PET SEGMENTATION AI PROJECT WITH GUI USING A PRE TRAINED CNN MODEL ON OXFORD PETS DATASET

-

OXFORD PETS VERİ SETİ İLE ÖNCEDEN EĞİTİLMİŞ CNN MODELİ İLE KULLANICI ARAYÜZÜNE SAHİP YAPAY ZEKA PROJESİ

Ömer Burak ÖZGÜR

omerburakozgur1@gmail.com

github.com/omerburakozgur

ÖZET

Proje I Ara Dönem Proje Raporu

U-NET İLE EVCİL HAYVAN GÖRÜNTÜSÜ SEGMENTASYONU

Ömer Burak ÖZGÜR
Haziran 2022, 29 sayfa

Bu uygulama programımıza verilen herhangi bir evcil hayvan görüntüsünün hatasız bir şekilde tespit edilmesini amaçlar. Yazılım çalıştırıldığında görüntü yükleme arayüzü açılmaktadır, kullanıcı görüntüyü yükleyecek ve evcil hayvanın görüntünün neresinde bulunduğu yapay zeka modeli tarafından işaretlenecektir. Bu yazılımın oluşturulması sürecinde kullanılacak olan yöntem teknik ve uygulamalar konusunda bilgi verilmiştir. PyCharm geliştirme ortamında Python ile geliştirilmiştir , Tensorflow, Keras, PyQT, QT Desiner, MatPlotLib ve Numpy kütüphanesi, Arayüz yazılımları, OxfordPets veri kümesi ve U-Net Konvolüsyonel Sinir Ağları hakkında derlemeler yapılmıştır.

Anahtar Sözcükler : Oxford Pets, Evcil Hayvan, Yapay zeka, Derin öğrenme, Tensorflow, Keras, PyCharm, Python, U NET Konvolüsyonel Sinir Ağları.

ABSTRACT

Pet Image Segmentation

Ömer Burak ÖZGÜR
Jun 2022, 29 pages

This application aims to higlight any pet from the image that the user has given to the program. When the software is run, the image loading interface will open, the user will load the image containing cat or dog, and the AI model will try to guess where is the pet and also highlighting it. Information is given about the methods, techniques and applications that will be used in the creation of this software. It has been developed with Python in the PyCharm development environment, Tensorflow, Keras, PyQT, QT Desiner, MatPlotLib and Numpy libraries, Interface software, Oxford Pets dataset and U Net Convolutional Neural Networks have been compiled.

Key Words : Oxford Pets, Artificial intelligence, Deep learning, Tensorflow, Keras, PyCharm, Python, U NET Convolutional Neural Networks.

1. GİRİŞ ve TANITIM

Bu uygulama programa verilen herhangi bir evcil hayvan görüntüsünün hatasız bir şekilde tespit edilmesini amaçlar. Derin öğrenme ve yapay zeka bileşenleri yardımıyla görüntüleri doğru şekilde işaretler. İşaretlenmiş görüntüler kayıt edilebilir.

1.1. Sistemin Çalışma Mantığı

Yapay zeka modelimizi 7000+ tane evcil hayvan görüntüsü ile semantik segmentasyon için özel olarak geliştirilmiş olan U-Net evrişimli sinir ağı mimarisi ile Keras ve Tensorflow kütüphaneleri aracılığı ile eğittik.Program açıldığı zaman bizi bir kullanıcı arayüzü karşılıyor ve görüntü yükleme seçeneği bulunuyor. Görüntü yüklendiğinde bu görüntüyü yapay zeka modelimiz üzerinden belirli işlemlerden geçiyor sonrasında ise model görüntüyü değerlendiriyor, tahmin ediyor ve buna istinaden evcil hayvanın görüntü üzerinde bulunduğu alanları işaretliyor. Görüntüye verilen etiket kullanıcıya iletilir.Kullanıcı isterse işlenmiş görüntüyü kaydedebilir.

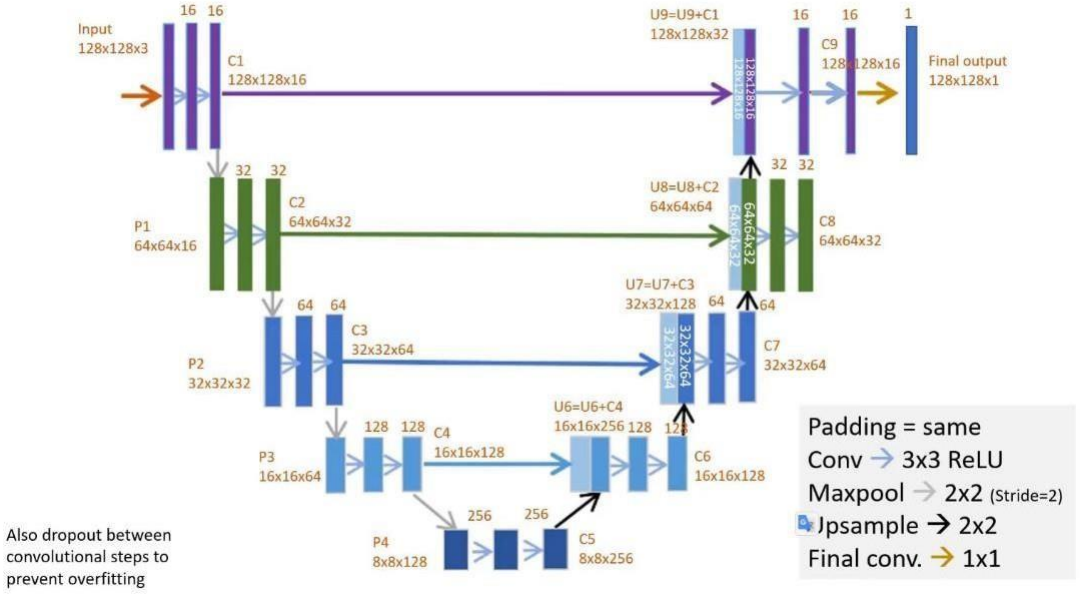
2. PROJE ÇALIŞMASINDA KULLANILAN MATERYALLER

2.1 Konvülasyonel (Evrişimli) Sinir Ağları Nedir ?

Evrişimsel sinir ağları, derin öğrenmenin bir alt dalıdır ve genellikle görsel bilginin analiz edilmesinde kullanılır. Yaygın kullanım alanları resim ve video tanıma, önerici sistemler resim sınıflandırma, tıbbi görüntü analizi ve doğal dil işleme olarak sıralanabilir.

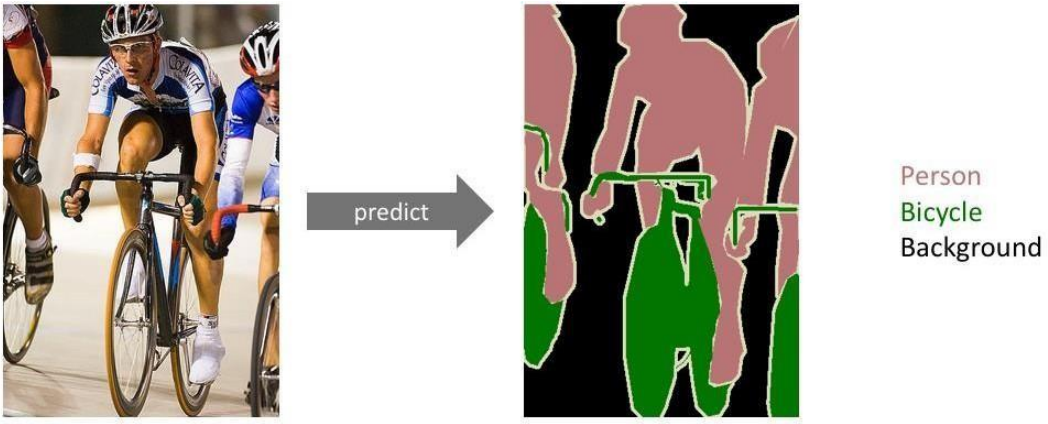
2.2 U-Net Nedir ?

U-Net Biyomedikal alanda görüntüler üzerinde daha verimli, doğru ve hızlı bir semantik segmentasyon uygulaması yapabilmesi üzere geliştirilmiş bir tür Derin Öğrenme, Evrişimli Sinir Ağı yaklaşımıdır, mimarisidir



2.3) Semantik Segmentasyon Nedir ?

Semantik segmentasyonun amacı bir görüntünün her pikselini dahil olduğu sınıfa göre etiketlemektir. Görüntüdeki tüm pikseller tahmin edildiği için bu işe dense prediction (yoğun tahmin) de denir. Önceki uygulamalardan ziyade buradaki segmentasyon işlemi sadece etiketleme ve yerelleştirme işlemleri ile kalmaz. Alınan çıktı her pikselin belli bir sınıfa göre sınıflandırıldığı yüksek çözünürlüklü bir görüntüdür, giriş görüntüsü ile aynı boyuttadır. Yapılan iş piksel boyutundaki bir görüntü sınıflandırmasıdır.



