

# BLM4540 Görüntü İşleme - Proje

Toygar Tanyel  
18011094

## Yöntem

Bu kısımda, önce verileri almak için PyTorch kullanarak hazır yöntemler denedim. Ancak, PyTorch veya TensorFlow'un özel tren-test-val oluşturmaya izin vermediği için önemli sorunlar vardı. Ayrıca, herhangi bir framework ile alınan verilerle düzenleme yapmaya uğraşmak çok daha karmaşık.

Bu nedenle, verileri manuel olarak ele aldım ve böldüm. Hazırlığın tamamı aşağıda bulunabilir (bazı fonksiyon içeriklerini atlayarak):

```
dataset_directory = os.path.join("./dataset")
os.mkdir(dataset_directory)

# resimleri getir ve zip'ten çıkar
filepath = os.path.join(dataset_directory, "images.tar.gz")
download_url(url="https://www.robots.ox.ac.uk/~vgg/data/pets/data/images.tar.gz",
filepath=filepath)
extract_archive(filepath)

# etiketleri getir ve zip'ten çıkar
filepath = os.path.join(dataset_directory, "annotations.tar.gz")
download_url(url="https://www.robots.ox.ac.uk/~vgg/data/pets/data/annotations.tar.
gz", filepath=filepath)
extract_archive(filepath)

# test ve train birleştir
filepath = os.path.join(dataset_directory)
merge_trainval_test(filepath)
-> Bunu yapma sebebi tf ve torch içinde veri seti %50 %50 train-test olarak ayrılmış ve bu genelde istediğimiz bir şey değil. Manuel olarak orijinal path'ten aldığım veri setini kullanarak yeni bir veri seti oluşturacağım fakat bunu yapmadan önce verileri uygun şekilde dosyalarımız gerekiyor. Aşağıdaki fonksiyon bu işi yapacak.

def merge_trainval_test(filepath):
    """
    # Image CLASS-ID SPECIES BREED ID
    # ID: 1:37 Class ids
    # SPECIES: 1:Cat 2:Dog
    # BREED ID: 1-25:Cat 1:12:Dog
    # All images with 1st letter as captial are cat images
    # images with small first letter are dog images
    """

```

```

merge_dir = os.path.dirname(os.path.abspath(f'{filepath}/annotations/data.txt'))
if os.path.exists(merge_dir):
    print("Merged data is already exists on the disk. Skipping creating new data
file.")
    return
df = pd.read_csv(f'{filepath}/annotations/trainval.txt', sep=" ",
                 names=["Image", "ID", "SPECIES", "BREED ID"])
df2 = pd.read_csv(f'{filepath}/annotations/test.txt', sep=" ",
                  names=["Image", "ID", "SPECIES", "BREED ID"])
frame = [df, df2]
df = pd.concat(frame)
df.reset_index(drop=True)
df.to_csv(f'{filepath}/annotations/data.txt', index=None, sep=' ')
print("Merged data is created.")

```

Veriyi istediğimiz gibi birleştirdik fakat analiz etmek için bazı değiştirmemiz gereken şeyler daha var. (Manuel olarak çalışmanın zararları)

Herhangi bir orijinal dosyadan,

```
idx_to_class = dict(zip(range(len(classes)), classes))
```

sözlüğünü oluşturmalıyız. Bunun sebebi başta nümerik olarak hangi sınıflara sahip olduğumuzu biliyoruz fakat string olarak gösteremiyoruz.

- Filenam (string) processing ile classları çıkartacağız.
- Daha sonra bu sözlüğü kullanarak mapleme işlemi yapacağız.  
(Not: orijinal veri 0-36 değil 1-37 arasında, sözlüğü kullanabilmek için bu değerleri 0-36 aralığına çekmeliyiz)

-> İlgili mapleme işlemi sonrası:

```

{0: 'Abyssinian',
 1: 'American Bulldog',
 2: 'American Pit Bull Terrier',
 3: 'Basset Hound',
 4: 'Beagle',
 5: 'Bengal',
 6: 'Birman',
 7: 'Bombay',
 8: 'Boxer',
 9: 'British Shorthair',
 10: 'Chihuahua',
 11: 'Egyptian Mau',
 12: 'English Cocker Spaniel',
 13: 'English Setter',
 14: 'German Shorthaired',
 15: 'Great Pyrenees',
 16: 'Havanese',
 17: 'Japanese Chin',
 18: 'Keeshond',
 19: 'Leonberger',
 20: 'Maine Coon',
 21: 'Miniature Pinscher',
 22: 'Newfoundland',
 23: 'Persian',
 24: 'Pomeranian',
 25: 'Pug',
 26: 'Ragdoll',
 27: 'Russian Blue',
 28: 'Saint Bernard',
 29: 'Samoyed',
 30: 'Scottish Terrier',
 31: 'Shiba Inu',
 32: 'Siamese',
 33: 'Sphynx',
 34: 'Staffordshire Bull Terrier',
 35: 'Wheaten Terrier',
 36: 'Yorkshire Terrier'}

