**Name: Tahsin Reasat**

**VUID: Reasatt**

**Train.py**

**from** glob **import** glob

**import** os

**from** albumentations **import** (

Compose, Resize, Normalize, RandomBrightnessContrast,

HorizontalFlip,RandomRotate90,RandomCrop,

CenterCrop

)

**import** albumentations.pytorch **as** albu\_torch

**import** sys

sys.path.insert(1,**r'..\utility'**)

sys.path.insert(1,**r'..\models'**)

**from** dataloader **import** ISIC\_Dataset

**from** logger **import** Logger

**from** loss **import** bceWithSoftmax

**from** torch.utils.data **import** DataLoader

**from** models **import** ResNet18, ResNet50, DPN92

**import** torch.optim **as** optim

**import** torch

**import** time

**import** argparse

**import** numpy **as** np

**import** pickle

**import** pandas **as** pd

**from** metrics **import** get\_acc,get\_recall

device = torch.device(**"cuda:0" if** torch.cuda.is\_available() **else "cpu"**)

TIME\_STAMP=time.strftime(**'%Y-%m-%d-%H-%M-%S'**)

parser=argparse.ArgumentParser()

parser.add\_argument(**'--dir\_project'**, help=**'project directory'**, default=**r'..'**)

parser.add\_argument(**'--dir\_lf'**, help=**'directory large files'**,default=**r'D:\Data\cs-8395-dl'**)

parser.add\_argument(**'--folderData'**, help=**'data directory'**, default=**'assignment2\_data'**)

parser.add\_argument(**'--encoder'**,help=**'encoder'**,default=**'resnet18'**)

parser.add\_argument(**'--lr'**, help=**'learning rate'**, type=float, default=0.001)

parser.add\_argument(**'--batchSize'**, help=**'batch size'**, type=int, default=32)

parser.add\_argument(**'--epoch'**, help=**'epoch'**, type=int, default=400)

parser.add\_argument(**'--resume\_from'**, help=**'filepath to resume training'**)

parser.add\_argument(**'--bottleneckFeatures'**, help=**'bottleneck the encoder Features'**, type=int, default=1)

parser.add\_argument(**'--overrideLR'**, help=**'override LR from resumed network'**, type=int, default=1)

parser.add\_argument(**'--brightness'**,nargs=**'+'**, type=float)

parser.add\_argument(**'--contrast'**,nargs=**'+'**, type=float)

parser.add\_argument(**'--cropSize'**, type=int)

parser.add\_argument(**'--resize'**, type=int)

parser.add\_argument(**'--to\_ram'**,type=int, default=0)

parser.add\_argument(**'--loss\_weights'**,nargs=**'+'**, type=float)

args=parser.parse\_args()

*# setting up directories*

DIR\_LF = args.dir\_lf*#r'D:\Data\cs-8395-dl'*

dir\_data = os.path.join(DIR\_LF,args.folderData) *#os.path.join(DIR\_LF,'assignment1\_data')*

dir\_model = os.path.join(args.dir\_lf, **'model'**,TIME\_STAMP)

dir\_history = os.path.join(args.dir\_project, **'history'**)

dir\_log = os.path.join(args.dir\_project, **'log'**)

dir\_config = os.path.join(args.dir\_project, **'config'**)

**if** os.path.exists(dir\_history) **is False**:

os.mkdir(dir\_history)

**if** os.path.exists(dir\_log) **is False**:

os.mkdir(dir\_log)

**if** os.path.exists(dir\_config) **is False**:

os.mkdir(dir\_config)

**if** os.path.exists(os.path.join(args.dir\_lf, **'model'**)) **is False**:

os.mkdir(os.path.join(args.dir\_lf, **'model'**))

filepath\_hist = os.path.join(dir\_history, **'{}.bin'**.format(TIME\_STAMP))

filepath\_log = os.path.join(dir\_log, **'{}.log'**.format(TIME\_STAMP))

filepath\_cfg = os.path.join(dir\_config, **'{}.cfg'**.format(TIME\_STAMP))

sys.stdout = Logger(filepath\_log)

print(TIME\_STAMP)

print(os.path.basename(\_\_file\_\_))

config=vars(args)

config\_ls=sorted(list(config.items()))

print(**'--------------------------------------------------------------------------------------------------------------------'**)

**for** item **in** config\_ls:

print(**'{}: {}'**.format(item[0],item[1]))

print(**'--------------------------------------------------------------------------------------------------------------------'**)