Şekil 4.1 Semantik Segmentasyon

2.4) U-Net'in Sağladığı Kolaylıklar

Evrişimli derin öğrenme modellerinde model eğitimi işlemi için büyük veri kümelerine ihtiyaç duyulur. Bu veri kümelerini toplamak zaman, para ve iyi donanım gerektirir. Bunun yanında elde edilen bu görüntülerin etiketlenmesi konusunda uzman kişilere ve tecrübeye de ihtiyaç duyulur. Bu durumda U-Net klasik modellere göre az sayıda görüntü kullanılması durumunda bile daha başarılı bir sonuç verir bu nedenle araştırmacılara büyük kolaylık sağlar.

2.5) U-Net'in Teknik Açıdan Ayrıntılı Özellikleri

Evrişimli sinir ağı modeli boyunca uygulanan yükseklik ve genişlikteki boyut azaltma işlemi (pooling layer) modelin ikinci yarısında boyut arttırma şeklinde uygulanır. Burada görüntünün çözünürlüğünü arttırmak amaçlanmaktadır. Lokalizasyon için model boyunca yüksek çözünürlüklü öznitelikler ile örneklenen çıktı birleştirilir. Ardışık bir evrişim katmanı daha sonra bu bilgiye dayanarak daha kesin bir çıktı oluşturmayı amaçlamaktadır.

U-Net'in adını aldığı U harfi benzer mimarisinden gelmektedir. Giriş görüntüleri çıkışta segmente edilmiş/bölütlenmiş çıkış haritası olarak elde edilir. Mimarının en özel yanı ikinci yarısıdır. Ağın tamamen bağlı (fully connected) katmanı yoktur. Yalnızca evrişim katmanı kullanılmaktadır. Görüntülerin kesintisiz şekilde bölütlenebilmesi için sınır bölgesindeki pikseller görüntünün çevresine simetrik şekilde eklenir. Bu strateji sayesinde görüntü eksiksiz şekilde bölütlenir. Piksel ekleme yöntemi U-Net modelini büyük görüntülere uygulamak için önemlidir, aksi halde çözünürlük GPU belleğinin kapasitesi ile sınırlı kalacaktır.

Klasik otokodlayıcı mimarisinde problem girişi doğrusal olarak sıkıştırır ve tüm öznitelikleri iletemediği bir darboğaz meydana gelir.U-Net ise kod çözücü (yani ikinci yarıda) tarafında ters evrişim işlemi gerçekleştirir ve buna ek olarak mimarinin kodlayıcı tarafından gelen bağlantılar sayesinde özniteliklerin kaybolmasına sebep olan bu 'darboğaz' probleminin üstesinden gelebilmektedir.
Biyomedikal görüntüde dokudaki en yaygın varyasyon deformasyondur ve gerçekçi deformasyonlar verimli bir şekilde simüle edilebilir. Bu şekilde elimizdeki az sayıda veriyi artırmak için elastik deformasyon yaklaşımı ile öğrenme işleminin daha başarılı olması sağlanır. Tabi ki bölütleme işlemi yalnızca medikal alanda istenmez; İçince örüntü bulunan yer bilimleri ya da uydu görüntülerinden uzaktan algılama sistemleri gibi başka uygulama alanları da vardır.

3. PROJE YAPIMINDA KULLANILAN KÜTÜPHANELER

3.1 Tensorflow Kütüphanesi

Açık kaynak kodlu bir derin öğrenme kütüphanesidir. Esnek yapısı sayesinde, tek bir API ile platform farketmeksizin hesaplamaları, bir veya birden fazla CPU, GPU kullanarak deploy etmenize olanak sağlar. Temelinde Python kullanılarak geliştirilen bu framework, günümüzde Python'ın yanısıra C++, Java, C#, Javascript ve R gibi birçok dili desteklemektedir.

Şekil 4.2 Tensorflow

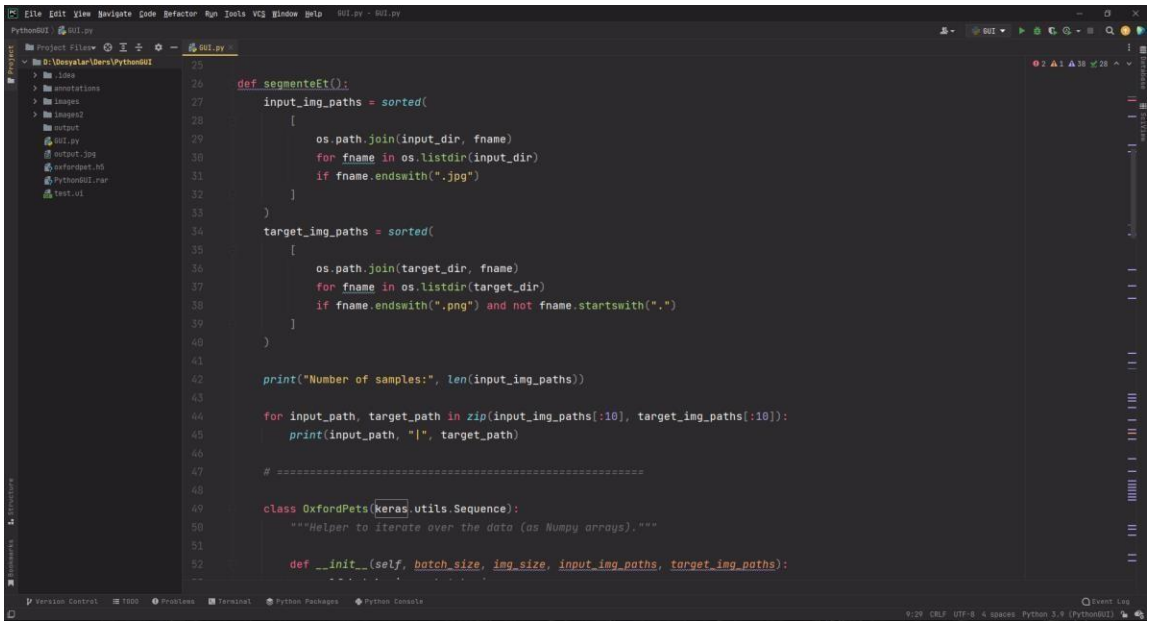
3.2 Keras Kütüphanesi

Keras, Theano veya TensorFlow üzerinde çalışabilen, derin öğrenmeye yönelik minimalist bir Python kütüphanesidir. Araştırma ve geliştirme için derin öğrenme modellerinin olabildiğince hızlı ve kolay uygulanmasını sağlamak için geliştirilmiştir.

Python 2.7 veya 3.5 üzerinde çalışır ve temel frameworklere göre GPU'lar ve CPU'lar üzerinde sorunsuz bir şekilde çalışabilir.

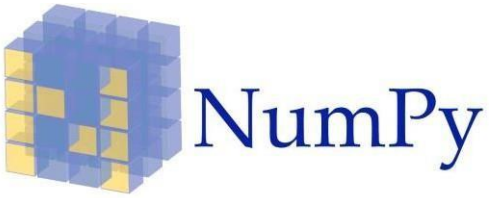
Şekil 4.3 Keras

3.3 Pycharm Geliştirme Arayüzü



3.4 NumPy Kütüphanesi

NumPy (Numerical Python) Veri bilimciler tarafında sıklıkla kullanılan, çok boyutlu dizilerle ve matrislerle çalışmamızı sağlayan ve matematiksel işlemler yapabileceğimiz Python dili kütüphanelerindendir. Numpy ile matematiksel işlemler yapılabilir ve bu işlemler, Python'un dahili dizilerini kullanarak mümkün olana oranla daha verimli ve daha az kodla yürütülür.



3.5 PIL Kütüphanesi

Python Image Library , yani Python Resim Kütüphanesi, Pythonda image işlemlerini kolayca yapabilmek için geliştirilmiş kütüphanedir . 2009'dan beri geliştirilmemiş onun yerine pythonda PIL'ın forklanmış hali Pillow kullanılmaktadır. Bu kütüphaneyi kendi uygulamamızda görüntü düzenlemek ve göstermek için kullandık.

3.6 MatPlotLib Kütüphanesi

MatPlotLib verileri görselleştirmeye yarayan bir Python kütüphanesidir. Veri görselleştirme, karmaşık ve dağınık verileri düzenleyerek kolay anlaşılabilir, yorumlanabilir hale getirmektir. Bir başka ifade ile veri görselleştirme, elimizdeki soyut verilerin görsel hale getirilerek somutlaştırılması ve bundan bir ön bilgi elde edilmesi işidir.

