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 ayrıştıralarak 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.

```python
import numpy as np
import torch.nn as nn

class Color_to_label_map(nn.Module):
    def __init__(self):
        super(Color_to_label_map, self).__init__()
        self.car_color=(142, 0, 0)
        self.human_colors = [(60, 20, 220), (0, 0, 255)]


    def forward(self,x):#RGB İMAGE
            # Görüntüyü düzenle
            self.zeros_mask=np.zeros(x.shape[:2])
            mask_car = np.all(x == np.array(self.car_color), axis=2)
            mask_human = np.any(np.all(x[:, :, None, :] == np.array(self.human_colors)[None, None, :, :], axis=3), axis=2)

            self.zeros_mask[mask_car] = 1
            self.zeros_mask[mask_human] = 2

            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 ulaşılamamaştır. Resim 2’ de ilgili modelin kodu verilmiştir.

```python
import torch
import torch.nn as nn
import torchvision.models.segmentation as models
import torchvision
import warnings
warnings.filterwarnings("ignore")
class DeepLabV3(nn.Module):
    def __init__(self, num_classes):
        super(DeepLabV3, self).__init__()
        self.deeplabv3 = torchvision.models.segmentation.deeplabv3_resnet101(pretrained=True)

        self.deeplabv3.classifier[-1] = nn.Conv2d(256, num_classes, kernel_size=(1, 1))
        #print(self.deeplabv3)
    def forward(self, x):
        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.

```python
def __model_init(self):

        self.model = smp.Unet(
            encoder_name=self.encoder,
            encoder_weights=self.encoder_weights,
            in_channels=3,
            classes=self.num_classes #model output
        )
```

Resim 3 Unet Model Mimarisinin Kodu

U-net modeli baz alınarak encoder olarak "resnet34" mimarisi kullanılmıştır. Encoder weights olarak ise "imagenet" kullanılmıştır. "in_channels" modele girecek olan verinin channel (kanal) sayısını, classes ise kaç sınıflı bir model eğitiminde buluncağımızı belirtmektedir. Modelin çıktısını 3 (kanal) olarak ayarlayarak her kanalın bir sınıfı temsil etmesini sağlıyoruz.

Bunlara ek olarak eğitim için FocalLoss ve Adam optimizasyon algoritması kullanılmıştır. Learning rate "0.001" seçilmiştir. İlk eğitim aşamalarında CrossEntropyLoss kullanılmıştır. Ancak veri setinin unbalanced dağılımından dolayı FocalLoss' da daha iyi sonuçlar alınabileceği düşünülüp FocalLoss' lu bir eğitim de yapılmıştır. FocalLoss'lu eğitimin sonuçları daha yüksek doğruluklu olmuştur. FocalLoss modelin yanlış tahminlerine odaklanarak bir sonuç oluşturmaktadır. Resim 4'de FocalLoss algoritması verilmiştir.

```python
class FocalLoss(nn.Module):
    def __init__(self, gamma=0, alpha=None, reduction='mean'):
        super(FocalLoss, self).__init__()
        self.reduction = reduction
        self.gamma = gamma
        self.alpha = alpha
        if isinstance(alpha, (float, int, torch.LongTensor)): self.alpha = torch.Tensor([alpha, 1 - alpha])
        if isinstance(alpha, list): self.alpha = torch.Tensor(alpha)

    def forward(self, input, target):
        if input.dim() > 2:
            input = input.view(input.size(0), input.size(1), -1)  # N,C,H,W => N,C,H*W
            input = input.transpose(1, 2)  # N,C,H*W => N,H*W,C
            input = input.contiguous().view(-1, input.size(2))  # N,H*W,C => N*H*W,C
        target = target.view(-1, 1)

        logpt = F.log_softmax(input)
        logpt = logpt.gather(1, target)
        logpt = logpt.view(-1)
        pt = Variable(logpt.data.exp())

        if self.alpha is not None:
            if self.alpha.type() != input.data.type():
                self.alpha = self.alpha.type_as(input.data)
            at = self.alpha.gather(0, target.data.view(-1))
            logpt = logpt * Variable(at)

        loss = -1 * (1 - pt) ** self.gamma * logpt
        if self.reduction == "none":
            return loss
        elif self.reduction == "mean":
            return loss.mean()
        elif self.reduction == "sum":
            return loss.sum()
```