**with** open(filepath\_cfg, **'w'**) **as** file:

**for** item **in** config\_ls:

file.write(**'{}: {}\n'**.format(item[0], item[1]))

**if** os.path.exists(dir\_model)==0:

print(**'creating directory to save model at {}'**.format(dir\_model))

os.mkdir(dir\_model)

filepath\_model\_best = os.path.join(dir\_model, **'{}\_{}\_best.pt'**.format(TIME\_STAMP, args.encoder)) *##*

filepath\_model\_latest = os.path.join(dir\_model, **'{}\_{}\_latest.pt'**.format(TIME\_STAMP, args.encoder)) *##*

dir\_data\_train = os.path.join(dir\_data, **'train'**)

dir\_data\_test = os.path.join(dir\_data, **'test'**)

*# get train filenames*

filepath\_train\_label = os.path.join(dir\_data, **'labels'**,**'Train\_labels.csv'**)

df\_train = pd.read\_csv(filepath\_train\_label)

df\_train.set\_index(**'image'**,inplace=**True**)

files\_train = df\_train.index.values

labels\_train\_one\_hot=[df\_train.loc[flname].values **for** flname **in** files\_train]

labels\_train\_cat = [np.argmax(label) **for** label **in** labels\_train\_one\_hot]

*# get test filenames*

filepath\_test\_label = os.path.join(dir\_data, **'labels'**,**'Test\_labels.csv'**)

df\_test = pd.read\_csv(filepath\_test\_label)

df\_test.set\_index(**'image'**,inplace=**True**)

files\_test = df\_test.index.values

labels\_test\_one\_hot=[df\_test.loc[flname].values **for** flname **in** files\_test]

labels\_test\_cat = [np.argmax(label) **for** label **in** labels\_test\_one\_hot]

*# Dataloader Parameters*

aug ={

**'train'**: Compose([

HorizontalFlip(),

RandomRotate90(),

RandomBrightnessContrast(

brightness\_limit=args.brightness,

contrast\_limit=args.contrast,

),

RandomCrop(args.cropSize, args.cropSize, p=0.5),

Resize(args.resize,args.resize),

Normalize(),

albu\_torch.ToTensorV2()

]),

**'valid'**: Compose([

Resize(args.resize,args.resize),

Normalize(),

albu\_torch.ToTensorV2()

])

}

BATCH\_SIZE=args.batchSize

LR = args.lr

EPOCH=args.epoch

Dataset\_train = ISIC\_Dataset(dir\_data=dir\_data\_train, files=df\_train.index.values, label\_cat=labels\_train\_cat, transform=aug[**'train'**])

loader\_train=DataLoader(Dataset\_train,batch\_size=BATCH\_SIZE, shuffle=**True**)

print(**'train samples {}'**.format(len(Dataset\_train)))

Dataset\_valid = ISIC\_Dataset(dir\_data=dir\_data\_test, files=df\_test.index.values, label\_cat=labels\_test\_cat, transform=aug[**'valid'**])

loader\_valid=DataLoader(Dataset\_valid,batch\_size=BATCH\_SIZE, shuffle=**False**)

print(**'validation samples {}'**.format(len(Dataset\_valid)))

*# Model*

**if** args.encoder == **'resnet18'**:

model = ResNet18(pretrained=**True**, bottleneckFeatures=args.bottleneckFeatures).to(device)

**if** args.encoder == **'resnet50'**:

model = ResNet50(pretrained=**True**, bottleneckFeatures=args.bottleneckFeatures).to(device)

**if** args.encoder == **'dpn92'**:

model = DPN92().to(device)

*# print(model)*

*# Optimizer*

optimizer = optim.Adam(model.parameters(), lr=LR, betas=(0.9, 0.999), eps=1e-08, weight\_decay=0,

amsgrad=**False**)

*# Train*

**if** args.resume\_from **is not None**:

*# Resume?*

print(**'resuming training from {}'**.format(args.resume\_from))

train\_states = torch.load(args.resume\_from)

model.load\_state\_dict(train\_states[**'model\_state\_dict'**])

**if** args.overrideLR==0:

optimizer.load\_state\_dict(train\_states[**'optimizer\_state\_dict'**])

epoch\_range = np.arange(train\_states[**'epoch'**]+1, train\_states[**'epoch'**]+1+EPOCH)

