

VİSEA TEST PROJESİ RAPORU

HEDEF: Segmentasyon ve sınıflandırma derin öğrenme modelleri kullanarak insan ve araba tespiti yapan bir uygulamanın geliştirilmesi.

Tarafıma gönderilmiş test proje raporu incelenip bir akış şeması oluşturuldu. Buna göre ilk önce veri seti incelendi ve bir semantic segmentasyon modeli nasıl eğitilir araştırıldı.

İncelenen veri setinden her sınıfın hangi RGB pixel değerleriyle etiketlendiği bulundu. Bu sayede insan ve araba sınıfları ayrıştırılarak 0 değerli pixeller arka plan, 1 değerli pixeller araba ve 2 değerli pixeller insan olacak şekilde bir maske veri seti oluşturuldu. Resim 1 'de ilgili kod bloğu verilmiştir.

```
1 import numpy as np
2 import torch.nn as nn
3
4 class Color_to_label_map(nn.Module):
5     def __init__(self):
6         super(Color_to_label_map, self).__init__()
7         self.car_color=(142, 0, 0)
8         self.human_colors = [(60, 20, 220), (0, 0, 255)]
9
10
11     def forward(self,x):#RGB IMAGE
12         # Görüntüyü düzenle
13         self.zeros_mask=np.zeros(x.shape[:2])
14         mask_car = np.all(x == np.array(self.car_color), axis=2)
15         mask_human = np.any(np.all(x[:, :, None, :] == np.array(self.human_colors)[None, None, :, :], axis=3), axis=2)
16
17         self.zeros_mask[mask_car] = 1
18         self.zeros_mask[mask_human] = 2
19
20         return self.zeros_mask
```

Resim 1 Sınıf Etiketlenmesi

Veri seti düzenlendikten sonra model eğitimi için uygun modeller aranmaya başlanmıştır. İlk başta Pytorch'un pre-trained modellerinden olan DeepLabV3 denenmiştir. Ancak bu model üzerinde denemeler yapılmış olsada bir türlü istenilen sonuca ulaşamamıştır. Resim 2' de ilgili modelin kodu verilmiştir.

```
1 import torch
2 import torch.nn as nn
3 import torchvision.models.segmentation as models
4 import torchvision
5 import warnings
6 warnings.filterwarnings("ignore")
7 class DeepLabV3(nn.Module):
8     def __init__(self, num_classes):
9         super(DeepLabV3, self).__init__()
10         self.deeplabv3 = torchvision.models.segmentation.deeplabv3_resnet101(pretrained=True)
11
12         self.deeplabv3.classifier[-1] = nn.Conv2d(256, num_classes, kernel_size=(1, 1))
13         #print(self.deeplabv3)
14     def forward(self, x):
15         return self.deeplabv3(x)['out']
```

Resim 2 DeepabV3 Pre-trained Modelin Kodu

Uzun uğraşlara rağmen bu modelden istenilen sonuç alınamayınca GitHub'da bulunan segmentation_models repository-si incelendi. Model bu repository-e göre tasarlanıp train ve test kodlarının custom yazılmasına karar verildi. Resim 3' de kullanılan modelin mimari kodu verilmiştir.

```
1 def __model_init(self):
2
3     self.model = smp.Unet(
4         encoder_name=self.encoder,
5         encoder_weights=self.encoder_weights,
6         in_channels=3,
7         classes=self.num_classes #model output
8     )
```

Resim 3 U-Net Model Mimarisinin Kodu

U-Net mimarisi baz alınarak encoder olarak “resnet34” mimarisi kullanılmıştır. Encoder weights olarak ise “imagenet” kullanılmıştır. “in_channels” modele girecek olan verinin channel sayısını, classes ise kaç sınıflı bir model eğitiminde buluncağımızı belirtmektedir. Modelin çıktısını 3 channel olarak ayarlayarak her channel-ın bir sınıfı temsil etmesini sağlıyoruz.