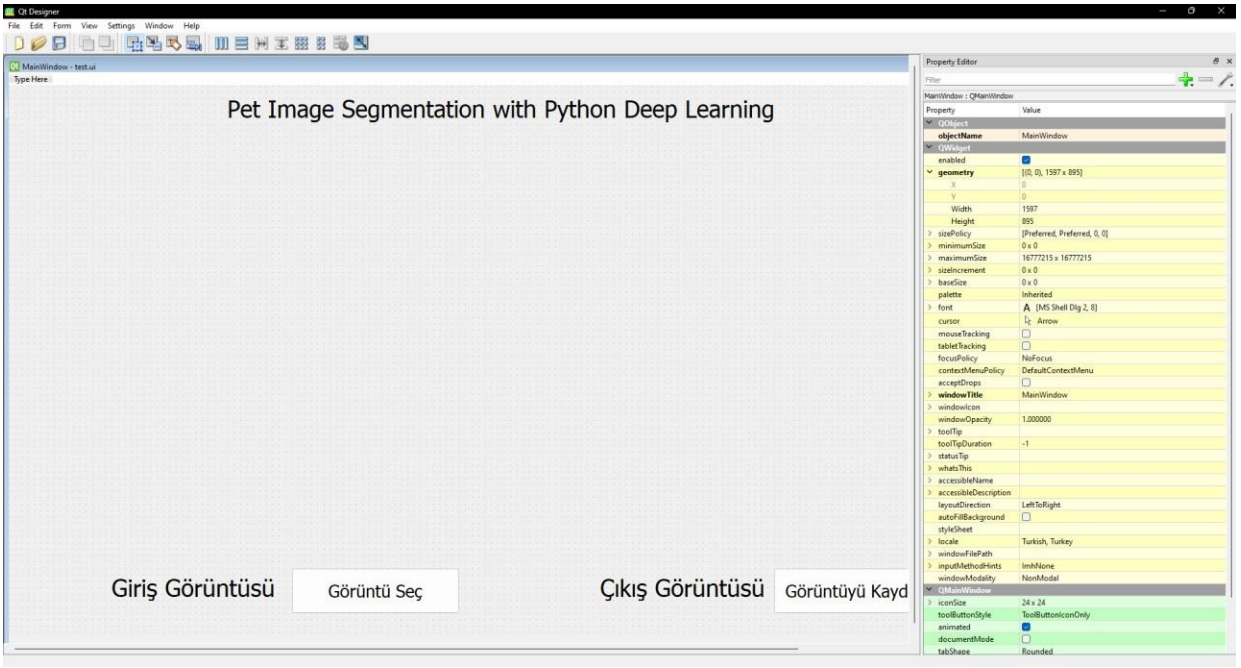
3.7 PyQT5 Kütüphanesi

PyQt, çapraz platform uygulama geliştirmeye yarayan ve C++ ile yazılmış olan Qt kütüphanesinin Python bağlamasıdır. Bir programlama dili değildir. Python ile grafiksel kullanıcı arayüzlü programlar oluşturmamızı sağlar.

PyQt, grafiksel kullanıcı arayüzleri, XML işlemeyi, ağ iletişimini, SQL veritabanlarını ve Qt'de bulunan diğer teknolojileri de kapsayan 620'den fazla sınıfı içerisinde barındırır.

3.8 QT Desigmer Arayüzü

Qt Designer, Qt kütüphanesi ile gelen Widget' lar ile grafiksel kullanıcı arayüzleri tasarlamak için kullanılan Qt aracıdır yada programıdır. Qt Designer kullanarak uzun uzadıya kod yazmadan çok daha efektif, profesyonel ve hızlı bir şekilde tasarımlar oluşturarak Python uygulamamıza daha çok odaklanabiliriz. Dizayn ettiğimiz arayüzü daha sonrasında kullanmak üzere .ui uzantılı olarak kaydetmemiz gerekiyor.



3.9 Shutil Kütüphanesi

Shutil dosyaları kolayca kopyalamamızı sağlayan bir Python kütüphanesidir. Projemizde kullanıcıdan verilen görüntüleri kopyalamak ve kullanıcının istediği yere kaydetmek için kullandık.

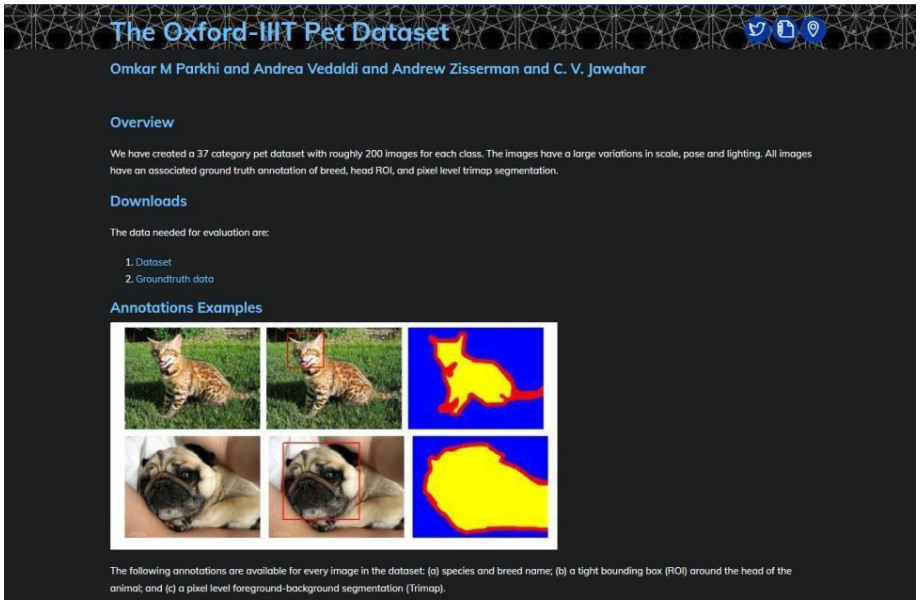
3.10 QFileDialog Kütüphanesi

QFileDialog kullanıcıdan herhangi bir dosya, görüntü vs. almak için kullanılan bir Python kütüphanesidir. Çalışmamızda kullanıcı tarafından görüntü yüklenmesi ve çıktı görüntüsünün kullanıcının istediği yere kaydedilmesi gerektiği için QFileDialog kütüphanesini kullandık.

4. MODELİN EĞİTİLME AŞAMASI

4.1 Veri Setinin Belirlenmesi ve İndirilmesi

Kullanıma açık olan birçok hayvan veri seti bulunmakta fakat biz projemizde sadece kedi ve köpekler üzerine çalışacağımız için veri setimizi Oxford – IIIT Pet Dataset olarak seçtik.



4.2 Veri Setinin İndirilmesi

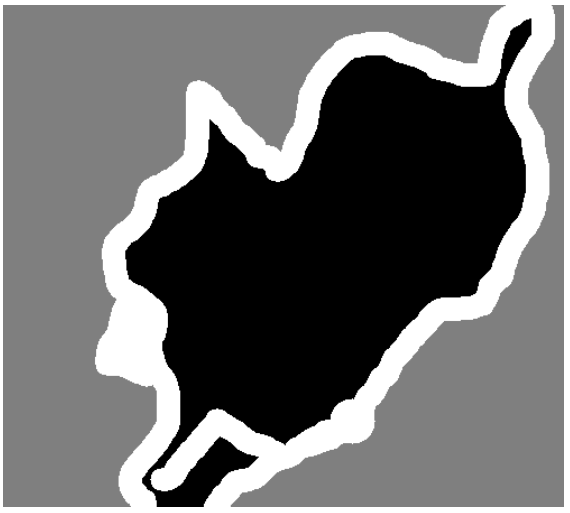
```
!curl -O https://www.robots.ox.ac.uk/~vgg/data/pets/data/images.tar.gz
!curl -O https://www.robots.ox.ac.uk/~vgg/data/pets/data/annotations.tar.gz
!tar -xf images.tar.gz
!tar -xf annotations.tar.gz
```

4.3 Giriş Görüntülerinin ve Segmentasyon Maskelerinin Dizinlerinin Hazırlanması

```
import os

input_dir = "images/"
target_dir = "annotations/trimaps/"
img_size = (160, 160)
num_classes = 3
batch_size = 32

input_img_paths = sorted(
    [
        os.path.join(input_dir, fname)
        for fname in os.listdir(input_dir)
        if fname.endswith(".jpg")
    ]
)
target_img_paths = sorted(
    [
        os.path.join(target_dir, fname)

 for fname in os.listdir(target_dir)
        if fname.endswith(".png") and not fname.startswith(".")
    ]
)

print("Number of samples:", len(input_img_paths))

for input_path, target_path in zip(input_img_paths[:10], target_img_paths[:10]):
    print(input_path, "|", target_path)
```

4.4 Örnek Görüntü ve Görüntü Maskesi

```
from IPython.display import Image, display
from tensorflow.keras.preprocessing.image import load_img
import PIL
from PIL import ImageOps

# Display input image #7
display(Image(filename=input_img_paths[9]))
# Display auto-contrast version of corresponding target (per-pixel categories)
img = PIL.ImageOps.autocontrast(load_img(target_img_paths[9]))
display(img)
```