**else**:

train\_states = {

**'epoch'**: 0,

**'model\_state\_dict'**: model.state\_dict(),

**'optimizer\_state\_dict'**: optimizer.state\_dict(),

**'model\_save\_criteria'**: np.inf,

}

epoch\_range = np.arange(1,EPOCH+1)

loss\_train=[]

loss\_valid=[]

acc\_train = []

acc\_valid=[]

recall\_macro\_valid = []

recall\_micro\_valid = []

compute\_loss = bceWithSoftmax(weights=args.loss\_weights)

**for** epoch **in** epoch\_range:

running\_loss = 0

running\_acc = 0

model.train()

**for** i, sample **in** enumerate(loader\_train):

optimizer.zero\_grad()

img = sample[0].to(device)

target = sample[1].to(device)

output = model(img)

*# print(target,output)*

loss = compute\_loss(output,target)

loss.backward()

optimizer.step()

running\_loss += loss.item()

running\_acc += get\_acc(target.cpu(),output.cpu())

mean\_loss = running\_loss / (i + 1)

mean\_acc = running\_acc / (i + 1)

print(**'train >>> epoch: {}/{}, batch: {}/{}, mean\_loss: {:.4f}, mean\_acc: {:.4f}'**.format(

epoch,

epoch\_range[-1],

i+1,

len(loader\_train),

mean\_loss,

mean\_acc

))

loss\_train.append(mean\_loss)

acc\_train.append(mean\_acc)

model.eval()

running\_loss = 0

output\_all=torch.FloatTensor([])

target\_all=torch.FloatTensor([])

**with** torch.no\_grad():

**for** i, sample **in** enumerate(loader\_valid):

img = sample[0].to(device)

target = sample[1].to(device)

output = model(img)

output\_all=torch.cat((output\_all,output.float().cpu()),dim=0)

target\_all=torch.cat((target\_all,target.float().cpu()),dim=0)

loss = compute\_loss(output,target)

running\_loss += loss.item()

running\_acc += get\_acc(target.cpu(),output.cpu())

mean\_loss = running\_loss / (i + 1)

recall\_macro = get\_recall(target\_all, output\_all, average=**'macro'**)

recall\_micro = get\_recall(target\_all, output\_all, average=**'micro'**)

mean\_acc=get\_acc(target\_all, output\_all)

acc\_valid.append(mean\_acc)

print(**'valid >>> epoch: {}/{}, mean\_loss: {:.4f}, mean\_acc: {:.4f}'**.format(

epoch,

epoch\_range[-1],

mean\_loss,

mean\_acc

))

print(**'recall\_micro\_valid: {:.4f}, recall\_macro\_valid: {:4f}'**.format(recall\_micro, recall\_macro))

loss\_valid.append(mean\_loss)

recall\_macro\_valid.append(recall\_macro)

recall\_micro\_valid.append(recall\_micro)

*# save train history*

log = {

**'loss\_train'**:loss\_train,

**'loss\_valid'**:loss\_valid,

**'acc\_train'**: acc\_train,

**'acc\_valid'**: acc\_valid,

**'recall\_micro\_valid'**:recall\_micro\_valid,

**'recall\_macro\_valid'**:recall\_macro\_valid

}

**with** open(filepath\_hist, **'wb'**) **as** pfile:

pickle.dump(log, pfile)

*# save best model*

**if** mean\_loss<train\_states[**'model\_save\_criteria'**]:

print(**'criteria decreased from {:.4f} to {:.4f}, saving best model at {}'**.format(train\_states[**'model\_save\_criteria'**],

mean\_loss,

filepath\_model\_best))

train\_states = {

**'epoch'**: epoch,

**'model\_state\_dict'**: model.state\_dict(),

**'optimizer\_state\_dict'**: optimizer.state\_dict(),

**'model\_save\_criteria'**: mean\_loss,

}

torch.save(train\_states, filepath\_model\_best)

*# save latest model*

train\_states = {

**'epoch'**: epoch,

**'model\_state\_dict'**: model.state\_dict(),

**'optimizer\_state\_dict'**: optimizer.state\_dict(),

**'model\_save\_criteria'**: mean\_loss,

}

torch.save(train\_states, filepath\_model\_latest)

print(TIME\_STAMP)

**Test.py**

**from** glob **import** glob

**import** os

**from** albumentations **import** (

Compose, Resize, Normalize, RandomBrightnessContrast, HorizontalFlip,

CenterCrop

)

