

DERİN ÖĞRENME TABANLI BEYİN TÜMÖRÜ SEGMENTASYONU TESPİT SİSTEMİ

Proje Tematik Alanı

Bilgi ve İletişim Teknolojileri

Sağlık

Kastamonu Üniversitesi

Bilgisayar Mühendisliği Bölümü

Sema Nur KAYA

174410027

Bahri ÇÖKMEZ

174410035

Proje Danışmanı

Dr. Öğr. Üyesi Yasemin GÜLTEPE

İçindekiler

1. Özet.....	3
2. Problem Tanımı	
2.1. MRI Sekansları.....	3
2.2 MRI Sekansları.....	3
3. Projede Kullanılan Yöntem.....	3
3.1. Veri Seti.....	4
3.2. Kullanılacak Sinir Ağı.....	4
3.3. Eğitim(Training).....	4
3.4. Derin Öğrenmesi Adımları.....	4
4. Programlama Bilgileri.....	6
5. Görev Aşamaları.....	6
6. Sonuç.....	6
7. Maliyet Raporu.....	7
8. Kaynaklar.....	7

1. Özet

Yapay sinir ağlarının daha fazla gizli katman içererek büyük veriler üzerinde karmaşık kalıpları kolayca öğrenmesi ile artık günümüzde birçok problem daha hızlı çözülebilir hale geldi. Artık birçok farklı sektörde bile derin öğrenme modelleri kullanılarak daha hızlı çözümler üretebiliyoruz. Özellikle son zamanlarda sağlık alanında bu modellerin uygulamaya alınmasıyla birlikte akıllı sistemler, birçok hastalığı önceden tespit edebilmekte veya uzman bir kişinin göremediği ayrıntıları gözden kaçırmamaktadır. Medikal görüntüleme alanında bir proje geliştirmeye ve MRI ile beyin tümörü tespiti ve segmentasyonu üzerine çalışmaya karar verdik. Bu projeyi seçmemizdeki temel amaç; medikal görüntüleme alanına giriş yapmak ve fikir edinmek, farklı derin öğrenme modellerini uygulayarak anlamak ve geliştirmek, tecrübe kazanmak. Kanser gibi hastalıkların önceden tespiti için bu teknolojilerin kullanılması, birçok insan hayatını kurtarabilir veya farklı çözümler üretebiliriz. Mevcut veri setleri ve kullanılan derin öğrenme algoritmaları ile oluşturulan model, en yüksek performansı vermek üzere kurulmaktadır.

Beyin tümörü, dünya genelinde en çok ölüme neden olan ve hızlı ilerleyen kanser türlerinden biri olarak kabul edilmektedir. Bu durumun en önemli nedeni olarak hastaların ileri evrede doktora gitmesi ve geciken teşhis gösterilmektedir. Günümüz yapay zeka teknolojilerinden derin öğrenme ile kanser teşhisine yönelik objektif ve daha hassas sonuçlar elde edilmiştir. Derin öğrenmesi, karmaşık örüntü algılama ve veriye dayalı karar verebilme özellikleri ile ele alınan problemin çözümünü kendi kendine öğrenebilen bilgisayar algoritmalarının genel adıdır.

Bu çalışmada, beyin tümörü segmentasyonu tespiti için güncel bir veri kümesi üzerinde literatürde bulunan çeşitli derin öğrenmesi sınıflandırma yöntemleri kullanılarak karşılaştırmalı bir analiz yapılacaktır. Analiz sonuçları, gerçekleştirilecek derin öğrenmesi yöntemlerinin kullanılan veri kümesi üzerinde ne kadar başarılı olacağının sonuçlarını gösterecektir. Elde edilen sonuç değerleri literatürdeki benzer çalışmalarla kıyaslanarak beyin tümörü segmentasyonun tespitine çözüm için hangi derin öğrenmesinin daha başarılı olabileceğine dair öngörü sahibi olmayı mümkün kılacaktır.

2. Problem Tanıma

Beyin tümörleri ciddi hastalıklardır. Ancak beyninde tümör taşıyıp hayatına hiçbir şey yokmuş gibi devam eden şanslı insanlar da bulunmaktadır. Bu şanslı insanlar sadece yüzde 1’lik bir orana sahip olsalar da zamanında tespit edildiğinde ameliyatla bu tümörlerden kurtulmak mümkündür. Beyin tümörü grubunda ameliyat edilebilen grup sadece %10’luk dilimdir ve bu çok önemlidir. Çünkü %90’lık kesimi için birincil tedavi aşaması olan cerrahi tedavinin uygulanamadığı anlamına gelmektedir. Beyin tümörü beyindeki hücrelerin veya kitlenin anormal büyümesidir. Her tümör öldürücü olmasa da beyin tümörlerinde, beyin dokusunun istisnai bir durumu vardır. İyi huylu tümörler de beyin kafatası içinde kapalı bir odada yer aldığından öldürücü olabilir. Bu nedenle beyin tümörlerinin tümü öldürücü olmasa da mutlaka kontrol altında tutulmalı ve doğru müdahale edilmelidir. Proje sayesinde beyin tümörüne sahip olup olmadığı bilinmeyen bir hastanın, tümörünün olup olmadığını belirlemek üzere hasta hakkında bir öngörü elde edilecektir. Bu tümör türü ile ilgili mevcut veriseti üzerinde derin öğrenmesi yöntemleri kullanılacaktır. derin öğrenmesi yöntemleriyle ortaya çıkacak modelin erken teşhis başta olmak üzere birçok avantajı da beraberinde getireceği söylenebilir.