4.5 Dizilerin Hazırlanması ve Veri Kümelerinin Vektör Haline Getirilmesi

```
from tensorflow import keras
import numpy as np
from tensorflow.keras.preprocessing.image import load_img


class OxfordPets(keras.utils.Sequence):
    """Helper to iterate over the data (as Numpy arrays)."""

    def __init__(self, batch_size, img_size, input_img_paths, target_img_paths):
        self.batch_size = batch_size
        self.img_size = img_size
        self.input_img_paths = input_img_paths
        self.target_img_paths = target_img_paths

    def __len__(self):
        return len(self.target_img_paths) // self.batch_size

    def __getitem__(self, idx):
        """Returns tuple (input, target) correspond to batch #idx."""
        i = idx * self.batch_size
        batch_input_img_paths = self.input_img_paths[i : i + self.batch_size]
        batch_target_img_paths = self.target_img_paths[i : i + self.batch_size]
        x = np.zeros((self.batch_size,) + self.img_size + (3,), dtype="float32")
        for j, path in enumerate(batch_input_img_paths):
            img = load_img(path, target_size=self.img_size)
            x[j] = img
        y = np.zeros((self.batch_size,) + self.img_size + (1,), dtype="uint8")
```

```
        for j, path in enumerate(batch_target_img_paths):
            img = load_img(path, target_size=self.img_size, color_mode="grayscale")
            y[j] = np.expand_dims(img, 2)
            # Ground truth labels are 1, 2, 3. Subtract one to make them 0, 1, 2:
            y[j] -= 1
        return x, y
```

4.6 U-Net Modelinin Hazırlanması

```
from tensorflow.keras import layers


def get_model(img_size, num_classes):
    inputs = keras.Input(shape=img_size + (3,))

    ### [First half of the network: downsampling inputs] ###

    # Entry block
    x = layers.Conv2D(32, 3, strides=2, padding="same")(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    previous_block_activation = x # Set aside residual

    # Blocks 1, 2, 3 are identical apart from the feature depth.
    for filters in [64, 128, 256]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

        # Project residual
        residual = layers.Conv2D(filters, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual]) # Add back residual
        previous_block_activation = x # Set aside next residual

    ### [Second half of the network: upsampling inputs] ###

    for filters in [256, 128, 64, 32]:
        x = layers.Activation("relu")(x)
        x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)
```

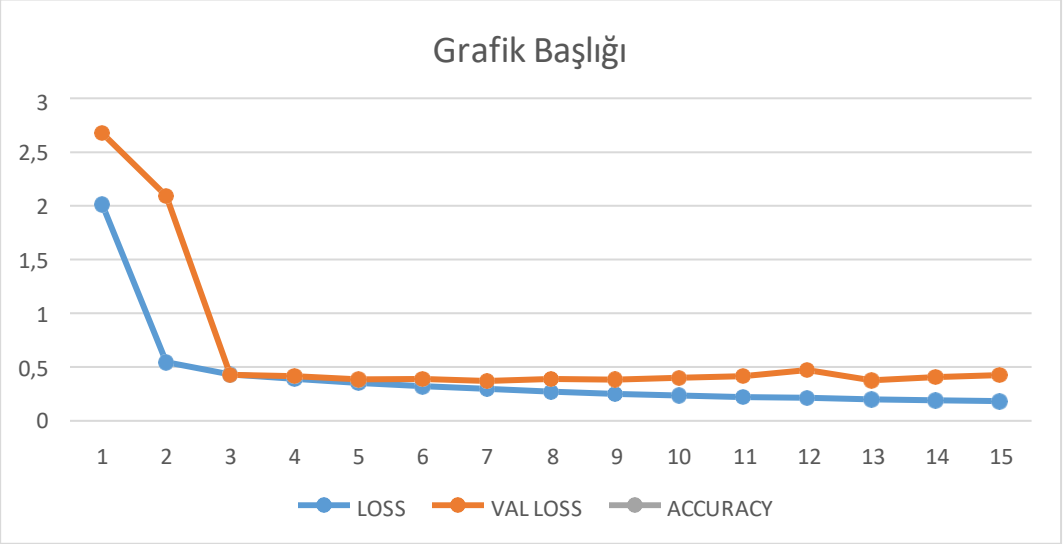
```
        x = layers.Activation("relu")(x)
        x = layers.Conv2DTranspose(filters, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.UpSampling2D(2)(x)

        # Project residual
        residual = layers.UpSampling2D(2)(previous_block_activation)
        residual = layers.Conv2D(filters, 1, padding="same")(residual)
        x = layers.add([x, residual]) # Add back residual
        previous_block_activation = x # Set aside next residual

    # Pixel başına açıklama layer'ı ekleyelim
    outputs = layers.Conv2D(num_classes, 3, activation="softmax", padding="same")(x)

    # Modeli Belirleyelim
    model = keras.Model(inputs, outputs)
    return model
keras.backend.clear_session()

# Modeli Buildleyelim
model = get_model(img_size, num_classes)
model.summary()
```

4.7 Test kümesinin Ayırılması

```
import random

# Görüntülerimizi eğitim ve test için 2 parçaya böldük
val_samples = 1000
random.Random(1337).shuffle(input_img_paths)
random.Random(1337).shuffle(target_img_paths)
train_input_img_paths = input_img_paths[:-val_samples]
train_target_img_paths = target_img_paths[:-val_samples]
val_input_img_paths = input_img_paths[-val_samples:]
val_target_img_paths = target_img_paths[-val_samples:]

# Her ayrım için veri dizileri oluşturduk
train_gen = OxfordPets(
    batch_size, img_size, train_input_img_paths, train_target_img_paths
)
val_gen = OxfordPets(batch_size, img_size, val_input_img_paths, val_target_img_paths)
```

4.8 Modelin Eğitilmesi

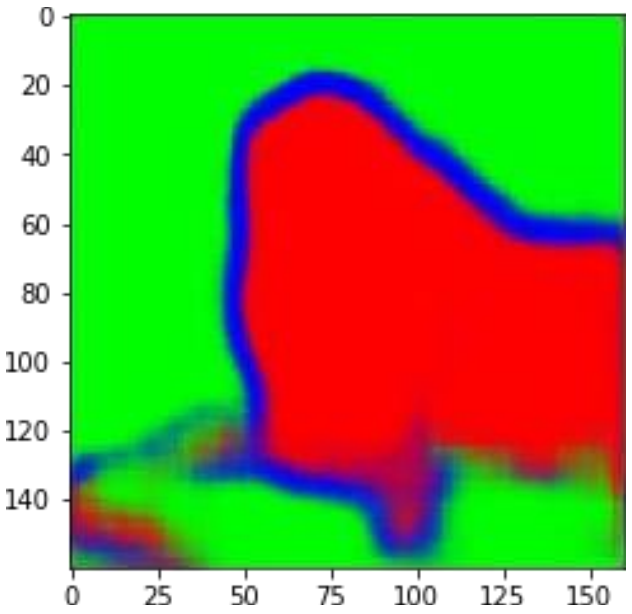
```
# Modeli eğitim işlemi için ayarladık
# Kategorisel crossentropy için "Sparse" versiyonunu kullanıyoruz
# çünkü hedef veri tipi tam sayı.
model.compile(optimizer="rmsprop", loss="sparse_categorical_crossentropy")

callbacks = [
    keras.callbacks.ModelCheckpoint("oxford_segmentation.h5", save_best_only=True)
]

# Modeli eğitiyoruz, Her epoch işlemi sonunda doğrulama yapıyor.
epochs = 15
model.fit(train_gen, epochs=epochs, validation_data=val_gen, callbacks=callbacks)
```



4.9 Modelin Kullanılması

```
# Test setindeki tüm görüntülerin bulunduğu bir tahmin dizisi oluşturalım
import tensorflow as tf
import matplotlib.pyplot as plt
model = tf.keras.models.load_model("oxfordpet.h5")
val_gen = OxfordPets(batch_size, img_size, val_input_img_paths, val_target_img_paths)
val_preds = model.predict(val_gen)


def display_mask(i):
    """Quick utility to display a model's prediction."""
    mask = np.argmax(val_preds[i], axis=-1)
    mask = np.expand_dims(mask, axis=-1)
    img = PIL.ImageOps.autocontrast(keras.preprocessing.image.array_to_img(mask))
    display(img)
```

```
# 10. Test görüntüsü için sonuçları gösterelim #10
i = 3
img = PIL.ImageOps.autocontrast(load_img(val_target_img_paths[i]))
# Display input image
display(Image(filename=val_input_img_paths[i]))
display(img)
plt.imshow(val_preds[i])
plt.show()
```



Model Tahmini

4.10 Modelin Kaydedilmesi ve Yüklenmesi

Modeli daha sonrasında kullanabilmek için kaydetmemiz gerekiyor. Farklı uzantı tipleri ile kaydedilebiliyor fakat ben h5 tipini tercih ettim.

```
model.save('savedmodel.h5')
model = tf.keras.models.load_model("oxfordpet.h5")
```

4.11 Colab Not Defterinin PyCharm'a aktarılması

Colab ortamındaki kodları olduğu gibi PyCharm'da kullanamıyoruz o yüzden belli başlı değişiklikler ile aktarma işlemini yaptık.