Resim 4 Focal Loss Kodu

Tüm bunlarin 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.

```python
def __train_fit(self,model, dataloader, data, optimizer, criterion):
    print('-------------Training-----------------')

    self.model.train()
    train_running_loss = 0.0

    counter = 0
    num_batches = int(data / dataloader.batch_size)

    for i, d in tqdm(enumerate(dataloader), total=num_batches):
        counter += 1
        image, mask = d[0].to(self.device), d[1].to(self.device)

        optimizer.zero_grad()
        outputs = model(image)
        mask=mask.squeeze(1).long()

        loss = criterion(outputs, mask)

        train_running_loss += loss.item()

        loss.backward()
        optimizer.step()

    train_loss = train_running_loss / counter


    return train_loss
```

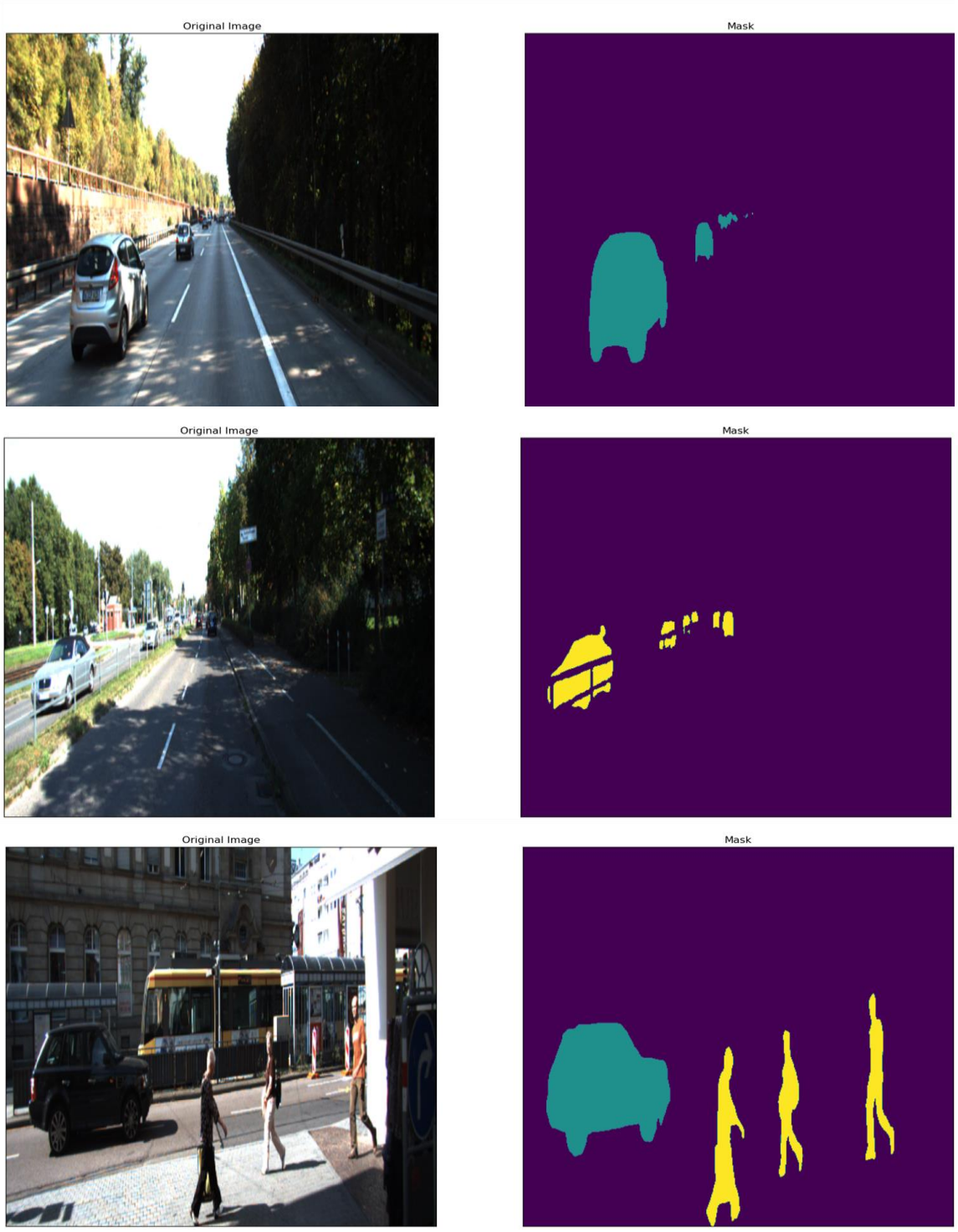
Resim 5 Train Kodu

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

```python
