Face Recognition

face_detection = RetinaNet()
(ResNet18 + FPN)
face Recognition = SNFaceNetModel()
((SqueezeNet1.1)

Input_Img_size = 224

input_image = Image.open(opt.input_file_name)

retinanet = model.resnet18 (num_class=1)

model_checkpoint = torch.load(opt.face_detect_model,
~~map_location=lepo~~)

Remove keys related to DataParallel.

new_state_dict = OrderedDict()

for K, V in model_checkpoint['state_dict'].items():

name = K[7:]

new_state_dict[name] = V

retinanet.load_state_dict(new_state_dict)

If use_gpu:

retinanet = torch.nn.DataParallel(retinanet)

facenet = model.SNFaceNetModel()

OpenVINO:

classification, regression = iter(net.output) # face_detection outputs

output_blob_recog, output_blob_recog_gender = iter(net_recog.output)
face_recognition outputs

exec_net = plugin.load(network=net) # load the face detection network.

exec_net_recog = plugin.load(network=net_recog) # load face recognition ~~detection~~ network

(512, 64, 256, 256)

x = 512, —

gen_out = (512, 64) → Conv
+ BN
+ ReLU
+
gaffee (64, 2) →

So,
gen_out = [male, female]

x = (512, 128)

return x, gen_out
|| ||
[128] [2]

if FACE_ALIGN:

if FACE_ALIGN_POINTS = 68:

sp = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')

else sp = ———— 5_face ————)

Finding embeddings of reference images:

folder_list = sorted([f.name for f in os.scandir(opt.ref_images) ---])

refimg_list = [[] for _ in range(len(folder_list))]

refimg_encoded_list = [[] for - in range(len(folder_list))]

for i in range(len(folder_list)):

for name in files:

refimg_list[i].append(name)

i are all folders which contains 4 images (2/3) for that person

refimg_list [[roshan, roshan], [guru], [sunil, sunil]---]

running the iface recognition infer request for reference images

ref_image_encoded = exec_net_reg.infer(inputs = {input_blob: batch})

ref_image_encoded_list[i].append(ref_image_encoded)

running infer request for face recognition model:

face_image_encoded = exec_net_reg.infer(input = {input_blob: batch})

inference_image → detection + encoding

[encodings]

reference_images → resize_center_crop

(only one time when we start the script)

encoding (only encoding model)

(Not detection)

[encodings] ∀ ref images

↳ Calculate the dist b/w face_encodings & reference images encoding.

↳ predict the class (using min. dist).

---

train → Retinaned → (class, face_face)
→ local loss

⇒ SNFaceNet(·)

↳ triplet loss.

(4/5 samples from each class)

face_image_encoded = ~~face_image_encoded~~
l2_norm(face_image_encoded[output_blob_recog]).

face_image_encoded *= 10

face_img_encoded_list.append(face_image_encoded).

for id in range(len(folder_list)):
    distances = []
    for j in range(len(ref_images_list[id])):
        dist = l2_dist(face_image_encoded, face_img_encoded_list[id][j]).
        distances.append(min_dist.detach().numpy().item())

top_5_idx = np.argsort(distances)[:5]

facenet_with_gender_classification_train

model = FaceNetModel( ) → resNet-34

gen_model = SNFaceNetModel( ) → Squeezenet1.1

face_recog

- Face-Recognition- summary

① Face detection + Face Recognition

↓ ↓

RetinaNet SNFaceNet

↓ ↓

(Base model is ResNet-18) (Base model is Squeezenet)

+ ↓

(FPN) Generate 128 length embeddings

+ +

focal loss. triplet loss fn is used.

② use OpenVino. (for inference).

③ x_q → inference image

we kept a set (list of) of Reference images

- inference image (x_{eq}) → detection (RetinaNet) + encodings (SNFaceNet)
- reference images → (resize → CenterCrop) preprocessing

↓

encoding (ONLY encoding no detection).

↓

[embeddings] of reference images.

Is then calculate the ~~distance~~ (L2 Norm) distance (L2 Norm) b/w face encoding (x_q) & reference images encodings.

and, Compute the distance from all images in reference-img list.