**import** albumentations.pytorch **as** albu\_torch

**import** sys

sys.path.insert(1,**r'..\utility'**)

sys.path.insert(1,**r'..\models'**)

**from** dataloader **import** ISIC\_Dataset

**from** logger **import** Logger

**from** loss **import** bceWithSoftmax

**from** torch.utils.data **import** DataLoader

**from** models **import** ResNet18, ResNet50, DPN92

**import** torch.optim **as** optim

**import** torch

**import** time

**import** argparse

**import** numpy **as** np

**import** pickle

**import** pandas **as** pd

**from** metrics **import** get\_acc,get\_recall,conf\_mat

**from** tqdm **import** tqdm

**import** matplotlib.pyplot **as** plt

**from** metrics **import** pretty\_plot\_confusion\_matrix

**from** pandas **import** DataFrame

device = torch.device(**"cuda:0" if** torch.cuda.is\_available() **else "cpu"**)

parser=argparse.ArgumentParser()

parser.add\_argument(**'--filepath'**, help=**'project directory'**, required=**True**)

parser.add\_argument(**'--encoder'**,help=**'encoder'**,default=**'dpn92'**)

parser.add\_argument(**'--batchSize'**, help=**'batch size'**, type=int, default=32)

parser.add\_argument(**'--load\_from'**, help=**'filepath to load model'**, default=**r'D:\Data\cs-8395-dl\model\2020-02-10-22-06-20\2020-02-10-22-06-20\_dpn92\_best.pt'**)

parser.add\_argument(**'--resize'**, type=int, default=256)

args=parser.parse\_args()

*# setting up directories*

BATCH\_SIZE=args.batchSize

dir\_data\_part = os.path.dirname(args.filepath)

files = os.path.basename(args.filepath).split(**'.'**)[0]

*# Dataloader Parameters*

aug =Compose([

Resize(args.resize,args.resize),

Normalize(),

albu\_torch.ToTensorV2()

])

Dataset\_valid = ISIC\_Dataset(dir\_data=dir\_data\_part, files=[files], label\_cat=[ 0 ], do\_cc=**True**,transform=aug)

loader\_valid=DataLoader(Dataset\_valid,batch\_size=BATCH\_SIZE, shuffle=**False**)

*# print('validation samples {}'.format(len(Dataset\_valid)))*

*# Model*

**if** args.encoder == **'resnet18'**:

model = ResNet18(pretrained=**False**, bottleneckFeatures=0).to(device)

**if** args.encoder == **'resnet50'**:

model = ResNet50(pretrained=**False**, bottleneckFeatures=0).to(device)

**if** args.encoder == **'dpn92'**:

model = DPN92().to(device)

*# print(model)*

*# print('loading model from {}'.format(args.load\_from))*

train\_states = torch.load(args.load\_from)

*# print('loading model from epoch ', train\_states['epoch'])*

model.load\_state\_dict(train\_states[**'model\_state\_dict'**])

model.eval()

**with** torch.no\_grad():

**for** sample **in** loader\_valid:

img = sample[0].to(device)

target = sample[1].to(device)

output = model(img)

output=torch.softmax(output,dim=1).detach().cpu().numpy()

print(output.argmax())

**Dataloader.py**

**from** torch.utils.data **import** Dataset, DataLoader

**from** PIL **import** Image

**from** tqdm **import** tqdm

**import** os

**import** random

**import** numpy **as** np

**from** skimage **import** io

**import** torch

**from** glob **import** glob

**from** skimage **import** io

**from** scipy.ndimage **import** gaussian\_filter

**from** albumentations **import** (

Resize,HorizontalFlip,

Compose,

Normalize,

RandomBrightnessContrast,

CenterCrop,

)

**import** albumentations.pytorch **as** albu\_torch

**import** pandas **as** pd

**from** matplotlib **import** pyplot **as** plt

**from** sampler **import** BalancedBatchSampler

**import** sys

sys.path.insert(1,**r'..\preprocessing'**)

**from** color\_constancy **import** ColorConstancy

**def** reverse\_transform(img\_t,mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]):

img\_r = np.array(img\_t)

img\_r = img\_r.transpose([1,2,0])

img\_r = img\_r\*std+mean

img\_r \*=255

img\_r=img\_r.astype(np.uint8)

img\_r = np.squeeze(img\_r)