```

Bu classlar bizim için önemli çünkü veriyi dağıtırken sınıflarına uygun olarak dağıtmalıyız. Stratify kullanmadan dağıtırsak hangi türden nereye kaç tane gideceğini kontrol edemeyiz.

		200	Image	ID	SPECIES	BREED ID	nID	class
Yorkshire Terrier		200		0	Abyssinian_100	1	1	Abyssinian
Samoyed		200		1	Abyssinian_101	1	1	Abyssinian
Russian Blue		200		2	Abyssinian_102	1	1	Abyssinian
Ragdoll		200		3	Abyssinian_103	1	1	Abyssinian
Pug		200		4	Abyssinian_104	1	1	Abyssinian
Pomeranian		200		...	...	...	...	...
Persian		200	7344	yorkshire_terrier_96	37	2	25	36 Yorkshire Terrier
Miniature Pinscher		200	7345	yorkshire_terrier_97	37	2	25	36 Yorkshire Terrier
Maine Coon		200	7346	yorkshire_terrier_98	37	2	25	36 Yorkshire Terrier
Leonberger		200	7347	yorkshire_terrier_99	37	2	25	36 Yorkshire Terrier
American Bulldog		200	7348	yorkshire_terrier_9	37	2	25	36 Yorkshire Terrier
Japanese Chin		200						
Havanese		200						
Great Pyrenees		200						
German Shorthaired		200						
English Setter		200						
Shiba Inu		200						
Chihuahua		200						
British Shorthair		200						
Sphynx		200						
Wheaten Terrier		200						
Birman		200						
Bengal		200						
Beagle		200						
Basset Hound		200						
American Pit Bull Terrier		200						
Saint Bernard		200						
Siamese		199						
Scottish Terrier		199						
Keeshond		199						
Boxer		199						
Abyssinian		198						
Newfoundland		196						
English Cocker Spaniel		196						
Egyptian Mau		190						
Staffordshire Bull Terrier		189						
Bombay		184						
Name: class, dtype: int64								

-> Elimizdeki tüm veri (train-val-test)

Stratify ile ayırdıktan sonra özel veri kümem sunları içermeye:

- eğitim: 4115 örnek
- doğrulama: 1764 örnek
- test: 1470 örnek

Ayrılma aşaması: tüm veri -> trainval %80 - test %20  
train-val -> train %70 - validation %30

class		class		class	
count	4115	count	1764	count	1470
unique	37	unique	37	unique	37
top	Shiba Inu	top	Abyssinian	top	Sphynx
freq	112	freq	48	freq	40
train		validation		test	

\*\*\*\*\* Class Distribution \*\*\*\*\*

Shiba Inu  
 Japanese Chin  
 Saint Bernard  
 Maine Coon  
 Miniature Pinscher  
 Chihuahua  
 English Setter  
 British Shorthair  
 Samoyed  
 Wheaten Terrier  
 Bengal  
 Basset Hound  
 Pomeranian  
 American Bulldog  
 German Shorthaired  
 Havanese  
 Sphynx  
 Leonberger  
 Great Pyrenees  
 Yorkshire Terrier  
 Russian Blue  
 Persian  
 American Pit Bull Terrier  
 Pug  
 Beagle  
 Birman  
 Ragdoll  
 Abyssinian  
 Keeshond  
 Boxer  
 Scottish Terrier  
 Siamese  
 Newfoundland  
 English Cocker Spaniel  
 Egyptian Mau  
 Staffordshire Bull Terrier  
 Bombay  
 Name: class, dtype: int64

\*\*\*\*\* Class Distribution \*\*\*\*\*

112 Abyssinian  
 112 Ragdoll  
 112 British Shorthair  
 112 Russian Blue  
 112 Chihuahua  
 112 Japanese Chin  
 112 Havanese  
 112 Bengal  
 112 Boxer  
 112 Keeshond  
 112 Persian  
 112 Miniature Pinscher  
 112 Birman  
 112 English Setter  
 112 Yorkshire Terrier  
 112 German Shorthaired  
 112 Pomeranian  
 112 Pug  
 112 Saint Bernard  
 112 American Bulldog  
 112 Wheaten Terrier  
 112 Sphynx  
 112 Scottish Terrier  
 112 Great Pyrenees  
 112 Siamese  
 112 Basset Hound  
 112 American Pit Bull Terrier  
 111 Maine Coon  
 111 Shiba Inu  
 111 Beagle  
 111 Leonberger  
 111 Samoyed  
 110 English Cocker Spaniel  
 110 Newfoundland  
 107 Egyptian Mau  
 106 Staffordshire Bull Terrier  
 103 Bombay  
 Name: class, dtype: int64

