()
    accuracy_test.append(correct/len(y_test[i]))
accuracy_test=(np.mean(accuracy_test))
loss_test=np.mean(loss_test)
print("Accuracy test:",accuracy_test,"    loss_test:",loss_test)
```

```
model=torch.load("model.pth")
dataloader_test=torch.utils.data.DataLoader(np.arange(len(x_test)), 1000, True)
model.eval()
get_batch=iter(dataloader_test)
loss_test=[]
accuracy_test=[]
for i in get_batch:
    pred=model(x_test[i].to("cuda"))
    loss_test.append(mse(pred,y_test[i].to("cuda")).item())
    pred=pred>0.5
    target=y_test[i].to("cuda")
    correct=torch.sum(pred==target).item()
    accuracy_test.append(correct/len(y_test[i]))
accuracy_test=(np.mean(accuracy_test))
loss_test=np.mean(loss_test)
print("Accuracy test:",accuracy_test,"    loss_test:",loss_test)
```

```
Accuracy test: 0.9363000000000001     loss_test: 0.1963238537311554
```

همان طور که می بینیم ،دقت 93.6 درصد شده که دقت بالایی دارد.