**return** img\_r

**class** ISIC\_Dataset(Dataset):

**def** \_\_init\_\_(self, dir\_data, files, label\_cat, to\_ram = **False**, transform=**None**, do\_cc=**False**):

self.dir\_data = dir\_data

self.transform = transform

self.files = files

self.to\_ram = to\_ram

self.image\_all=[]

self.label\_cat=label\_cat

self.do\_cc = do\_cc

self.color\_constancy = ColorConstancy(verbose=**False**, thresh\_bg=**None**)

**if** self.to\_ram:

print(**'loading images to RAM'**)

**for** file **in** tqdm(self.files):

*# file = self.files[idx]*

path\_img = os.path.join(self.dir\_data, file+**'.jpg'**)

image = io.imread(path\_img)

**if** self.do\_cc:

image=self.color\_constancy.comp(image)

*# print(image.shape)*

self.image\_all.append(image)

**def** \_\_len\_\_(self):

size = len(self.files)

**return** size

**def** \_\_getitem\_\_(self, idx):

**if** self.to\_ram:

image=self.image\_all[idx]

**else**:

*# print(self.files[idx])*

path\_img = os.path.join(self.dir\_data, self.files[idx] + **'.jpg'**)

image = io.imread(path\_img)

target=self.label\_cat[idx]

*# print(self.files[idx],image.shape)*

transformed=self.transform(image=image)

img = transformed[**'image'**]

**return** img,torch.tensor(target)

**Loss.py**

**import** torch

**def** bceWithSoftmax(weights=**None**):

*# i didn't the like the official name of the loss hence the function*

**if** weights **is not None**:

weights = torch.FloatTensor(weights).cuda()

**return** torch.nn.CrossEntropyLoss(weights)

**if** \_\_name\_\_==**'\_\_main\_\_'**:

loss = bceWithSoftmax()

input = torch.randn(2, 3, requires\_grad=**True**)

target = torch.empty(2, dtype=torch.long).random\_(3)

print(**'input'**,input)

print(**'target'**,target)

input\_sm = torch.softmax(input,dim=1)

print(**'softmax'**,input\_sm)

print(-torch.log(input\_sm))

print(-torch.log(1-input\_sm))

output = loss(input, target)

print(output)

*# output.backward()*

**Sampler.py**

*# https://raw.githubusercontent.com/galatolofederico/pytorch-balanced-batch/master/sampler.py*

is\_torchvision\_installed = **True**

**try**:

**import** torchvision

**except**:

is\_torchvision\_installed = **False**

**import** torch.utils.data

**import** random

**import** torch

**class** BalancedBatchSampler(torch.utils.data.sampler.Sampler):

**def** \_\_init\_\_(self, dataset, labels=**None**,shuffle=**False**):

self.labels = labels

self.dataset = dict() *# keys are class labels, values are set of sample indices associated with each label*

self.balanced\_max = 0

self.shuffle = shuffle

*# Save all the indices for all the classes*

**for** idx **in** range(0, len(dataset)):

label = self.\_get\_label(dataset, idx)

**if** label **not in** self.dataset:

self.dataset[label] = list()

self.dataset[label].append(idx)

self.balanced\_max = len(self.dataset[label]) \

**if** len(self.dataset[label]) > self.balanced\_max **else** self.balanced\_max

*# Oversample the classes with fewer elements than the max*

**for** label **in** self.dataset:

**while** len(self.dataset[label]) < self.balanced\_max:

self.dataset[label].append(random.choice(self.dataset[label]))

self.keys = list(self.dataset.keys())

self.currentkey = 0 *# keeps track of which class should be sampled*

self.indices = [-1] \* len(self.keys) *# keeps track of number of samples per class*

print(**'balanced\_max: '**,self.balanced\_max)

print(**'number of samples in balanced dataset {}'**.format(self.balanced\_max\*len(self.keys)))

*# print(self.indices)*

**def** \_\_iter\_\_(self):

**if** self.shuffle:

print(**'shuffling dataset'**)

**for** label **in** self.dataset:

random.shuffle(self.dataset[label])

*# print(self.dataset)*

**while** self.indices[self.currentkey] < self.balanced\_max - 1:

self.indices[self.currentkey] += 1

**yield** self.dataset[self.keys[self.currentkey]][self.indices[self.currentkey]]

self.currentkey = (self.currentkey + 1) % len(self.keys) *# I geuss an assertion that currentkey stays between 0 and num\_class-1?*

self.indices = [-1] \* len(self.keys)