↳ take the min distance.

↳ min dist should be less than a threshold.

↳ I used face landmarks using dlib to calculate the distance b/w eyes (eg b/w ref & reference images) for fine tunning the model:

↳ Based on the previous threshold & this threshold final class Name was predicted.

↳ Name was taken from ref. images folder Name.

• My Role:

(1) RetinaNet - training setup. (& test setup) & integrated it into pipeline.

(2) Understand the ArcFaceNet model & integrated the embeddings to pipeline.

· face Recognition

① ODI (RetinaNet)

② embeddings (SNFaceNet())

· vms (license plate)

① ODI (Retinanet)

(WPOD) Network

② Character Segmentating (RetinaNet)

+ Character classification (CNN)

· note

① ODI → YOLO

- (C.N, UN no., hazard symbols, seals etc)

(Retrained on new dataset).

② UN no. ~~segment~~ character Segmentation

(w/ RetinaNet).

+ UN no. character classification.

↳ post-processing

① K-means clustering.

② take the max. occurance of a character in Real time. (inference).

$$\|d(x_1, x_2)\|_2 = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

$x_1 = [x_{11}, x_{12}, x_{13}, \ldots x_{1n}]$

$x_2 = [x_{21}, x_{22}, x_{23}, \ldots x_{2n}]$

Triplet loss:

$$\|f(x_i^a) - f(x_i^p)\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha$$

$$\forall \left(f(x_i^a), f(x_i^p), f(x_i^n)\right) \in T$$

$$\text{Loss} = \max\left(0, \sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]\right)$$

i.e,

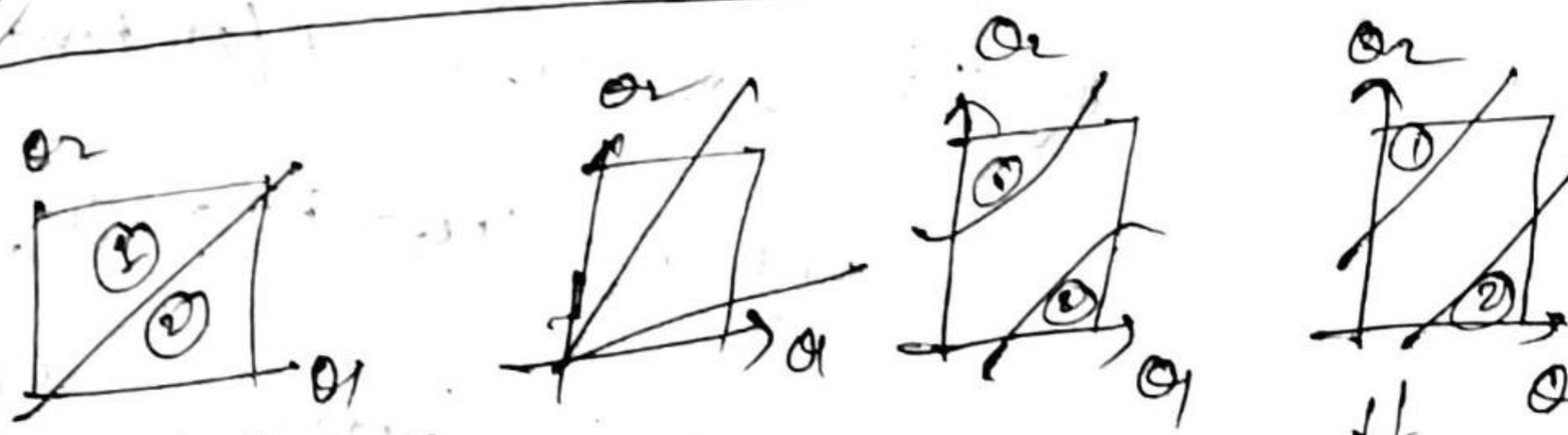
$$\text{Loss}_{\text{Triplet}} = \max\left(0, [\text{Pos\_dist} - \text{Neg\_dist} + \alpha]\right)$$