Bunlara ek olarak eğitim için Focal Loss ve Adam optimizasyon algoritmaları kullanılmıştır. Learning rate “0.001” seçilmiştir. İlk eğitim aşamalarında Cross Entropy Loss algoritması kullanılmıştır. Ancak veri setinin unbalanced dağılımından dolayı Focal Loss algoritmasında daha iyi sonuçlar alınabileceği düşünülüp Focal Loss’ lu bir eğitim de yapılmıştır. Focal Loss’lu eğitimin sonuçları daha yüksek doğruluklu olmuştur. Focal Loss algoritması modelin yapmış olduğu yanlış tahminlere odaklanarak bir sonuç oluşturmaktadır. Resim 4’de Focal Loss algoritmasının kodu verilmiştir.

```
1 class FocalLoss(nn.Module):
2     def __init__(self, gamma=0, alpha=None, reduction='mean'):
3         super(FocalLoss, self).__init__()
4         self.reduction = reduction
5         self.gamma = gamma
6         self.alpha = alpha
7         if isinstance(alpha, (float, int, torch.LongTensor)): self.alpha = torch.Tensor([alpha, 1 - alpha])
8         if isinstance(alpha, list): self.alpha = torch.Tensor(alpha)
9
10    def forward(self, input, target):
11        if input.dim() > 2:
12            input = input.view(input.size(0), input.size(1), -1) # N,C,H,W => N,C,H*W
13            input = input.transpose(1, 2) # N,C,H*W => N,H*W,C
14            input = input.contiguous().view(-1, input.size(2)) # N,H*W,C => N*H*W,C
15            target = target.view(-1, 1)
16
17            logpt = F.log_softmax(input)
18            logpt = logpt.gather(1, target)
19            logpt = logpt.view(-1)
20            pt = Variable(logpt.data.exp())
21
22            if self.alpha is not None:
23                if self.alpha.type() != input.data.type():
24                    self.alpha = self.alpha.type_as(input.data)
25                at = self.alpha.gather(0, target.data.view(-1))
26                logpt = logpt * Variable(at)
27
28            loss = -1 * (1 - pt) ** self.gamma * logpt
29            if self.reduction == "none":
30                return loss
31            elif self.reduction == "mean":
32                return loss.mean()
33            elif self.reduction == "sum":
34                return loss.sum()
```

Resim 4 Focal Loss Kodu

Tüm bunların eşliğinde bir model eğitimi yapılmıştır. Train veri setinin %10 'lık kısmı validation olarak ayrılmıştır. Batch_size 6 olarak seçilip veriler (512,512) boyutlarında işlenilmiştir. Resim 5’de train kodu verilmiştir.

```
1 def __train_fit(self,model, dataloader, data, optimizer, criterion):
2     print('-----Training-----')
3
4     self.model.train()
5     train_running_loss = 0.0
6
7     counter = 0
8     num_batches = int(data / dataloader.batch_size)
9
10    for i, d in tqdm(enumerate(dataloader), total=num_batches):
11        counter += 1
12        image, mask = d[0].to(self.device), d[1].to(self.device)
13
14        optimizer.zero_grad()
15        outputs = model(image)
16        mask=mask.squeeze(1).long()
17
18        loss = criterion(outputs, mask)
19
20        train_running_loss += loss.item()
21
22        loss.backward()
23        optimizer.step()
24
25    train_loss = train_running_loss / counter
26
27
28    return train_loss
```

Resim 5 Train Kodu

Eğitim sonucu en düşük validation loss-u olan modeller ve belirli epochlar kaydedilmiştir. Modelin çıktılarını görebilmek için bir inference kodu yazılmıştır. Resim 6’ da inference-prediction kodu verilmiştir.

```
1 x_tensor = image.to(self.device).unsqueeze(0).float()
2 mask = model.predict(x_tensor).squeeze(0)
3
4 if self.device == "cpu":
5     output_predictions = torch.argmax(torch.softmax(mask, dim=0), dim=0).unsqueeze(2)
6 else:
7     output_predictions = torch.argmax(torch.softmax(mask, dim=0), dim=0).cpu().unsqueeze(2)
```

Resim 6 Prediction Kodu

Model çıktıları matplotlib kütüphanesi ile görselleştirilmiştir. Model çıktılarının görüntüleri Resim 7’de verilmiştir.



Resim 7 Sonuçlar

Sonuçlar bu şekilde alındıktan sonra görevlerde belirtilen Connected Component Detection algoritmasının uygulanıp sınıflandırma yapılabilmesi için şu şekilde bir yöntem izlenilmiştir.