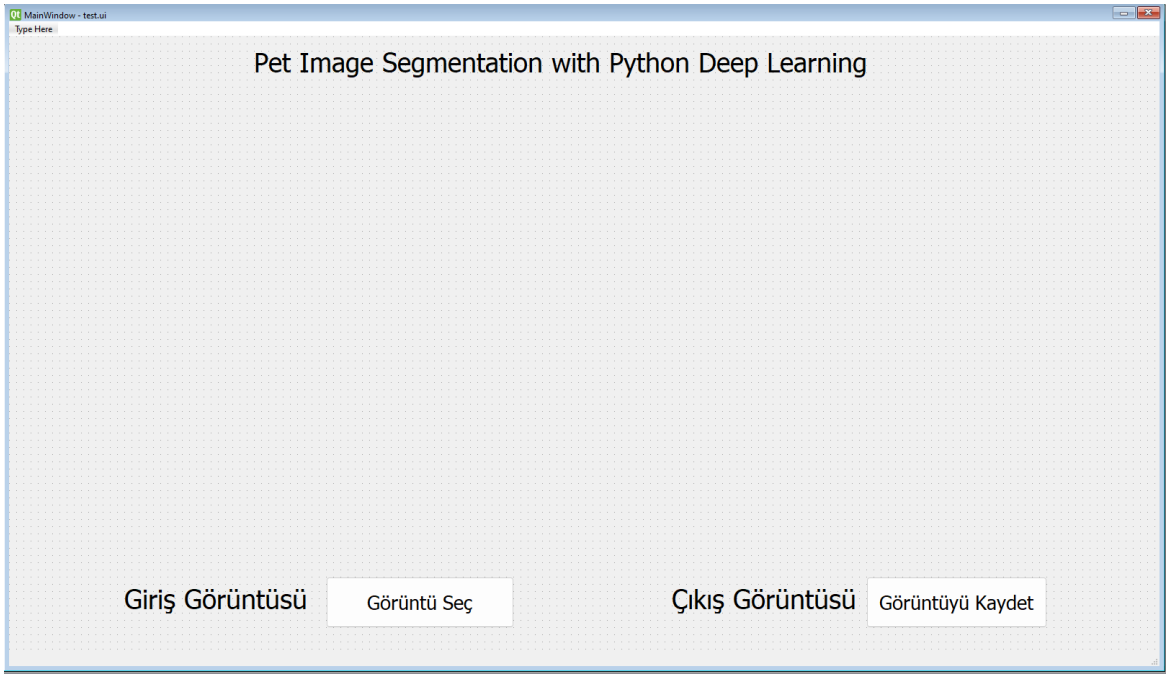
```
def segmenteEt():
    input_img_paths = sorted(
        [
            os.path.join(input_dir, fname)
            for fname in os.listdir(input_dir)
            if fname.endswith(".jpg")
        ]
    )
    target_img_paths = sorted(
        [
            os.path.join(target_dir, fname)
            for fname in os.listdir(target_dir)
            if fname.endswith(".png") and not fname.startswith(".")
        ]
    )

    print("Number of samples:", len(input_img_paths))
```

```
for input_path, target_path in zip(input_img_paths[:10], target_img_paths[:10]):
    print(input_path, "|", target_path)

# ============================================================

class OxfordPets(keras.utils.Sequence):
    """Helper to iterate over the data (as Numpy arrays)."""

    def __init__(self, batch_size, img_size, input_img_paths, target_img_paths):
        self.batch_size = batch_size
        self.img_size = img_size
        self.input_img_paths = input_img_paths
        self.target_img_paths = target_img_paths

    def __len__(self):
        return len(self.target_img_paths) // self.batch_size

    def __getitem__(self, idx):
        """Returns tuple (input, target) correspond to batch #idx."""
        i = idx * self.batch_size
        batch_input_img_paths = self.input_img_paths[i: i + self.batch_size]
        batch_target_img_paths = self.target_img_paths[i: i + self.batch_size]
        x = np.zeros((self.batch_size,) + self.img_size + (3,), dtype="float32")
        for j, path in enumerate(batch_input_img_paths):
            img = load_img(path, target_size=self.img_size)
            x[j] = img
        y = np.zeros((self.batch_size,) + self.img_size + (1,), dtype="uint8")
        for j, path in enumerate(batch_target_img_paths):
            img = load_img(path, target_size=self.img_size, color_mode="grayscale")
            y[j] = np.expand_dims(img, 2)
            # Ground truth labels are 1, 2, 3. Subtract one to make them 0, 1, 2:
            y[j] -= 1
        return x, y

# ============================================================

val_samples = 100
# random.Random(1337).shuffle(input_img_paths)
# random.Random(1337).shuffle(target_img_paths)
train_input_img_paths = input_img_paths[:-val_samples]
train_target_img_paths = target_img_paths[:-val_samples]
val_input_img_paths = input_img_paths[-val_samples:]
val_target_img_paths = target_img_paths[-val_samples:]

# Instantiate data Sequences for each split
train_gen = OxfordPets(
    batch_size, img_size, train_input_img_paths, train_target_img_paths
)
val_gen = OxfordPets(batch_size, img_size, val_input_img_paths, val_target_img_paths)

model = tf.keras.models.load_model("oxfordpet.h5")

val_gen = OxfordPets(batch_size, img_size, val_input_img_paths, val_target_img_paths)
val_preds = model.predict(val_gen)
# Display results for validation image #10
i = 0
# Display ground-truth target mask
img = PIL.ImageOps.autocontrast(load_img(val_target_img_paths[i]))
img = PIL.ImageOps.fit(img, img_size)
#plt.imshow(img)
#plt.show()
plt.imshow(val_preds[i])
#plt.axis("off")
plt.savefig("output.jpg")
plt.show()
```

5. KULLANICI ARAYÜZÜ

5.1 Arayüzün Dizayn Edilmesi

Kullanıcı arayüzümüzü QT Designer aracı ile dizayn ettik. Arayüz görüntü seçmek ve kaydetmek için birer tuş ve görüntüleri göstermek için de 2 tane label nesnesi bulunduruyor. Dizayn işleminden sonra arayüzümüzü sonrasında kullanmak için kaydettik.



5.2 Arayüzün Python Ortamına Yüklenmesi

PyQt5 Kütüphanesinde bulunan "uic" modülünü projemize dahil ettikten sonra arayüz kodlarımızın Main Window kısmında arayüzümüzü yükleyebiliriz. Bu kullanım tipinde arayüz üzerinde değişiklik yapmak oldukça kolaydır, istediğimiz değişikliği .ui dosyasında yapınca projemize de uygulanmış olacaktır.

```
from PyQt5 import uic

uic.loadUi("test.ui",self)
```

5.3 Arayüz Nesnelerinin Kodlanması

Projemizde kullanıcı ile aktif iletişime geçecek arayüz nesneleri üzerinde değişiklik yapabilmek için bu nesneleri tanımlamamız gerekmektedir.

```
self.btnSec = self.findChild(QPushButton, "btnGoruntuSec")
self.btnKaydet = self.findChild(QPushButton, "btnGoruntuKaydet")
self.lblCikis  = self.findChild(QLabel, "lblGirisGoruntusu")
self.lblGiris  = self.findChild(QLabel, "lblCikisGoruntusu")
```