Arcface → Modified softmax loss

$$\text{Loss} = \frac{-1}{N} \sum_{i=1}^{N} \log \frac{e^{W_{y_i} x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j x + b_j}}$$

~~Largface = $\frac{1}{N}\sum \log \frac{e^{s(\cos(\theta_{y_i}+m)-\cos\theta)}}{\cdots}$~~

$$Largface = \frac{-1}{N}\sum \log \frac{e^{s(\cos(\theta_{y_i}+m))}}{e^{s(\cos(\theta_{y_i}+m)} + \sum_{\substack{j=1 \\ j\neq y_i}}^{n} e^{s\cos\theta_j}}$$



Cargo : Arcface

train.py *optimizer = Adam(list(model.parameters()) + list(detach_model.parameters()), lr=lr, amsgrad=True, weight_decay=1e-4)

class Arcface(nn.Module):

return SaOutput

model = S.NFaceNetModel()

detach_model = Arcface()

criterion = nn.CrossEntropyLoss() → training loss

def train(model, -- detach_model):

for epoch ——:

anc_embed = model(anc_img)

thetas = detach_model(anc_embed, anc_cls)

loss = criterion(thetas, anc_cls).
↳ label

optimizer.zero_grad(); loss.backward(), optimizer.step()
loss_sum += loss.item()
avg_loss = loss_sum/len(dataloader['train']); print(CE-loss = avg_loss)

Valid(model, ---dataloader, --- detach_model):

for batch_idx, batch_triple in enumerate(dataloader[valid]):
anc_embed = model(anc_img)
pos_embed = model(pos_img)
neg_embed = model(neg_img)

pos_dist = l2_dist.forward(anc_embed, pos_embed)
neg_dist = l2_dist.forward(anc_embed, neg_embed)

all = (neg_dist - pos_dist < margin).

tripletloss = TripletMarginLoss(margin).forward(anc_hard_emb,
pos_hard_embed, neg_hard_embed).

dista = l2_dist.forward(anc_embed, pos_embed)
distances.append(dista.data.cpu().numpy())
label.append(np.ones(dista.size(0)))

distb = l2_dist.forward(anc_embed, neg_embed)
distances.append(distb.data.cpu().numpy())
labels.append(np.zeros(dist.size(0)))

triplet_loss_sum += triplet_loss.item()

avg_triplet_loss = triplet_loss_sum / len(dataloaders['valid'])
labels = np.array([sublabel for label in labels for
subsublabel in label])
distances = np.array([subdist for dist in distances
for subdist in dist])

```
tpr, fpr, accuracy = evalute (distances, labels)

if phase == 'valid':
    torch.save (model.state_dict())
    torch.save (detech_model.state_dict())

plot_roc( fpr, tpr, figure_nae = "abc.png")

val_loss = avg_triplet_loss

return val_loss.
```

label list for dist

label → anc & pos → 1
↘ anc & Neg → 0. for dist.

t])

Scanned by TapScanner

ALPR (WPod ⇒ Wraped planar Object dection)