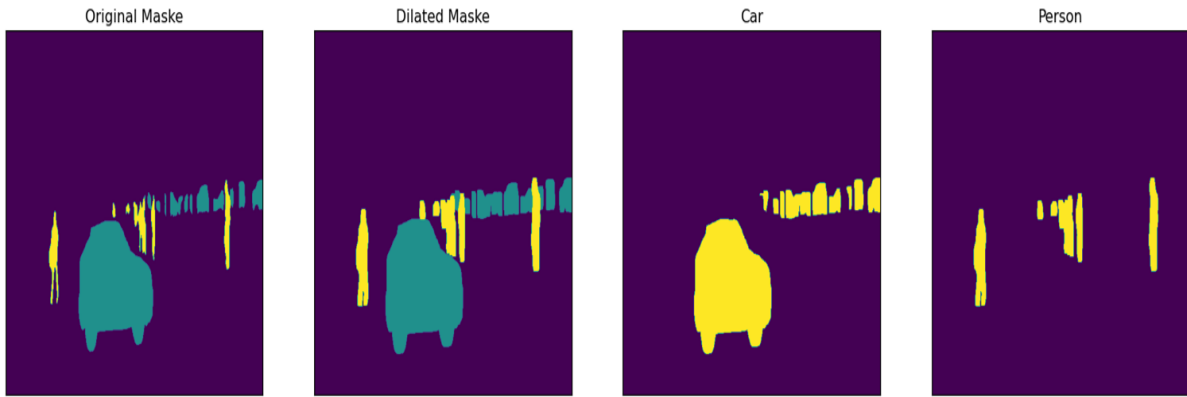
1-) Oluşturulmuş model ile tahmin edilen maskelere önce morphological operasyonlardan “dilate” işlemi uygulanmıştır. Resim 8’de verilen örnekteki demirliklerin aracı birden fazla parçaya bölmesinden dolayı bu tarz verilerde nesnenin bütünlüğünün korunması amaçlanmıştır.

2-) “Dilate” edilmiş tek chanel-lı maskeler, one-hot-encoding algoritmasıyla işlenerek 3 channel-e çıkarılmıştır ve her channel tek bir sınıfı içermektedir. 1.channel background, 2.channel araba ve 3.channel ise insan sınıfını temsil etmektedir.

3-) Connected Component Detection için OpenCV kütüphanesinden “cv2.connectedComponentsWithStats” algoritması kullanılmıştır.

4-) Her biri ayrı bir sınıfı temsil eden channel-ların bounding boxes-ları connected component algoritmasıyla ayrı ayrı olarak hesaplanarak sınıflandırma ve konum bulma işlemi tamamlanmıştır.

Resim’10 da one-hot-encoding sonuçları ve dilate edilmiş maske görüntüleri verilmiştir.



