سوال ۱-

اول از همه باید دیتاست را دانلود کنیم:

```
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth=GoogleAuth()
gauth.credentials=GoogleCredentials.get_application_default()
drive=GoogleDrive(gauth)
# https://drive.google.com/file/d/1KWnX3eMPJrzhsegi0LmyGUbUV5pqKw_R/view?usp=sharing
file_id='1KWnX3eMPJrzhsegi0LmyGUbUV5pqKw_R'
downloaded=drive.CreateFile({'id':file_id})
downloaded.FetchMetadata(fetch_all=True)
downloaded.GetContentFile(downloaded.metadata['title'])
!unzip ss_dataset.zip -d .
```

ساختار دیتاست مانند تمرین قبلی و مسئله semantic segmentation است.

```
train_image_root = '/content/train'
train_label_root = '/content/train_masks'
val_image_root = '/content/validation'
val_label_root = '/content/validation_masks'
test_image_root = '/content/test'
test_label_root = '/content/test_masks'
```

چهار تا فولدر ساختم، بخشی از دیتاست رو برای train ، بخشی برای validation و بخشی برای test در نظر گرفتم و لیبل های هر کدوم هم که خودشون یه تصویر هستند(که باید تبدیل به فایل تکست بشن) رو توی پوشه های جدا با اسمی مشابه گذاشتم، مثلا train_masks .

```
images = list()
labels = list()
for (dirpath, dirname, filenames) in os.walk(data_dir):
  for filename in filenames:
    img_name = filename.split('.')[0]
    if 'label' in img_name:
      labels.append(dirpath + f'/{filename}')
    else:
      images.append(dirpath + f'/{filename}')
```

اینجا اومدم آدرس همه عکسهای ورودی رو توی لیست images و عکسهای لیبل رو توی لیست labels ریختم.

```
images = sorted(images)
labels = sorted(labels, key=lambda x:x.replace("_label",""))
t = int(0.8*len(images))
v = int(0.8*t)
train_images = images[:v]
val_images = images[v:t]
test_images = images[t:]
train_labels = labels[:v]
val_labels = labels[v:t]
test_labels = labels[t:]
```

حالا برای اینکه بتونم آموزشی رو از تست و از validation جدا کنم، باید توی لیبل ها هم معادل همین ها باشن، پس باید ابتدا سورت کنیم و از اونجا که سورت در اینجا به صورت الفبایی هست، برای همین توی اسم فایلهای لیبل باید کلمه label_ رو در نظر نگیریم.

همونطور که از این کد مشخص هست، 0.2 دیتاست رو برای تست در نظر گرفتیم و از 0.8 باقیمانده، 0.2 ش رو برای validation در نظر گرفتیم و باقی برای آموزش.(بعدا میبینیم فرمت درخواستی مدل yolov7 هر سه نوع را میخواهد.)

```
for img_path in train_images:
  file_name = img_path.split('/')[-1].split('.')[0]
  img = Image.open(img_path)
  img = img.resize((256, 256))
  dir_name = img_path.split('/')[-2]
  img.save(train_image_root + '/' + dir_name + '_' + file_name + '.png', 'png')
for img_path in val_images:
  file_name = img_path.split('/')[-1].split('.')[0]
  img = Image.open(img_path)
  img = img.resize((256, 256))
  dir_name = img_path.split('/')[-2]
  img.save(val_image_root + '/' + dir_name + '_' + file_name + '.png', 'png')
```

در اینجا عکسها را ریسایز میکنیم و درون پوشه‌ی دیگری که خودمان ساختیم و بالاتر آدرسشون رو گذاشتم، میریزیم. برای validation و test هم همینطور.

```
for label_path in train_labels:
  file_name = label_path.split('/')[-1].split('.')[0].replace('_label', '')
  img = Image.open(label_path)
  img = img.resize((256, 256))
  dir_name = label_path.split('/')[-2]
  img.save(train_label_root + '/' + dir_name + '_' + file_name + '.png', 'png')
for label_path in val_labels:
  file_name = label_path.split('/')[-1].split('.')[0].replace('_label', '')
  img = Image.open(label_path)
  img = img.resize((256, 256))
  dir_name = label_path.split('/')[-2]
  img.save(val_label_root + '/' + dir_name + '_' + file_name + '.png', 'png')
```

برای لیبل ها هم همینطور.