\*\*\*\*\* Class Distribution \*\*\*\*\*

Sphynx  
 American Bulldog  
 Siamese  
 Chihuahua  
 American Pit Bull Terrier  
 Basset Hound  
 Wheaten Terrier  
 Scottish Terrier  
 Persian  
 Leonberger  
 Keeshond  
 Maine Coon  
 Bengal  
 Havanese  
 Great Pyrenees  
 Japanese Chin  
 Pug  
 English Setter  
 Samoyed  
 German Shorthaired  
 Beagle  
 Miniature Pinscher  
 British Shorthair  
 Russian Blue  
 Yorkshire Terrier  
 Shiba Inu  
 Saint Bernard  
 Birman  
 Pomeranian  
 Boxer  
 Ragdoll  
 Newfoundland  
 English Cocker Spaniel  
 Abyssinian  
 Staffordshire Bull Terrier  
 Egyptian Mau  
 Bombay  
 Name: class, dtype: int64

Train içerisindeki bazı görüntüler ve maskeleri:



Image



Ground truth mask



Image



Ground truth mask



Image



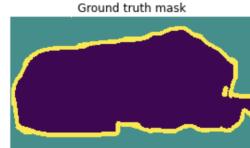
Ground truth mask



Image

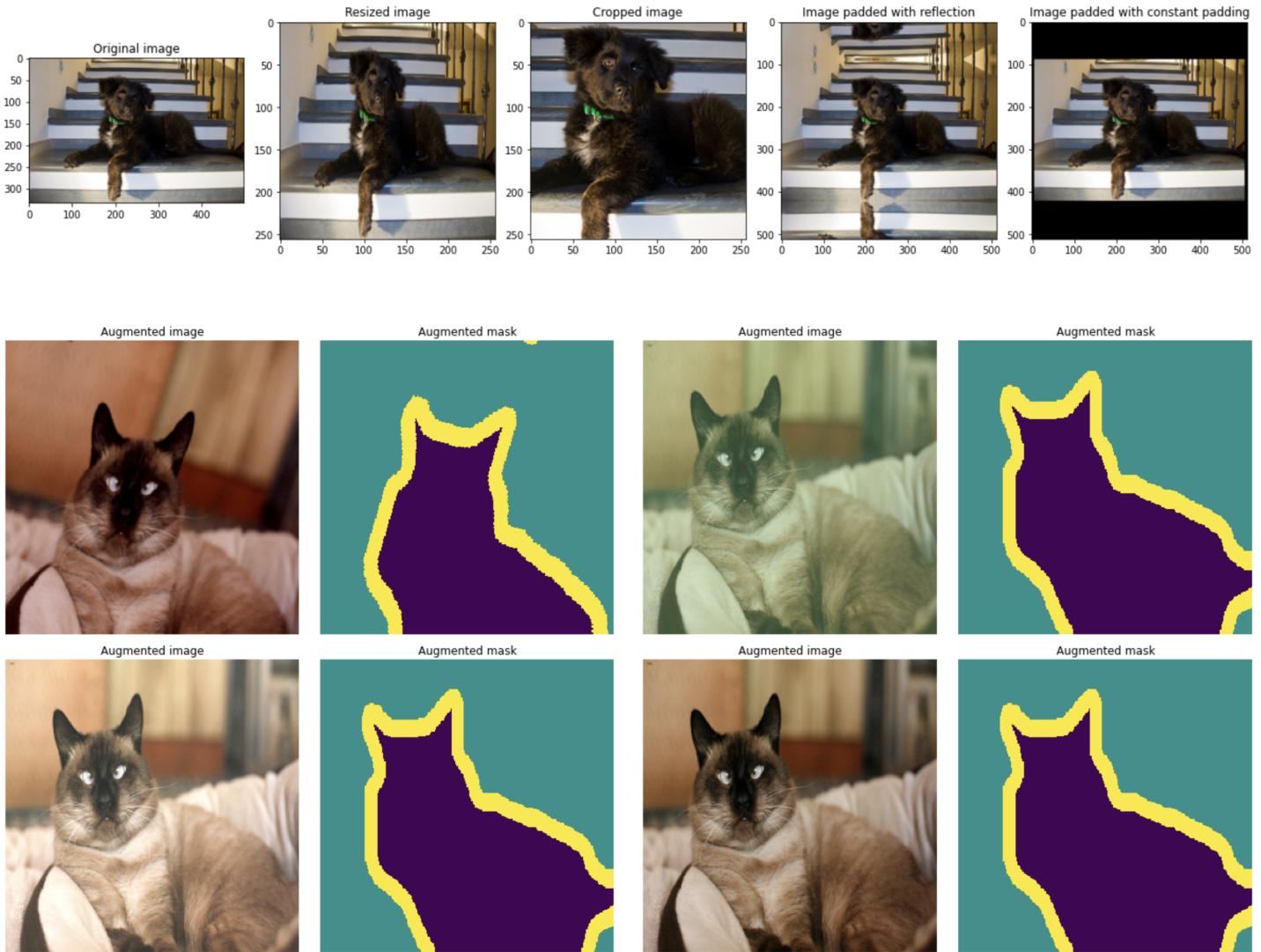


Image



Ground truth mask

## Veri artırılırken uygulanan işlemler:



Custom veri setinin farklı şekillerde nasıl oluşturulabileceğini anlamak için ilgili dokümantasyon aşağıda bulunabilir:

-> Verinin tüm ayarlarını kendim yapılığımdan oldukça uğraştırdı. Veriyi hazırlarken tek bir tensor işleminin bile yanlış yapılması sanki model ya da tahminde problem varmış gibi gözükyor. Veri artırmayı daha sonra iptal ettim. Nedenini anlatıyor olacağım.

- [https://pytorch.org/vision/stable/\\_modules/torchvision/datasets/oxford\\_iiit\\_pet.html#OxfordIIITPet](https://pytorch.org/vision/stable/_modules/torchvision/datasets/oxford_iiit_pet.html#OxfordIIITPet)
- <https://pytorch.org/vision/stable/generated/torchvision.datasets.OxfordIIITPet.html#torchvision.datasets.OxfordIIITPet>
- <https://discuss.pytorch.org/t/how-to-split-dataset-into-test-and-validation-sets/33987>
- <https://blog.paperspace.com/dataloaders-abstractions-pytorch/>
- [https://albumentations.ai/docs/examples/pytorch\\_semantic\\_segmentation/](https://albumentations.ai/docs/examples/pytorch_semantic_segmentation/)
- [https://www.tensorflow.org/datasets/catalog/oxford\\_iiit\\_pet](https://www.tensorflow.org/datasets/catalog/oxford_iiit_pet)