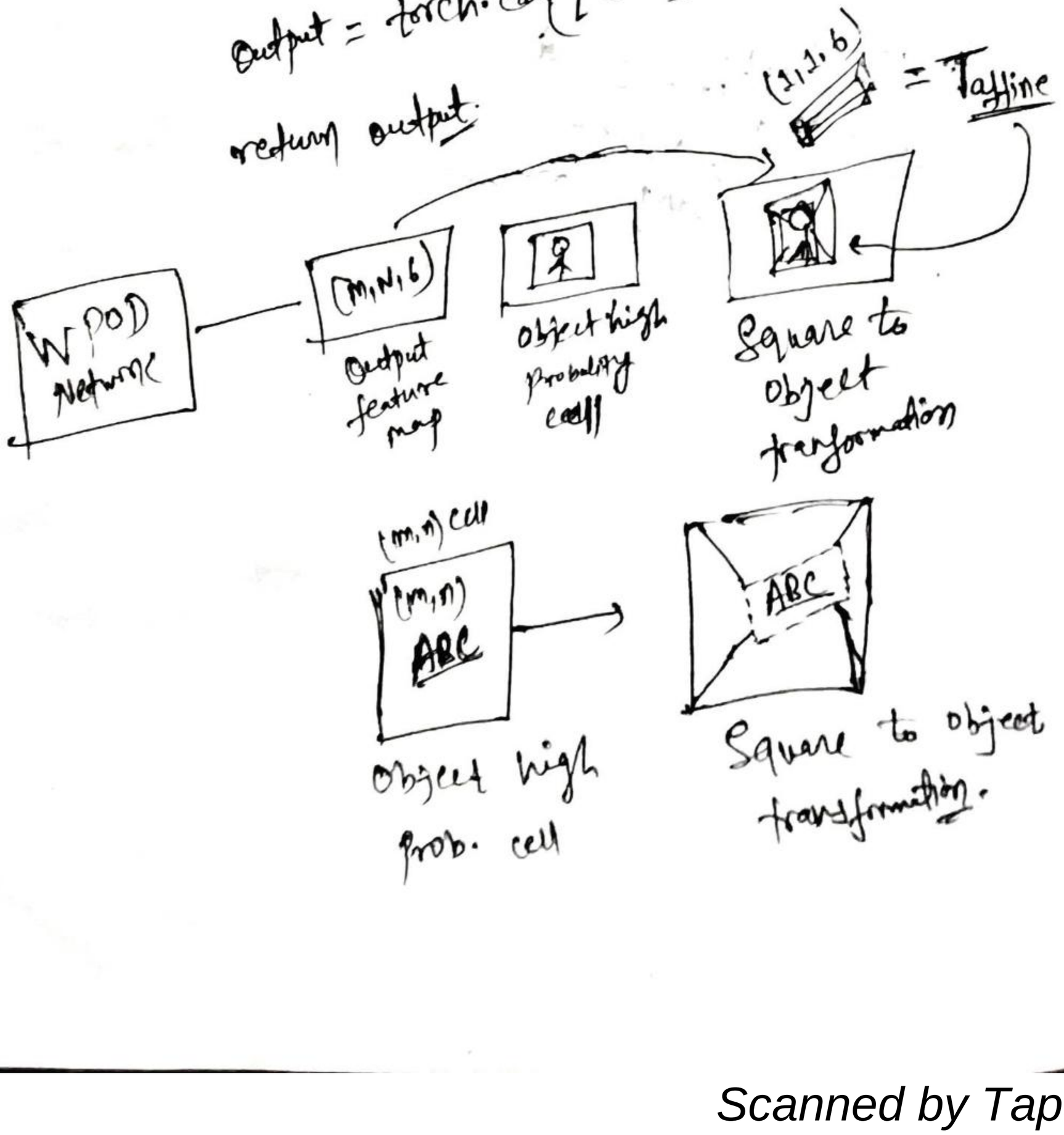
ResNet o/p ⇒ (128,)

self.clf ⇒ (128, 2)

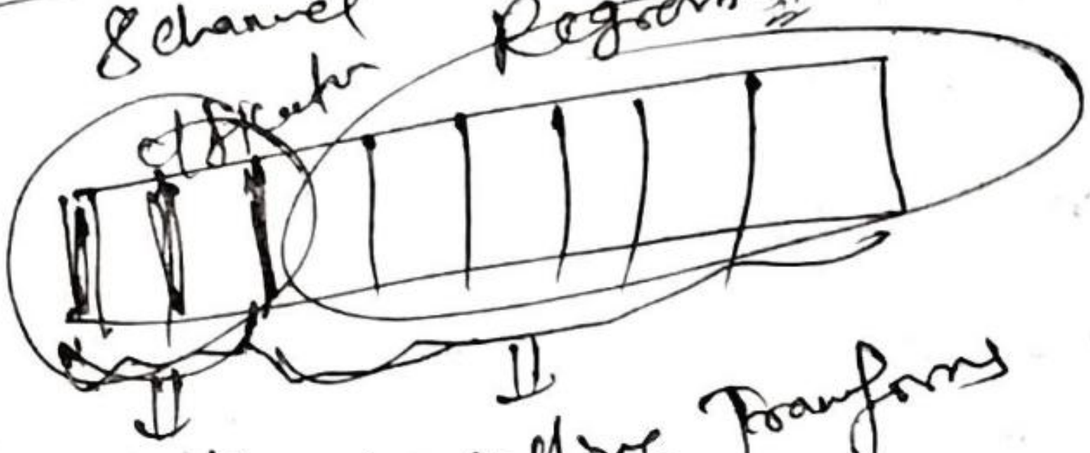
self.regression ⇒ (128, 6)