فرمت درخواستی yolov7 برای دیتاست بدین گونه است که لیبل ها باید با عکسهای ورودی در یک پوشه باشند و هم اسم باشند، یکی فایل تکست و دیگری عکس.

```
for (dirpath, dirname, filenames) in os.walk(train_label_root):
  for filename in filenames:
    img_name = filename.split('.')[0]
    label_img = cv2.imread(dirpath+'/'+filename, cv2.IMREAD_UNCHANGED)
    generate_yolo_label(label_img, img_name, train_image_root)
```

حال باید لیبل های هر دسته را تبدیل به لیبلهای مناسب برای yolov7 کرده و در همان پوشه عکسهای ورودی ذخیره کنیم. یک تابع جدا به اسم generate_yolo_label نوشتم برای این کار.

```
def generate_yolo_label(img, img_name, folder_name):
  img[img < 50] = 0
  img[img >= 50] = 1
  contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
  f = open(folder_name + "/" + img_name + ".txt", "a")
  for contour in contours:
    x,y,w,h = cv2.boundingRect(contour)
    f.write(f"0 {x/img.shape[0]} {y/img.shape[1]} {w/img.shape[0]} {h/img.shape[1]}\n")
```

ورودیها:

Img: لیبل ورودی که عکس است.

Img_name: اسم فایل است. با همین نام ذخیره میشود.

Folder_name: اسم فولدری که میخواهیم، در آن ذخیره شود. اگر لیبل مربوط به آموزش باشد، در پوشه train ، اگر برای تست باشد، در پوشه test و در غیر اینصورت در پوشه validation ذخیره میشود.

در این تابع ابتدا تصویر را باینری میکنیم که بتوانیم آن را به findContours بدهیم. بعد از یافتن همه کانتورها، یک فایل تکست با همان اسم و در همان فولدر که در پارامترهای ورودی است، میسازیم و آن را در حالت a یعنی میخواهیم به آن متن اضافه کنیم، باز میکنیم.

برای هر کانتور، تابع boundingRect را روی آن صدا میزنیم تا مشخصات مستطیل محاط آن را بیابیم. این اطلاعات را با فرمت درخواستی yolov7 درون فایل مینویسیم، برای هر کانتور یک خط. Class_id را برابر با صفر قرار میدهیم، میتوانستیم ۱ هم بگذاریم، در آن صورت ۰ به معنای background بود.

در نهایت بخوام خلاصه بگم، سه تا فولدر به اسم های train,validation,test داریم حاوی عکسهای ورودی و لیبل هاشون که فایل تکست هستن.

```
!git clone https://github.com/WongKinYiu/yolov7.git
```

با این دستور در کولب، مدل yolov7 را دانلود میکنیم.

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

با این دستور کتابخونه های لازم رو دانلود میکنیم.

```
!python train.py --img-size 256 --batch-size 16 --epochs 50 --data data/custom.yaml --cfg cfg/training/yolov7.yaml
--weights 'yolov7.pt' --hyp data/hyp.scratch.p5.yaml --name yolov7-solarpanel
```

در نهایت از api ی که برامون فراهم شده، برای آموزش مدل استفاده میکنیم.

سایز عکس:۲۵۶ * ۲۵۶

سایز هر batch: ۱۶

تعداد epoch: ۵۰

data-- : برای این پارامتر، آدرس یک فایل yaml را میدهیم که اطلاعات دیتاست ما در آن ذخیره شده است. همچنین تعداد کلاس ها و اسمشان هم در این فایل قرار میدهیم.

اطلاعات درون فایل yaml :

```
nc:2
names=['solar panel','background']
train: path/to/train
val: path/to/validation
test: path/to/test
```

cfg-- : یکی از فایل های config که درون پوشه cfg بود را میدهیم، اطلاعات لایه ها و برخی اطلاعات دیگر در این فایل قرار میگیرد.

hyp-- : مقادیر hyper parameter ها در این فایل قرار میگیرد.

weights-- : از آنجا دیتاست محدودی داریم، حتما باید از انتقال یادگیری استفاده کنیم. چندین وزن اولیه داخل داکش بود که من اینو گذاشتم.

name-- : صرفا یک اسم برای این مدل آموزش دیده روی این دیتاست است.

سوال ۲-

۱. کاربرد ریاضی فیلتر کالمن برای پیش بینی ۱ پارامتری:

فیلتر کالمن یک الگوریتم ریاضی است که برای تخمین وضعیت یک سیستم بر اساس یک سری اندازه گیری های نویز استفاده می شود. به طور گسترده ای در سیستم های کنترل، پردازش سیگنال و برنامه های ردیابی استفاده می شود.

برای پیش بینی ۱ پارامتری، وضعیت سیستم با یک متغیر اسکالر منفرد مانند موقعیت یا سرعت نشان داده می شود. فیلتر کالمن وضعیت فعلی سیستم را بر اساس وضعیت قبلی و یک سری اندازه گیری های نویز تخمین می زند.