## LinkNet:

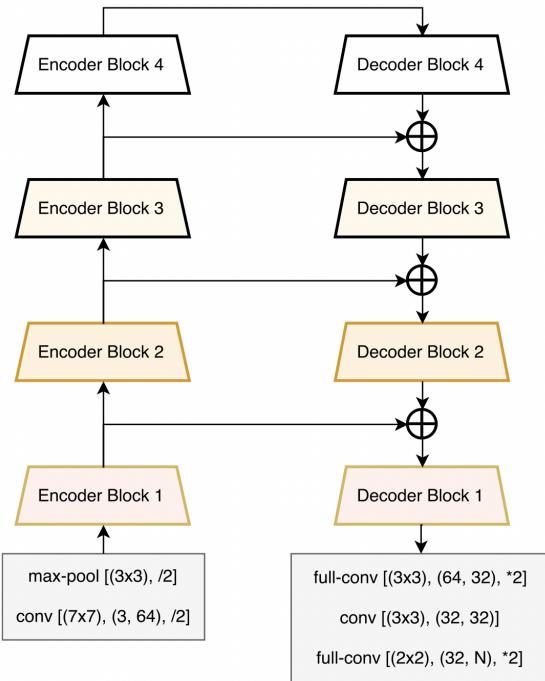


Fig. 1: LinkNet Architecture

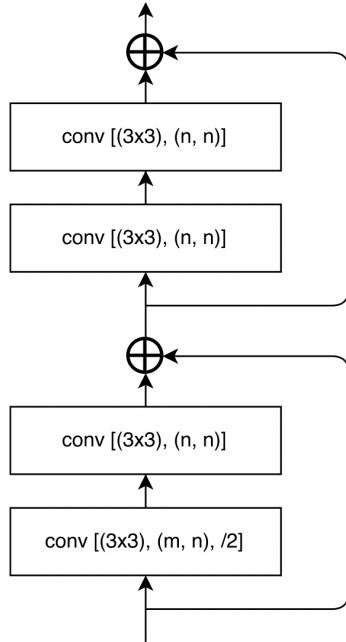


Fig. 2: Convolutional modules in *encoder-block (i)*

```
class LinkNet(nn.Module):
```

```
    def __init__(self, num_classes, num_channels=3, encoder='resnet34'):
        super().__init__()
        if encoder in ['resnet18', 'resnet34']:
            filters = [64, 128, 256, 512]
        else:
            filters = [256, 512, 1024, 2048]
```

```
        res = resnet.resnet34(pretrained=True)
```

```
        self.firstconv = res.conv1
```

```
        self.firstbn = res.bn1
```

```
        self.firstrelu = res.relu
```

```
        self.firstmaxpool = res.maxpool
```

### # Encoder

```
        self.encoder1 = res.layer1
```

```
        self.encoder2 = res.layer2
```

```
        self.encoder3 = res.layer3
```

```
        self.encoder4 = res.layer4
```

### # Decoder

```
        self.decoder4 = DecoderBlock(filters[3], filters[2])
        self.decoder3 = DecoderBlock(filters[2], filters[1])
        self.decoder2 = DecoderBlock(filters[1], filters[0])
        self.decoder1 = DecoderBlock(filters[0], filters[0])
```

### # Final Classifier

```
        self.finaldeconv1 = nn.ConvTranspose2d(filters[0], 32, 3, stride=2)
        self.finalrelu1 = nonlinearity(inplace=True)
        self.finalconv2 = nn.Conv2d(32, 32, 3)
        self.finalrelu2 = nonlinearity(inplace=True)
        self.finalconv3 = nn.Conv2d(32, num_classes, 2, padding=1)
```

```
    def forward(self, x):
```

### # Encoder

```
        x = self.firstconv(x)
```

```
        x = self.firstbn(x)
```

```
        x = self.firstrelu(x)
```

```
        x = self.firstmaxpool(x)
```

```
        e1 = self.encoder1(x)
```

```
        e2 = self.encoder2(e1)
```

```
        e3 = self.encoder3(e2)
```

```
        e4 = self.encoder4(e3)
```

### # Decoder with Skip Connections

```
        d4 = self.decoder4(e4) + e3
```

```
        d3 = self.decoder3(d4) + e2
```

```
        d2 = self.decoder2(d3) + e1
```

```
        d1 = self.decoder1(d2)
```

### # Final Classification

```