5.4 Butonların On-Click Eventlerinin Yazılması

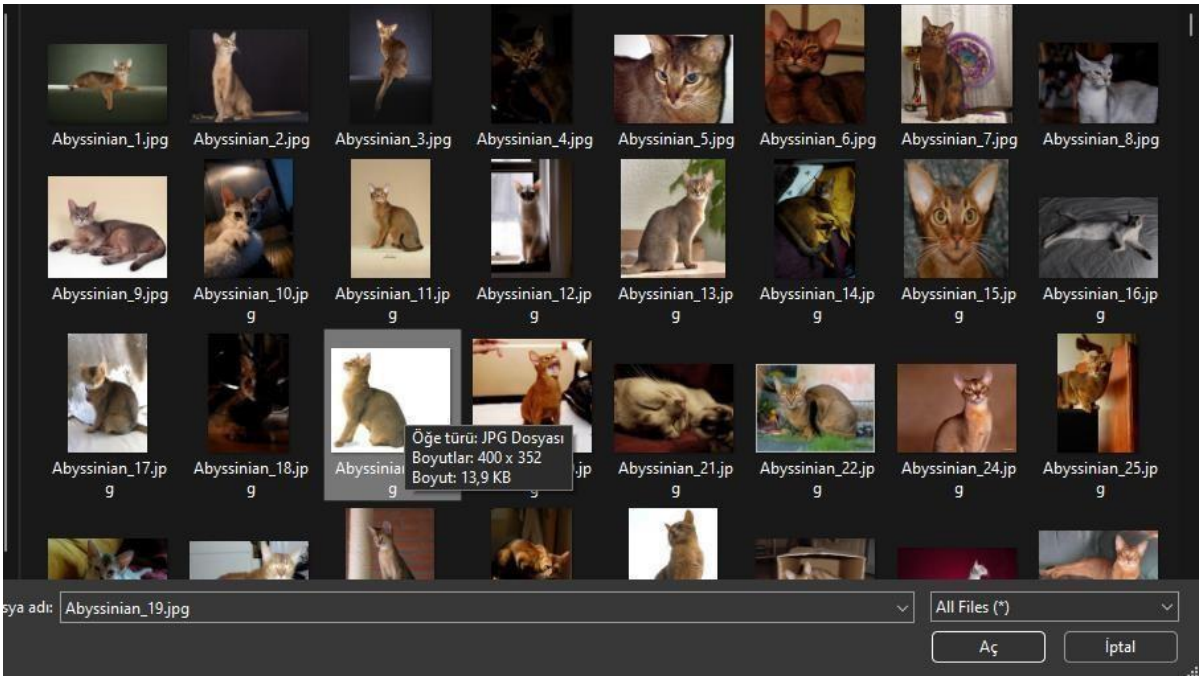
Butonlarımızın üzerlerine tıklanıldığında kullanıcı ile etkileşime geçebilmeleri için on-click eventlerinin yazılması gerekmektedir. Aşşağıdaki kodda buton nesnelerimizin "clicked" özelliklerine çalıştıracakları fonksiyonları ekliyoruz.

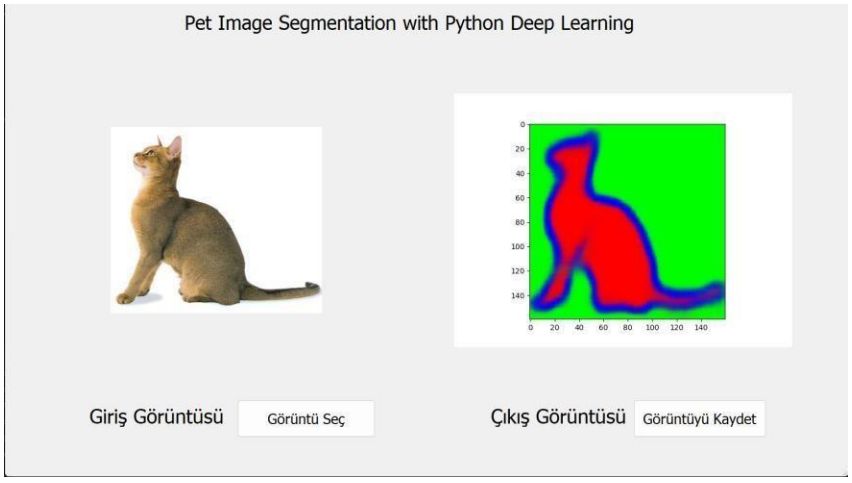
```
self.btnSec.clicked.connect(self.goruntuSec)
self.btnKaydet.clicked.connect(self.goruntuKaydet)
```

5.4.1 Görüntü Seçme Fonksiyonu

Görüntü Seç butonuna tıklandığında bizi bir File Dialog karşılamakta, buradan istediğimiz evcil hayvan görüntüsünü seçebiliriz. Görüntü seçildikten sonra görüntü model'e servis edilir yapay zeka modeli tahminini yapar, tahmin edilmiş görüntüyü görüntüyü output.jpg olarak kaydeder ve görüntüler PixMap'e çevirilir. Sonrasında arayüz üzerinde gösterilirler.

```
def goruntuSec(self):
    fname = QFileDialog.getOpenFileName(self, "Open File", "D:\\Dosyalar\\Ders\\PythonGUI\\images2",
"All Files (*)")
    imageLocation = fname
    #Open The Image
    self.pixmap = QPixmap(fname[0])
    self.pixmap2 = QPixmap("D:\\Dosyalar\\Ders\\PythonGUI\\output.jpg")
    #Add pic to label
    self.lblCikis.setPixmap(self.pixmap)
    self.lblGiris.setPixmap(self.pixmap2)
    shutil.copy(fname[0], "D:\\Dosyalar\\Ders\\PythonGUI\\images")
    segmenteEt()
```

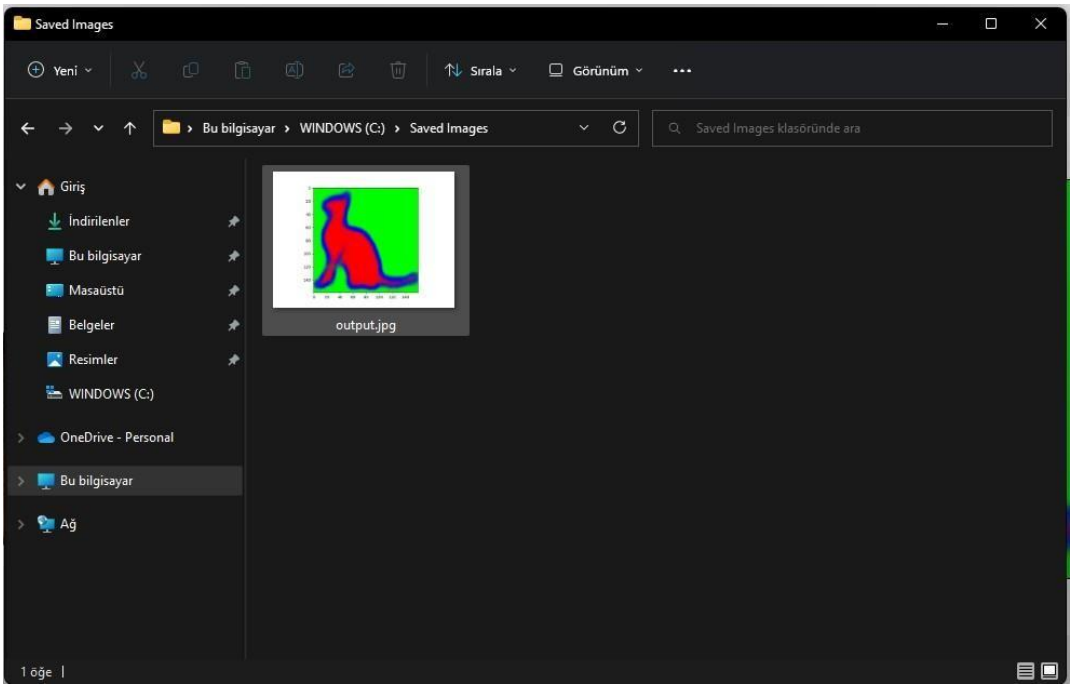




5.4.2 Görüntü Kaydetme Fonksiyonu

Görüntü kaydet butonuna basıldığı zaman yine bir File Dialog bizi karşılar. Kullanıcı dosyayı kaydetmek istediği dizin'i seçer ve önceden kaydetmiş olduğumuz output.jpg görüntüsü kullanıcının istediği lokasyona kaydedilir.

```
def goruntuKaydet(self):
    saveLocation = QFileDialog.getExistingDirectory(self, "Select Save Location", "c:\\")
    shutil.copy("D:\\Dosyalar\\Ders\\PythonGUI\\output.jpg", saveLocation)
```



5.5 Arayüzün Tamamı

```
import sys
import os.path
import PIL
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import shutil
from PIL import ImageOps
from tensorflow import keras
from tensorflow.keras.preprocessing.image import load_img
from PyQt5 import uic
from PyQt5.QtGui import QPixmap
from PyQt5.QtWidgets import QMainWindow, QApplication, QPushButton, QLabel, QFileDialog

input_dir = "images/"
target_dir = "annotations/trimaps/"
img_size = (160, 160)
num_classes = 3
batch_size = 32
imageLocation = ""

def segmenteEt():


class UI(QMainWindow):
    def __init__(self):
        super(UI, self).__init__()
        #Load UI
        uic.loadUi("test.ui",self)
```

```
        #Define Widgets
        self.btnSec = self.findChild(QPushButton, "btnGoruntuSec")
        self.btnKaydet = self.findChild(QPushButton, "btnGoruntuKaydet")
        self.lblCikis = self.findChild(QLabel, "lblGirisGoruntusu")
        self.lblGiris = self.findChild(QLabel, "lblCikisGoruntusu")

        #Dropdown Box
        self.btnSec.clicked.connect(self.goruntuSec)
        self.btnKaydet.clicked.connect(self.goruntuKaydet)

        #Show the App
        self.show()

    def goruntuSec(self):
        fname = QFileDialog.getOpenFileName(self, "Open File", "D:\\Dosyalar\\Ders\\PythonGUI\\images2", "All Files
(*)")
        imageLocation = fname
        #Open The Image
        self.pixmap = QPixmap(fname[0])
        self.pixmap2 = QPixmap("D:\\Dosyalar\\Ders\\PythonGUI\\output.jpg")
        #Add pic to label
        self.lblCikis.setPixmap(self.pixmap)
        shutil.copy(fname[0], "D:\\Dosyalar\\Ders\\PythonGUI\\images")
        segmenteEt()
        self.lblGiris.setPixmap(self.pixmap2)


    def goruntuKaydet(self):
        saveLocation = QFileDialog.getExistingDirectory(self, "Select Save Location", "c:\\")
        shutil.copy("D:\\Dosyalar\\Ders\\PythonGUI\\output.jpg", saveLocation)
        print("Save Location and File Name: " + saveLocation)


app = QApplication(sys.argv)
UIWindow = UI()
app.exec_()
```