معادلات ریاضی فیلتر کالمن را می توان به صورت زیر نشان داد:

مرحله پیش بینی:

$$x\_hat(k|k-1) = F(k)*x\_hat(k-1|k-1)$$

$$P(k|k-1) = F(k)*P(k-1|k-1)*F(k)' + Q(k)$$

مرحله به روز رسانی:

$$K(k) = P(k|k-1)*H(k)'*(H(k)*P(k|k-1)*H(k)' + R(k))^{-1}$$

$$x\_hat(k|k) = x\_hat(k|k-1) + K(k)*(z(k) - H(k)*x\_hat(k|k-1))$$

$$P(k|k) = (I - K(k)*H(k))*P(k|k-1)$$

- $x\_hat(k|k-1)$ وضعیت پیش بینی شده سیستم در زمان k بر اساس پیش بینی قبلی در زمان $k-1$ است.

- $F(k)$ ماتریس انتقال حالت است که حالت قبلی را به حالت فعلی مرتبط می کند.

- $P(k|k-1)$ ماتریس کوواریانس پیش بینی شده است که نشان دهنده عدم قطعیت در حالت پیش بینی شده است.

- $Q(k)$ ماتریس کوواریانس نویز فرآیند است که نشان دهنده عدم قطعیت در مدل سیستم است.

- $K(k)$ ماتریس بهره کالمن است که وزن داده شده به به روز رسانی اندازه گیری را تعیین می کند.

- $z(k)$ اندازه گیری در زمان k است.

- $H(k)$ ماتریس اندازه گیری است که حالت را به اندازه گیری مرتبط می کند.

- $R(k)$ ماتریس کوواریانس نویز اندازه گیری است که نشان دهنده عدم قطعیت در اندازه گیری است.

۲. عملیات ریاضی کالمن به صورت ماتریس برای پیش بینی حرکت برچسب ها (با ۴ ویژگی) در شبکه ای که تصاویر را در ۷ گره جاسازی می کند:

با فرض اینکه وضعیت سیستم با یک بردار ۴ بعدی [x, y, vx, vy] نشان داده شود، که در آن x و y مختصات موقعیت و vx و vy مؤلفه های سرعت هستند و اندازه گیری ۷- است. بردار بعدی نشان دهنده تعبیه یک تصویر در ۷ گره است، معادلات فیلتر کالمن را می توان به صورت ماتریسی به صورت زیر نمایش داد:

مرحله پیش بینی:

$$x\_hat(k|k-1) = F*x\_hat(k-1|k-1)$$

$$P(k|k-1) = F*P(k-1|k-1)*F' + Q$$

مرحله به روز رسانی:

$$K(k) = P(k|k-1)*H'*(H*P(k|k-1)*H' + R)^{-1}$$

$$x\_hat(k|k) = x\_hat(k|k-1) + K(k)*(z(k) - H*x\_hat(k|k-1))$$

$$P(k|k) = (I - K(k)*H)*P(k|k-1)$$

- x_hat(k|k-1) حالت پیش بینی شده ۴ بعدی سیستم در زمان k بر اساس پیش بینی قبلی در زمان k-1 است.

- F ماتریس انتقال حالت ۴*۴ است که حالت قبلی را به حالت فعلی مرتبط می کند.

- P(k|k-1) ماتریس کوواریانس پیش بینی شده ۴*۴ است که نشان دهنده عدم قطعیت در حالت پیش بینی شده است.

- Q ماتریس کوواریانس نویز فرآیند ۴*۴ است که نشان دهنده عدم قطعیت در مدل سیستم است.

- K(k) ماتریس بهره کالمن ۷*۴ است که وزن داده شده به به روز رسانی اندازه گیری را تعیین می کند.

- z(k) اندازه گیری ۷ بعدی در زمان k است.

- H ماتریس اندازه گیری ۴*۷ است که حالت را به اندازه گیری مرتبط می کند.

- R ماتریس کوواریانس نویز اندازه گیری ۷*۷ است که نشان دهنده عدم قطعیت در اندازه گیری است.

۳. هدف از شبکه Deep در الگوریتم Deep SORT:

الگوریتم Deep SORT نوعی از الگوریتم SORT است که از یک شبکه عصبی عمیق برای استخراج و تطبیق ویژگی ها استفاده می کند. هدف شبکه عمیق استخراج ویژگی های سطح بالا از تشخیص اشیا و استفاده از آنها برای بهبود عملکرد ردیابی است.

در الگوریتم Deep SORT، تشخیص اشیا ابتدا از طریق یک شبکه عصبی عمیق عبور داده می شود تا تعبیه های ویژگی استخراج شود. سپس از تعبیه‌های ویژگی برای تطبیق تشخیص‌ها در فریم‌ها و مرتبط کردن آنها با آهنگ‌های موجود استفاده می‌شود.