        x = self.finaldeconv1(d1)
```

```
        x = self.finalrelu1(x)
```

```
        x = self.finalconv2(x)
```

```
        x = self.finalrelu2(x)
```

```
        x = self.finalconv3(x)
```

```
        return x
```

LinkNet encoder bloğu, resnet katmanlarından olduğu için ayrı bir encoder sınıfı tanımlamadan bu atamaları yapabiliriz.

```

class DecoderBlock(nn.Module):
    def __init__(self, in_channels, n_filters):
        super().__init__()

        # B, C, H, W -> B, C/4, H, W
        self.conv1 = nn.Conv2d(in_channels, in_channels // 4, 1)
        self.norm1 = nn.BatchNorm2d(in_channels // 4)
        self.relu1 = nonlinearity(inplace=True)

        # B, C/4, H, W -> B, C/4, H, W
        self.deconv2 = nn.ConvTranspose2d(in_channels // 4, in_channels
                                        stride=2, padding=1, output_padding=1)
        self.norm2 = nn.BatchNorm2d(in_channels // 4)
        self.relu2 = nonlinearity(inplace=True)

        # B, C/4, H, W -> B, C, H, W
        self.conv3 = nn.Conv2d(in_channels // 4, n_filters, 1)
        self.norm3 = nn.BatchNorm2d(n_filters)
        self.relu3 = nonlinearity(inplace=True)

    def forward(self, x):
        x = self.conv1(x)
        x = self.norm1(x)
        x = self.relu1(x)
        x = self.deconv2(x)
        x = self.norm2(x)
        x = self.relu2(x)
        x = self.conv3(x)
        x = self.norm3(x)
        x = self.relu3(x)
        return x

```

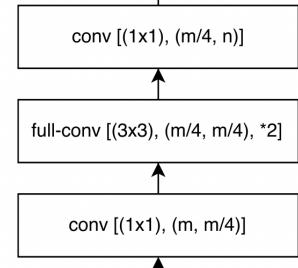


Fig. 3: Convolutional modules in *decoder-block (i)*

Not: Eğitim için veri setini elimle hazırladığım için çok fazla hata alıyordu. Sebebi veri seti değil, model ve diğer kısımlarda gibi gözüktüğü için bulana kadar oldukça zorlandım. Veri setinde, transformlar'da veri artırımı kısmını eğitim için kaldırıldım çünkü Albumentation kütüphanesi yerine torch ile DataLoader'ı tekrar oluşturdum. Bu açıklamanın sebebi, raporda önceki bölümde veri artırımı yapsaydım nasıl olacaktı ile ilgili görüntüler de mevcut. Eğitim sırasında bu artırımları kullanmadım fakat kod içerisinde bulunabilir (zamanım daraldığı için tekrar düzenlemeye vakit ayıramadım).