6. SONUÇ

6.1 Proje Özeti ve Geliştirme Süreci

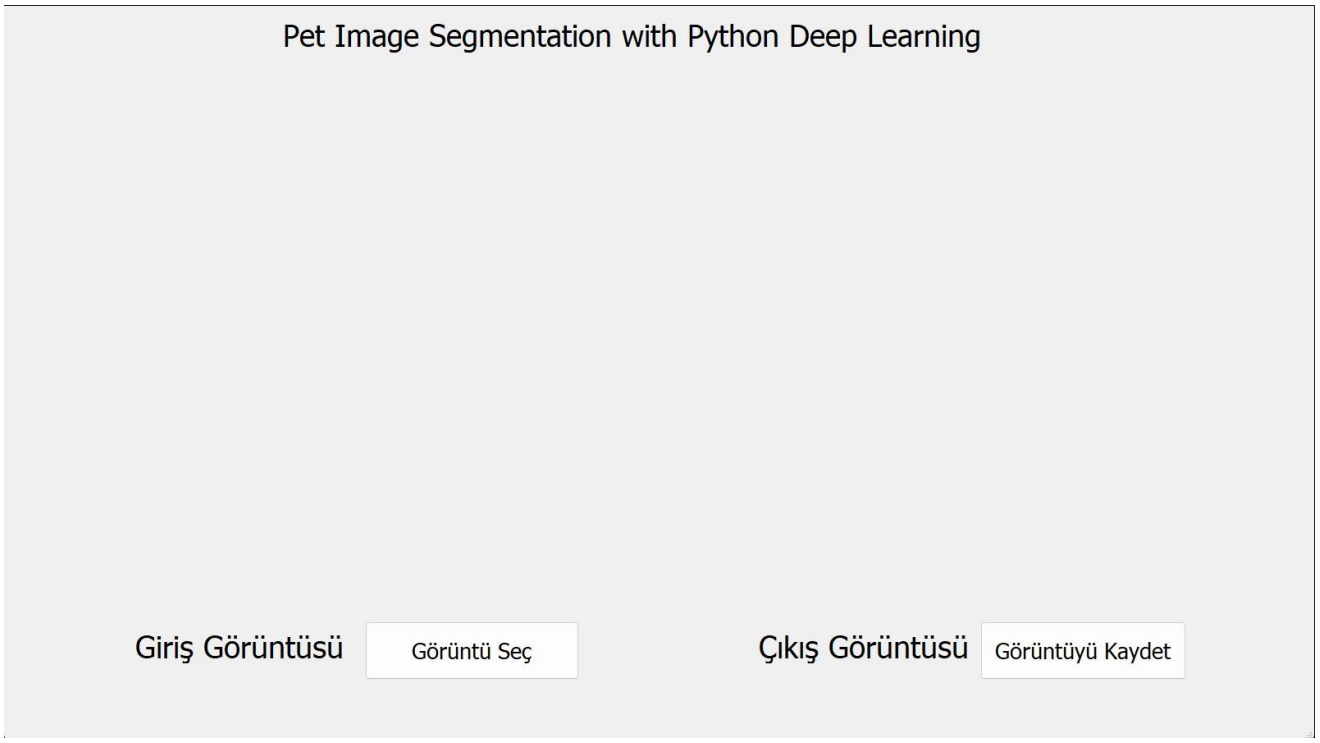
Bu noktada projemizi genel anlamı ile hayata geçirmiş bulunmaktayız. Bu bölümde projemizde yaptığımız işleri özet olarak ele alacağız ve bir sonuç çıkartacağız.

Projemize başlarken ilk olarak yaptığımız şeylerden birisi uygun yapay zeka modelini ve veri setine seçmek oldu. Seçimimizi U-Net evrişimli sinir ağı mimarisi olarak yaptık ve buna istinaden Oxford Pets veri kümesini seçtik. Sonrasında ise U-Net ve kullandığımız kütüphaneler ve framework'ler üzerinde araştırmamızı yaptık. Devamında ise verilerimizi yapay zeka modelini eğitmek üzere Google Colab ortamında servis ettik. Bunun sonucunda yapay zeka modelimizi elde ettik. Modelimizi kaydettikten sonra ise Colab ortamındaki çalışmamızı bilgisayar ortamında PyCharm arayüzüne entegre ettik. Sonrasında ise projemizi kullanılabilir bir hale getirmek için kullanıcı arayüzünü oluşturma işlemlerine başladık. QT Designer üzerinden PyQT5 kütüphanesi aracılığı ile arayüzümüzü oluşturduktan sonra projemiz kullanılmaya hazır hale geldi. Şimdi ise projemizin kullanım sürecini görseller ile anlatalım.

6.2 Programın Kullanım Aşamaları

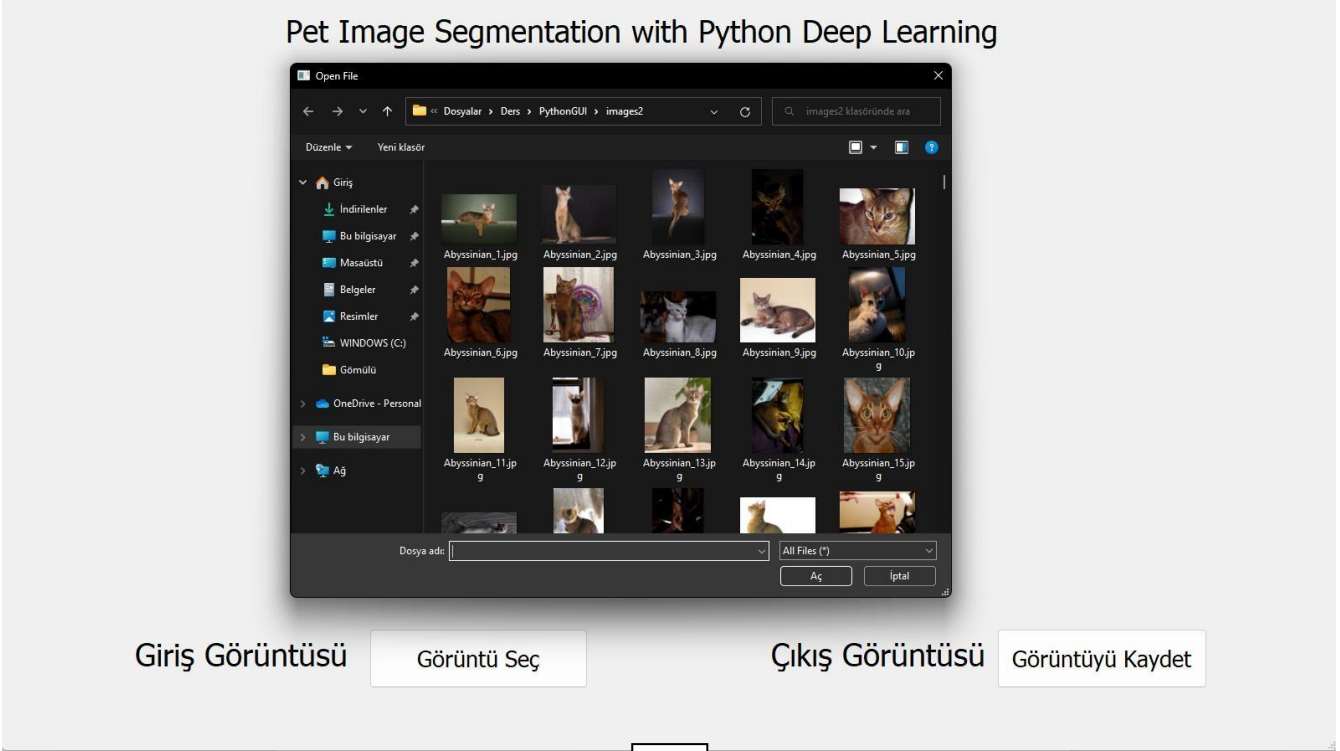
6.2.1 Açılış Ekranı

Kullanıcının proje ile ilk olarak etkileşime geçtiği yer.