شبکه عمیق بر روی مجموعه داده های مقیاس بزرگ آموزش داده شده است تا ویژگی های متمایز را یاد بگیرد که در برابر تغییرات ظاهری و شرایط نوری مقاوم هستند.

استفاده از یک شبکه عمیق در الگوریتم Deep SORT امکان عملکرد ردیابی دقیق تر و قوی تر، به ویژه در محیط های پیچیده با انسداد و پس زمینه های به هم ریخته را فراهم می کند. همچنین امکان پردازش سریع‌تر و ردیابی بلادرنگ را فراهم می‌کند و برای طیف گسترده‌ای از برنامه‌ها مناسب است.

سوال ۴-

۱. مزایا و محدودیت های معماری شبکه SiamFC در وظایف بینایی کامپیوتر:

مزایا:

- SiamFC یک معماری شبکه ساده و کارآمد است که می تواند به ردیابی شی در زمان RealTime دست یابد.
- نیازی به آموزش آنلاین یا بازآموزی ندارد و بردن آن در محیط های مختلف را آسان می کند.
- SiamFC می تواند تغییرات مقیاس و چرخش جسم مورد ردیابی را مدیریت کند.

- نشان داده شده است که در سناریوهای ردیابی چالش برانگیز، مانند انسداد و تاری حرکت، عملکرد خوبی دارد.

محدودیت ها:

- SiamFC محدود به ردیابی یک شی واحد در یک زمان است و نمی تواند ردیابی چندین شی را انجام دهد.

- ممکن است با ردیابی اجسام با ظاهر مشابه یا زمانی که جسم تحت تغییر شکل یا تغییر شکل قابل توجهی قرار می گیرد، مشکل داشته باشد.

۲. نحوه عملکرد SiamFC در ردیابی اشیاء بصری و اجزای اصلی آن:

معماری شبکه SiamFC (Siamese Fully Convolutional) از دو شبکه عصبی کانولوشنال (CNN) یکسان تشکیل شده است که وزن‌های مشترک دارند، جایی که یک شبکه ویژگی‌ها را از شی هدف در فریم اول استخراج می‌کند و شبکه دیگر ویژگی‌ها را از منطقه جستجو در فریم‌های بعدی استخراج می‌کند. سپس ویژگی‌های هر دو شبکه برای محاسبه امتیاز شباهت بین شی هدف و منطقه جستجو مقایسه می‌شوند.

اجزای اصلی SiamFC عبارتند از:

- دو CNN یکسان که وزن‌های مشترکی برای استخراج ویژگی‌ها از شی مورد نظر و منطقه جستجو دارند.

- یک لایه همبستگی متقابل که امتیاز شباهت بین شی هدف و منطقه جستجو را محاسبه می کند.

- یک لایه رگرسیون که مکان شی هدف را در منطقه جستجو پیش بینی می کند.

۳. چالش های مربوط به ردیابی اشیا و نحوه حل آنها توسط SiamFC:

چالش‌های ردیابی شی شامل تغییرات در ظاهر شی، شرایط نوری، انسداد و تاری حرکت است. SiamFC با استفاده از معماری سیامی که شی مورد نظر را با منطقه جستجو در فریم‌های بعدی مقایسه می‌کند، این چالش‌ها را برطرف می‌کند و آن را قادر می‌سازد علی‌رغم تغییر در ظاهر یا شرایط نور، شی را ردیابی کند. استفاده از شبکه‌های کاملاً کانولوشن به آن اجازه می‌دهد تا تغییرات مقیاس و چرخش را مدیریت کند، در حالی که لایه همبستگی متقابل آن را قادر می‌سازد تا انسداد و تاری حرکت را مدیریت کند.

۴. چگونه مفهوم معماری سیامی فراتر از ردیابی شی گسترش می یابد و سایر وظایف بینایی کامپیوتری می توانند از آن بهره مند شوند:

معماری سیامی یک رویکرد قدرتمند برای وظایف مختلف بینایی کامپیوتری است که شامل تطبیق شباهت است. یکی از این وظایف، یادگیری تک شات است، که شامل یادگیری تشخیص اشیاء جدید تنها با یک مثال است. از شبکه‌های سیامی می‌توان برای مقایسه ویژگی‌های شی جدید با اشیایی که قبلاً دیده شده‌اند، استفاده کرد و امکان تشخیص دقیق با حداقل داده‌های آموزشی را فراهم می‌کند. سایر وظایف بینایی رایانه ای که می توانند از معماری سیامی بهره مند شوند عبارتند از تأیید چهره، بازیابی تصویر و تشخیص اشیا.