x_tensor = image.to(self.device).unsqueeze(0).float()
mask = model.predict(x_tensor).squeeze(0)

if self.device == "cpu":
    output_predictions = torch.argmax(torch.softmax(mask, dim=0), dim=0).unsqueeze(2)
else:
    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-) Tahmin edilen maske verisi, one-hot-encoding edilerek 3 channel-e çıkarılmıştır ve her channel tek bir sınıfı içermektedir.

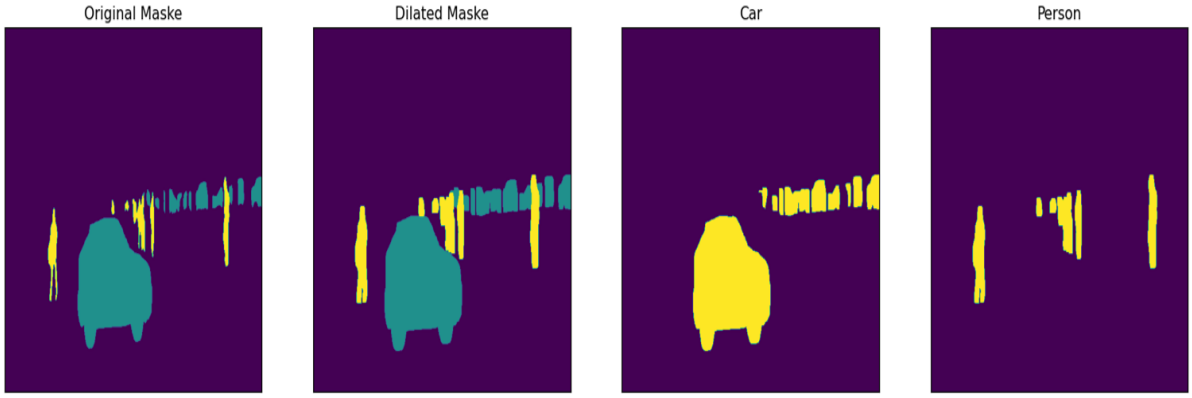
2-) 1.channel background 2.channel araba ve 3.channel ise insan sınıfını temsil etmektedir.

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

4-) Her channel-ın bounding boxes-ları ayrı olarak hesaplanarak sınıflandırma ve konum bulma işlemi tamamlanmıştır.

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

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



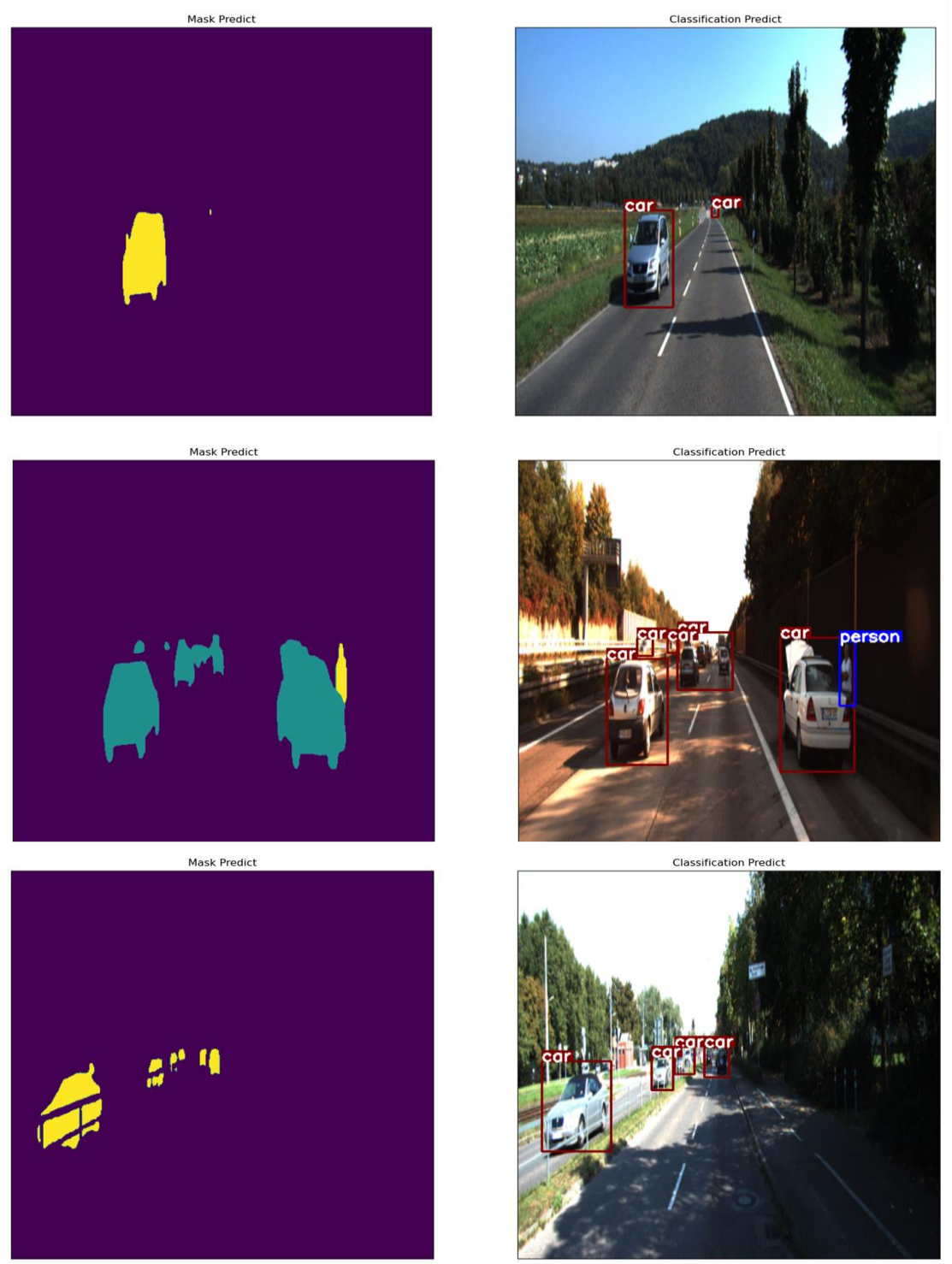
Resim 10 One-hot Encoding Sonucu

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