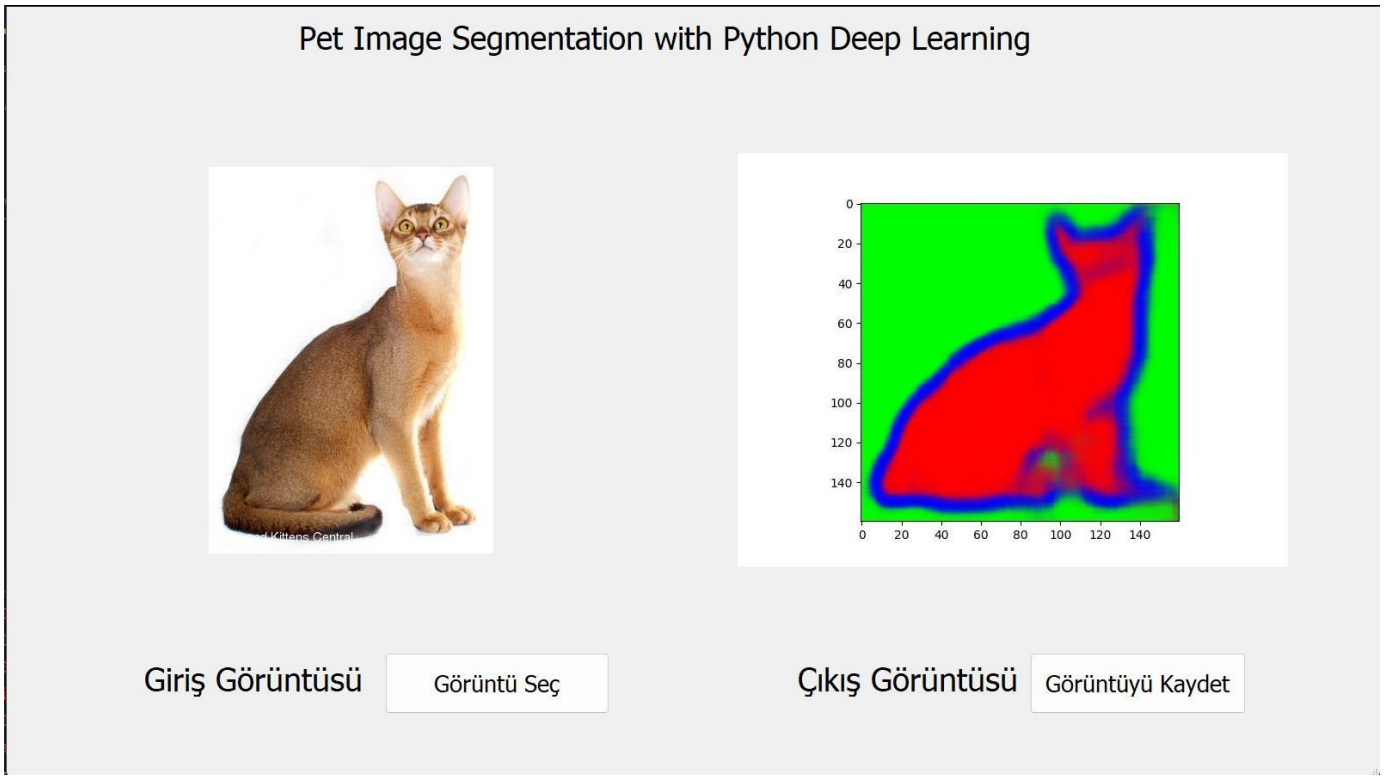
6.2.2 Görüntü Seçme Ekranı

Kullanıcının programı kullanmak üzere istediği görüntüyü seçebildiği seçim arayüzü.



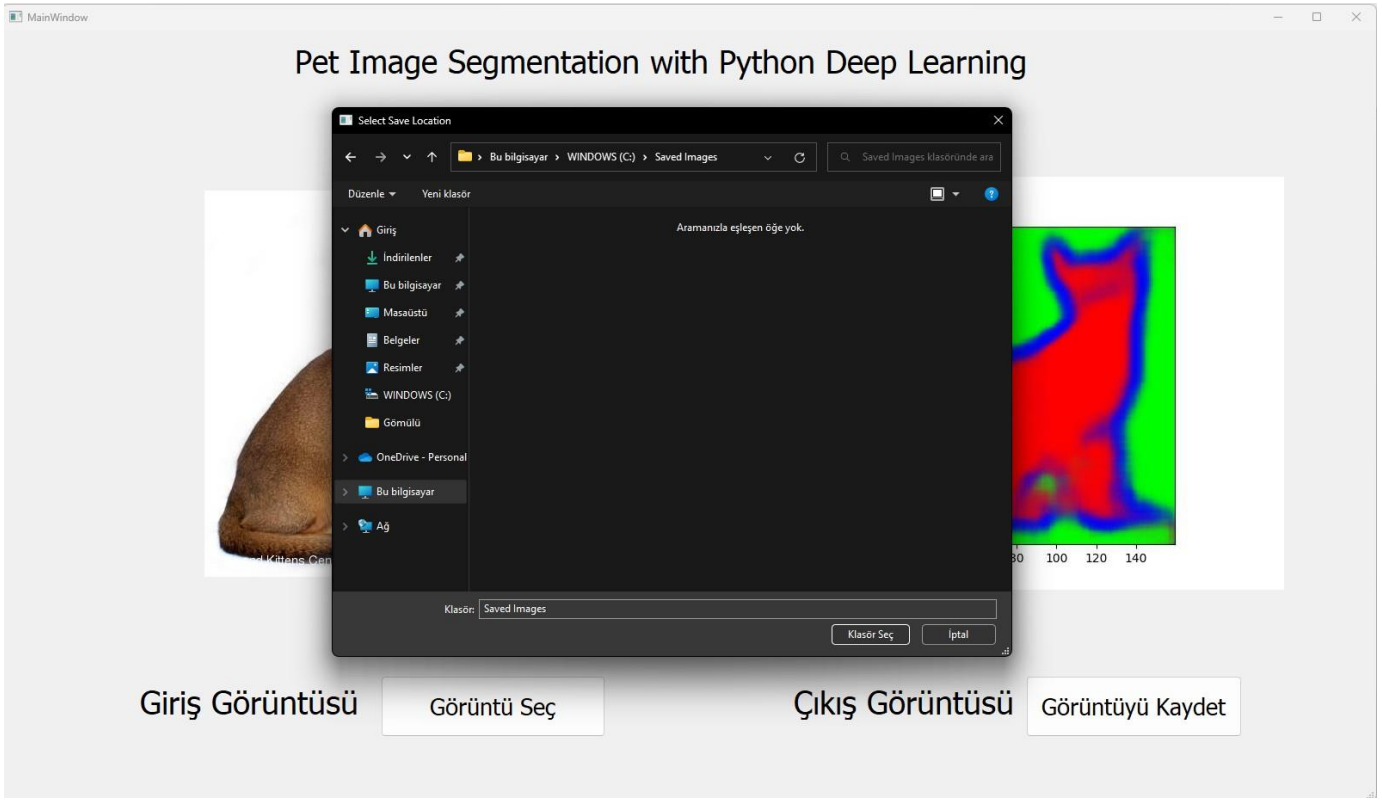
6.2.3 Sonucu Görüntüleme

Kullanıcının yapay zeka tahmin sonucunu görüntülediği kısım.



6.2.4 Çıktıyı Dışarı Aktarma

Kullanıcının yapay zeka tahmin sonucunu görüntülediği kısım.



7. KAYNAKLAR

1. Main owner of the AI model and article page / AI modelinin ana sahibi ve makale sayfası:
https://twitter.com/fchollet
https://keras.io/examples/vision/oxford_pets_image_segmentation/
2. Olaf Ronneberger, Philipp Fischer, and Thomas Brox "U-Net: Convolutional Networks for Biomedical Image Segmentation" University of Freiburg, Germany
3. Dr. Ayyüce Kızrak Ph.D, Görüntü Bölütleme için Derin Öğrenme: U-Net
4. Omkar M Parkhi and Andrea Vedaldi and Andrew Zisserman and C. V. Jawahar The Oxford-IIIT Pet Dataset
5. DigitalSreeni Youtube, Image Segmentation using U-Net - Part1 (What is U-net?)
6. Wikiwand.com, Convolutional Neural Networks
7. Özgür Doğan, Image Segmentation Nedir?
8. Birhan K. PyQt Nedir? Qt Designer Nedir? Python Arayüz Tasarımı
9. Codemy.com John ELDER, Build An Image Viewer App - PyQt5 GUI Thursdays #30
10. Connor Shorten, Image segmentation with a U-Net-like architecture - Keras Code Examples