Normalde Google Drive ve Colab üzerinden çalışıyordum fakat torch GPU'yu bir türlü görmediği için Kaggle GPU P100'de eğitimleri gerçekleştirdim.

```

params = {
    "device": "cuda",
    "lr": 0.001,
    "batch_size": 32,
    "num_workers": 2,
    "epochs": 15,
}

```

*device*: cpu'dan gpu'ya geçmeyi sağlıyor.  
*lr*: learning rate oranı.  
*batch\_size*: dataloader ve eğitim için  
 kullanacağımız batch boyutu  
*num\_workers*: çalışan worker  
*epochs*: epoch miktarı.

Bu projede fantastik yöntem ve parametre denemeleri yapmadım çünkü çok daha kompleks problemler vardı ve default isteneni ancak yetişirebildim.

## UYGULAMA

Eğitim sırasında anlık olarak loss; epoch başına dice score ve pixel accuracy gösteriyorum. Train ve Validation sırasında toplanan her epoch'ta alınan skorları özel bir history dataframe'i içerisinde korudum. Süreç ve history aşağıda bulunabilir. Bir sonraki sayfada da modelin tahminlerini içeren görsellerin karşılaştırması görülebilir.

Dice Coef, genellikle görüntü böülüme yöntemlerinin performansını ölçmek için kullanılır. Nesnelerin ne kadar benzer olduğunun bir ölçüsü olan Dice Coef hesaplayarak algoritmayı doğrulayabiliriz. Dolayısıyla, iki segmentasyonun örtüşme boyutunun iki nesnenin toplam boyutuna bölümür. Doğruluğu açıklamakla aynı terimleri kullanan Dice Score:

$$= \frac{\text{Dice score}}{\frac{2 \cdot \text{number of true positives}}{2 \cdot \text{number of true positives} + \text{number of false positives} + \text{number of false negatives}}}$$

"Dice Score" yalnızca kaç tane pozitif bulduğumuzun bir ölçüsü değildir, aynı zamanda precision'a benzer şekilde yöntemin bulduğu yanlış pozitifleri de cezalandırır. Bu nedenle accuracy'den çok precision'a benzer. Tek fark, yalnızca yöntemin bulduğu pozitifler yerine toplam pozitif sayıya sahip olduğunuz paydadır. Dolayısıyla "Dice Score", algoritmamızın/yöntemimizin bulamadığı pozitifleri de cezalandırıyor. [1]

```
model = create_model(params)
history = fit(model, train_loader, val_loader, params)

Epoch: 1. Train. Loss: 0.514: 100%|██████| 129/129 [00:48<00:00,  2.67it/s]
-> Epoch: 1.0. Train. Dice Score: 0.686 Accuracy: 0.876

Epoch: 1. Train. Loss: 0.467: 100%|██████| 56/56 [00:18<00:00,  3.08it/s]
-> Epoch: 1.0. Validation. Dice Score: 0.686 Accuracy: 0.799

Epoch: 2. Train. Loss: 0.322: 100%|██████| 129/129 [00:48<00:00,  2.68it/s]
-> Epoch: 2.0. Train. Dice Score: 0.860 Accuracy: 0.922

Epoch: 2. Train. Loss: 0.312: 100%|██████| 56/56 [00:17<00:00,  3.12it/s]
-> Epoch: 2.0. Validation. Dice Score: 0.860 Accuracy: 0.894

Epoch: 3. Train. Loss: 0.252: 100%|██████| 129/129 [00:49<00:00,  2.61it/s]
-> Epoch: 3.0. Train. Dice Score: 0.825 Accuracy: 0.893

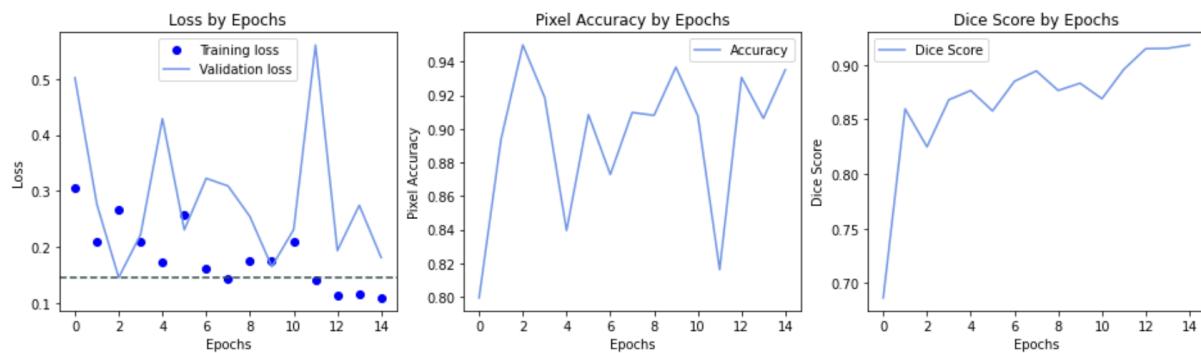
Epoch: 3. Train. Loss: 0.251: 100%|██████| 56/56 [00:19<00:00,  2.91it/s]
-> Epoch: 3.0. Validation. Dice Score: 0.825 Accuracy: 0.950

Epoch: 4. Train. Loss: 0.219: 100%|██████| 129/129 [00:49<00:00,  2.62it/s]
-> Epoch: 4.0. Train. Dice Score: 0.868 Accuracy: 0.921

Epoch: 4. Train. Loss: 0.224: 100%|██████| 56/56 [00:19<00:00,  2.88it/s]
-> Epoch: 4.0. Validation. Dice Score: 0.868 Accuracy: 0.919
```