**def** \_get\_label(self, dataset, idx):

**if** self.labels **is not None**:

**return** self.labels[idx].item()

**else**:

*# Trying guessing*

dataset\_type = type(dataset)

**if** is\_torchvision\_installed **and** dataset\_type **is** torchvision.datasets.MNIST:

**return** dataset.train\_labels[idx].item()

**elif** is\_torchvision\_installed **and** dataset\_type **is** torchvision.datasets.ImageFolder:

**return** dataset.imgs[idx][1]

**else**:

**raise** Exception(**"You should pass the tensor of labels to the constructor as second argument"**)

**def** \_\_len\_\_(self):

**return** self.balanced\_max \* len(self.keys)

**Metrics.py**

**from** sklearn.metrics **import** accuracy\_score

**from** sklearn.metrics **import** recall\_score

**from** sklearn.metrics **import** confusion\_matrix

**import** torch

*#imports*

**from** pandas **import** DataFrame

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** matplotlib.font\_manager **as** fm

**from** matplotlib.collections **import** QuadMesh

**import** seaborn **as** sn

**def** get\_acc(target,output):

output\_sm = torch.softmax(output, dim=1)

output\_cat = output\_sm.argmax(dim=1)

**return** accuracy\_score(target,output\_cat)

**def** get\_recall(target,output,average):

output\_sm = torch.softmax(output, dim=1)

output\_cat = output\_sm.argmax(dim=1)

**return** recall\_score(target,output\_cat,average=average)

**def** conf\_mat(target, output, labels=**None**):

output\_sm = torch.softmax(output, dim=1)

output\_cat = output\_sm.argmax(dim=1)

target=target.cpu().numpy()

output\_cat = output\_cat.cpu().numpy()

*# print(output\_cat)*

mat=confusion\_matrix(target, output\_cat, labels=labels, sample\_weight=**None**)

**return** mat

**Color\_constancy.py**

**import** numpy **as** np

**from** skimage **import** io

**import** argparse

**import** math

**from** matplotlib **import** pyplot **as** plt

**class** ColorConstancy():

**def** \_\_init\_\_(self,verbose=**False**, thresh\_bg=**None**):

self.verbose = verbose

self.thresh\_bg = thresh\_bg

**def** thresh\_img(self,img,thresh):

red\_range = thresh[0]!=img[:,:,0]

green\_range = thresh[1]!=img[:,:,1]

blue\_range = thresh[2]!=img[:,:,2]

valid\_range = np.logical\_or(red\_range, green\_range, blue\_range)

**return** valid\_range

**def** color\_constancy(self,img,preserve\_range=**True**):

e = np.zeros([3])

**for** i **in** range(3):

x = img[:,:,i]

**if** self.thresh\_bg **is not None**:

x=x[x!=0]

e[i]=x.mean()

**if** self.verbose: print(**'channel means'**,e)

e=e/math.sqrt(sum(e\*e))

**if** self.verbose: print(**'illumination estimate'**,e)

d=1/(math.sqrt(3)\*e)

**if** self.verbose: print(**'correction coefficient'**,d)

*# print(d)*

img\_t= img\*d

**for** i **in** range(3):

**if** self.verbose:

print(**'transformed image channel {} max\min: {}\{}'**.format(

i+1,img\_t[:,:,i].max(),img\_t[:,:,i].min()))

**if** preserve\_range:

**if** self.verbose:

print(**'setting values above 255 to 255'**)

img\_t=img\_t.flatten()

img\_t[img\_t>255]=255

img\_t=img\_t.reshape(img.shape)

**return** img\_t.astype(np.uint8)

**def** compute\_cc(self,img,path\_skin=**None**):

**if** img.shape[2]>3:

img=img[:,:,:3]

**if** path\_skin **is not None**:

mask\_skin=io.imread(path\_skin)

mask\_skin = mask\_skin/mask\_skin.max()

**if** len(mask\_skin.shape)<3:

mask\_skin = np.repeat(mask\_skin[:, :, np.newaxis], 3, axis=2)

img = (img\*mask\_skin).astype(np.uint8)

**if** self.thresh\_bg **is not None**:

mask = self.thresh\_img(img,self.thresh)

mask = np.repeat(mask[:, :, np.newaxis], 3, axis=2)

img = img\*mask

img\_tx = self.color\_constancy(img)

**return** img\_tx