```python
for counter_classes in range(1,self.num_classes): #i=1, i=2

    input_for_component_mask=np.array(one_hot_mask[:,:,counter_classes].unsqueeze(2),dtype=np.uint8)

    output = cv2.connectedComponentsWithStats(
    input_for_component_mask, 4, cv2.CV_32S)

    (numLabels, labels, stats, centroids) = output


    list_of_bbox.clear()

    for i in range(1, numLabels):
        x = stats[i, cv2.CC_STAT_LEFT]
        y = stats[i, cv2.CC_STAT_TOP]
        w = stats[i, cv2.CC_STAT_WIDTH]
        h = stats[i, cv2.CC_STAT_HEIGHT]
        bbox=(x,y,w,h) #bbox analys


        #kucuk maskeleri alma
        keepWidth = w > 5 and w < 1000
        keepHeight = h > 5 and h < 1000


        if all((keepWidth, keepHeight)):
            list_of_bbox.append(bbox)


    if(counter_classes==1): #back groundu hesaba katmıyoruz
        self.bbox_dict["car"] = list_of_bbox.copy()

    elif(counter_classes==2):
        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 boxeslar bulunmuştur. O channel-ın hangi sınıfı temsil ettiğinide bildiğimiz için sınıflandırma işlemi bu şekilde tamamlanmıştır. Aşağıda nihai sonuçlar verilmiştir.



Resim 12 Nihai Sonuçlar