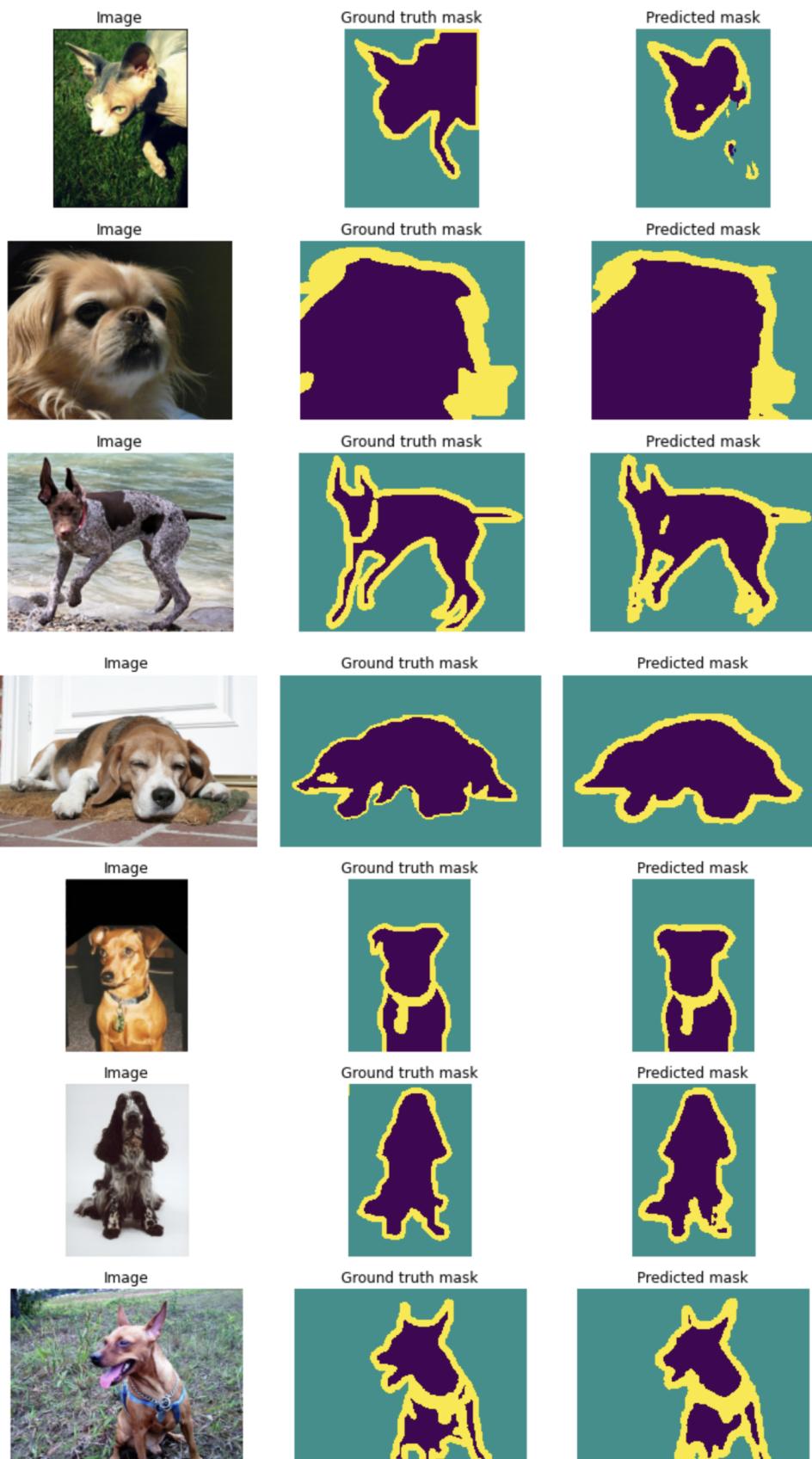
Bu kısım için biraz kafa yorduktan sonra özel bir history dataframe'i oluşturdum. Bu dataframe training loss, training accuracy, training dice score, validation loss, validation accuracy ve validation dice score bilgilerini tutmaktadır. Dice ve Accuracy skorları 15 epochs'ta, sırasıyla, %93.5 ve %89.1'i görebilmektedir. Dalgalanmalar gözükse de sonuçlar oldukça iyi gelmektedir.