Code:-

```
x = self.layer4(x)

#clf
c = self.classification(x);
c = self.softmax(c)

#reg.
r = self.regression(x)
output = torch.cat([c, r], dim=1)

return output
```



Feature maps



licence_plate_detection_vid.py → get_licence_plate, get_license_plate_pytorch

vehicle_detection_vid.py

Lpcenteplate_ocr_vid.py.

~~Lp_detection~~.models.py → WpodNet

lp_digit_recognition.py

ocr.py.

post_process.py

utils.py

~~vehicle_detection_v~~

vms_demo.py.

. Vehicle-detection.py

sys.path.append('./yolo')
from yolo predict
import darknet as dn

postprocess → lp_sort

ocr → character_recogtn

cfg import → wpod_net path, lp_detect_model

vcent plate_detection_vid import → get_license_plate,
Get_license_plate_pytorch.

lp_detection_model import → WpodNet

~~Yolo_v3, Lp_detection, Ocr~~

① vehicle detection (YOLO V3)

② Lp-detection (wpod)

③ OCR → Lp-Segmentation (RetinaNet)
Lp-classification (CNN)
↓
(ResNet)

reconstruct_pytorch (Img, I, Y, outsize, threshold=0.9)

Img → Image
I → resize_img
Y → recon_input
outsize → (240, 80)
threshold → 0.5

x = (2, 3, 1)

x[:, :, 0]

Alpr summary

① vehicle-detection → YOLOV3
② LP-detection → WPOD Net + Prospective Transform.
③ LP-segmentation → RetinaNet
④ LP-classification → CNN (ResNet) based Clf model.

- post processing:

① d2c map
② c2d map

#Irate ODI & UN Number SUMMARY

① ODI (YOLOV3)

- CN
- UN
- LP
- Seals
- LP Sticker
- Hazard category
- hole
- dent

↳ Yolo detects each of the above classes
↳ Saved croped image for each classes & stored it in respective input folder (in .npy format)
for further processing.

for some module & for some module it was saved in CV2 image format.

② UN seg. & clf.

- UN No. seg : → RetinaNet
- UN No. clf → CNN (ResNet) Based Model.
- UN No. post processing.
  - ↳ c2cmap
  - ↳ c2dmap
  - ↳ K-Means Clustering Algorithm.

KYCL SUMMARY

① Twist-lock ROI detection.
   ↳ RetinaNet

② Twist-lock classification
   ↳ CNN (ResNet) Based classification Model.

③ openVino conversion of pytorch Model.

Training set up

Face Recognition

(1) face detection → RetinaNet ⇒ our emp. data

(2) faceNet → (face Recog.) → LFW dataset (14~16 K)

VMS

- vehicle detection → YOLO
- LP-detection → WPOD
- LP-Segmentation → RetinaNet
- LP-classification → CNN(Resnet) based Model.

→ our Car dataset

Hate:

- ODI → YOLO (~60k images)
- UN No. → original, GAN Gen, Collected from internet
  - seg-RetinaNet
  - clf - CNN (ResNet)
    - Type-1 UN No.
    - Type-2 UN No.

KYC

- Twistlock RoI Detection - RetinaNet
- twist lock right clf → CNN (ResNet).

Cargo.

- cargo-classificon (Reg-Misc.) → CNN (ResNet)
- [illegible] (cargo box detion): ⇒ YOLO V3
- [illegible]king (DeepSort, Kalman filter)
- feature extraction (Arcface Model - [illegible])
  - [illegible]ss Entropy loss (train)
  - Triplet loss (val).

Cargo dataset