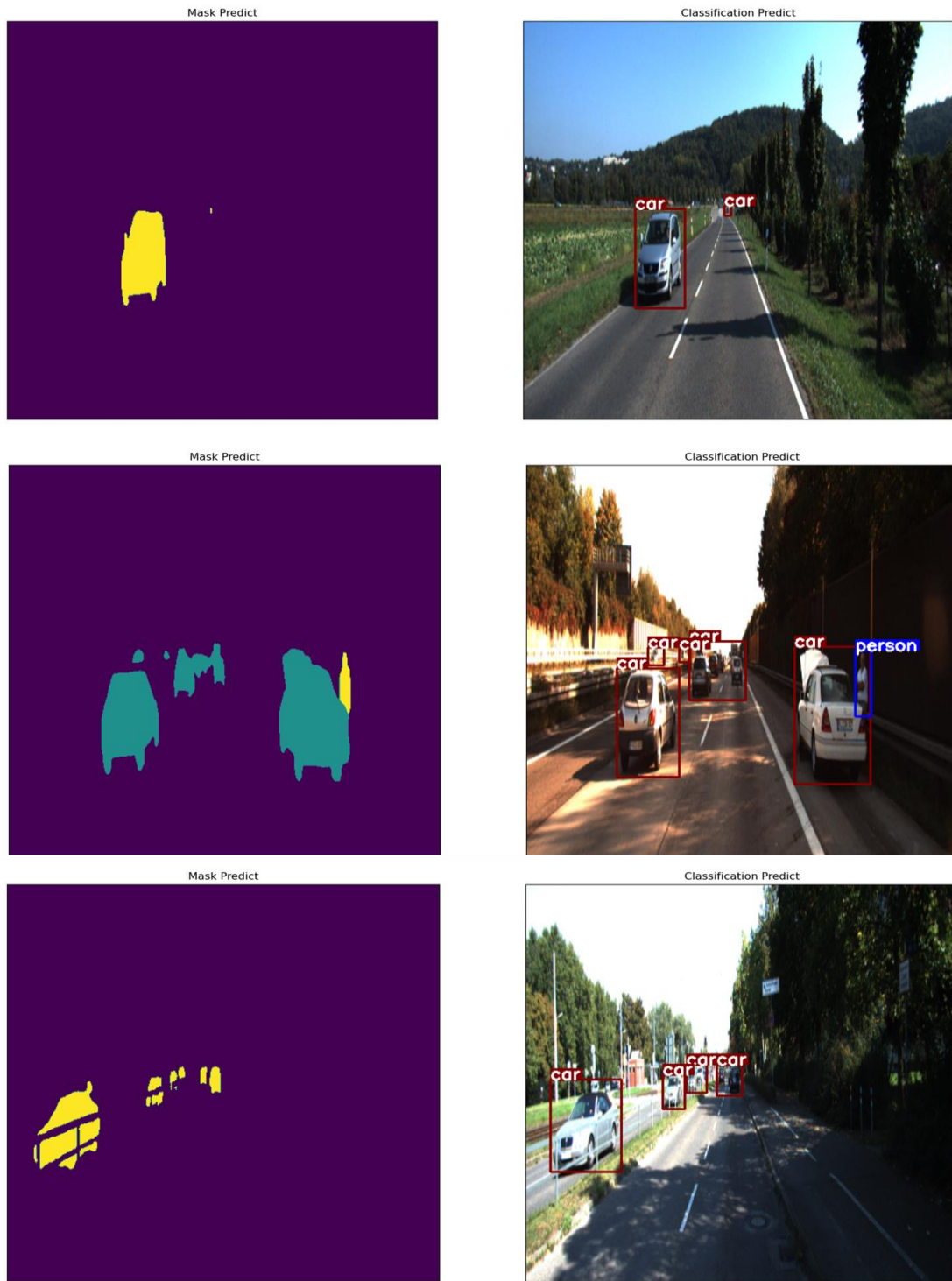
Resim 10 One-Hot Encoding ve Dilate Sonucu

Resim 11’de araba ve insan sınıflarını temsil eden channel-ın ayrı ayrı connected component algoritmasına verilip bounding boxes-ların nasıl çıkarıldığı gösterilmiştir.

```
1   for counter_classes in range(1,self.num_classes): #i=1, i=2
2
3       input_for_component_mask=np.array(one_hot_mask[:, :, counter_classes].unsqueeze(2),dtype=np.uint8)
4
5       output = cv2.connectedComponentsWithStats(
6           input_for_component_mask, 4, cv2.CV_32S)
7
8       (numLabels, labels, stats, centroids) = output
9
10
11      list_of_bbox.clear()
12
13      for i in range(1, numLabels):
14          x = stats[i, cv2.CC_STAT_LEFT]
15          y = stats[i, cv2.CC_STAT_TOP]
16          w = stats[i, cv2.CC_STAT_WIDTH]
17          h = stats[i, cv2.CC_STAT_HEIGHT]
18          bbox=(x,y,w,h) #bbox analys
19
20
21          #kucuk maskeleri alma
22          keepWidth = w > 5 and w < 1000
23          keepHeight = h > 5 and h < 1000
24
25
26          if all((keepWidth, keepHeight)):
27              list_of_bbox.append(bbox)
28
29
30      if(counter_classes==1): #back groundu hesaba katmıyoruz
31          self.bbox_dict["car"] = list_of_bbox.copy()
32
33      elif(counter_classes==2):
34          self.bbox_dict["person"] = list_of_bbox.copy()
```

Resim 11 One-Hot ve Connected Component İle Bounding Boxes Çıkarma Kodu

Bu işlemler sonucu her biri sınıflara ayrılmış channel-ın içerisinde bulunan bounding boxes-lar bulunmuştur. Her channel-ın hangi sınıfı temsil ettiğinin de bildiğimiz için sınıflandırma işlemi bu şekilde tamamlanmıştır. Resim 12’de nihai sonuçlar verilmiştir.



Resim 12 Nihai Sonuçlar