history						
	end_loss	end_correct	end_dice	end_val_loss	end_val_correct	end_val_dice
0	0.304524	0.8759532727693257	0.68639874	0.501830	0.79940796	0.63584095
1	0.209122	0.9217818410773025	0.8595894	0.276487	0.89362335	0.8457784
2	0.267043	0.8934655440481085	0.8248506	0.145792	0.9499016	0.9021065
3	0.210113	0.9213208650287829	0.86794734	0.222089	0.91866684	0.8624368
4	0.172435	0.9338828638980262	0.8764341	0.428821	0.8395233	0.7854198
5	0.256679	0.9128185071443256	0.85775906	0.230646	0.90846634	0.8393135
6	0.162709	0.9383295962685032	0.8848971	0.322502	0.8728485	0.78843033
7	0.143645	0.9457614296361019	0.8943864	0.308873	0.909729	0.82195014
8	0.175657	0.9317683169716282	0.87649155	0.254341	0.9080353	0.85166335
9	0.174287	0.9321923506887335	0.883139	0.165149	0.93668747	0.8779297
10	0.209068	0.9259410657380756	0.8689581	0.231060	0.9078331	0.85942745
11	0.142185	0.9430172568873355	0.8958198	0.560401	0.81621933	0.7383856
12	0.113986	0.9555487381784539	0.91474545	0.193678	0.93056107	0.87948525
13	0.115849	0.9541184274773848	0.9151056	0.274470	0.9062195	0.8323369
14	0.109291	0.9574858012952302	0.9181982	0.181377	0.93510056	0.89113975



Aslında bu seviyedeki dalgalanma overfitting olmadığını gösteren önemli göstergelerden birisi olarak yorumlanabilir. Task özelinde Dice Score yaklaşık 90%'a kadar çıkmaktadır ve test üzerinde maskeler tahmin edildiğinde sonuçların başarısı görülebilmektedir.

## Örnek test sonuçları:



## Sonuç:

Bu seviyede karmaşık bir görevi klasik yöntemlerle bu başarıyla yapmamız, hatta sadece yapabilmemiz bile oldukça zordur. Yapılamadığı için yıllarca görüntü işleme yöntemlerinin sınırlı ve görece daha kolay görevlerine ağırlık verildi.

Derin öğrenmeyle birlikte bu kadar zorlu bir görevi sadece 15 epochs'ta %90 gibi bir başarımla, gözle de açık bir şekilde görülebilecek bir segmentasyon görevini gerçekleştirebiliyoruz. Kullandığımız yöntem temelde bu proje kapsamında oldukça karmaşık olduğundan önemli bir kısmı hazır alınsa da tasarım açısından oldukça akıllıca düşünülmüştür. Encoder bloklarına ResNet beslemek ve yine aynı ResNet mantığıyla encoder decoder arasında skip connection'lar oluşturmak bu alandaki diğer modellere kıyaslanabilecek kadar iyi sonuçlar vermektedir. Oldukça sık bir çözüm olarak LinkNet semantik segmentasyon görevlerinde uNet ile birlikte denenecek modeller arasına girmiştir.