2.1.MRI Sekansları

MRI, nöroloji ve beyin cerrahisinde en sık kullanılan görüntüleme tekniklerinden biridir. Canlıların içyapısını görüntüleme amacıyla daha çok tıpta kullanılan bir yöntemdir. Beyin, omurilik ve damar anatomisinin ayrıntılarını sunar ve her üç düzlemde (Sagital düzlem, uzunlamasına düzlem, koronal düzlem) anatomiye görselleştirme avantajına sahiptir.

Sekans, MRI sinyali oluşturmak için uygulanan radyo frekans pulse, gradient ve sinyal toplama süreçlerini gösteren veridir. MRI’da 3 temel sekans vardır.

En sık kullanılan MRI sekansları T1, T2 ve Flair olarak adlandırılır. Bu her bir sekansın özellikleri birbirinden farklıdır.

T1 — Kısa zamanlamalı radyo frekansları kullanılır. Kontrast madde sonucunda görüntüyü daha parlak ve anatomiye daha iyi gösterir. Subakut kanamayı (methemoglobin) iyi gösterir.

T2 — Uzun zamanlamalı radyo frekanslar kullanılır. Kontrast ve parlaklık dokunun özelliğine göre belirlenir.

Flair — Çok uzun zamanlamalı radyo frekanslar kullanılır. T2'ye biraz benzerdir fakat görüntü daha karanlıktır ve anormallikler daha parlaktır.

Biz de eğiteceğimiz veriyi bu özellikleri araştırarak kararlaştırdık ve Flair görüntüleri üzerinden ilerleme kararı aldık. Flair üzerinden gitmemizin ana sebebi tümörün diğer sekanslara oranla daha belirgin gözükmesidir ve bu şekilde modelimizi daha kolay ve başarılı eğitebiliriz diye düşündük.

2.2.Segmentasyon (Görüntü Bölütleme)

Bir görüntüyü farklı özelliklerin tutulduğu anlamlı bölgelere ayırmayı bölütleme olarak tarif edebiliriz. Yani, her piksel için etiketler çıkartılır ve tahminler yapılarak çıkarımlar sağlanır. Görüntü bölütlemenin backbone (ana yapısını oluşturan) kısmını oluşturan birkaç önemli birkaç model:

- AlexNet
- GoogLeNet
- VGG-16
- ResNet

3. Projede Kullanılan Yöntem

3.1 Veri Seti

Derin öğrenmesi yöntemleri kadar kullanılan veri setindeki öznitelikler de bu yöntemlerin başarısını etkileyen önemli unsurlardan biridir. MRI görüntülerini nereden toplayacağımızı öğrenmek için ilk olarak internette bu alanda yapılan çalışmaları araştırdık ve bunun sonucunda BraTS adlı yarışmanın verilerini kullanmaya karar verdik. BraTS adlı bu

yarışma Pensilvanya Üniversitesi tarafından 2013 yılından beri her yıl düzenlenmektedir. MRI görüntülerini almak için üniversiteye başvurduk ve kendileri verileri paylaşmaya kabul ettiler. Biz de bu çalışmada BraTS 2019 MRI görüntülerini kullandık. BraTS, 259 adet örnek ve 1295 adet öznitelikten oluşmaktadır. Öznitelik listesinde öne çıkan öznitelikler; Baş ağrısı, baş ağrısına eşlik eden bulantı ve kusma, nöbetler, hareket ve denge sorunları gibi faktörlerdir.

3.2.Kullanılacak Sinir Ağı

U-Net: U-Net, Almanya'daki Freiburg Üniversitesi Bilgisayar Bilimleri Bölümü'nde, biyomedikal görüntü segmentasyonu için geliştirilen bir evrimsel sinir ağıdır. Ağ tamamen evrişimli sinir ağları üzerine kuruludur ve mimarisi, daha az eğitim görüntüsü ile çalışmak ve daha hassas bölümlere sağlamak için genişletildi.

3.3.Eğitim(Training)

U-Net'in verilen verileri ne kadar iyi modellediğini anlamak için kayıp /yitim (loss) fonksiyonları uyguladık.

Kayıp/Yitim (Loss) hesabı yaklaşımları

Standart ikili çapraz entropi ve Dice:

Dice yöntemi, özellikle biyomedikal görüntülerde sıklıkla kullanılan bir performans kriteridir. Biz de U-Net modelini eğitirken Dice yönteminden yararlandık.

Bölütleme performansını değerlendirmek için kullanılır ve piksel temelli bir ölçüttür. Hedeflenen matris ile tahmin edilen matris arasındaki örtüşen piksel oranıdır. Dice hesabı ile ilişkilidir.

3.4. Derin Öğrenmesi Adımları

MRI görüntüleri NIFTI formatında geliyor ve bu formattaki görüntüleri okumak için SimpleITK kütüphanesini kullandık. Eğer flair haricinde farklı sekans kullanılırsa, normalizasyon algoritması kullanılması gerektiğini de belirtelim. Her bir MRI görüntüsünde görüntüsünde toplam 600 slayt var ve her görüntünün 60–120 arasında olan slaytları eğitim verisi olarak kullandık. Çünkü bu aralıklarda tümör varsa direkt belli olmaktadır. Eğitime hazır hale getirmek ve görüntülerden anlamsal bir şeyler çıkarmak içinse görüntülerin

ortalamasını ve standart sapmasını alarak Sıfır Ortalama Normalizasyon / Birim Norm Normalizasyon (Zero Mean Normalization / Unit Norm Normalization, Boyut indirgeme işlemi uygulanan verisetinin üzerinde) uygulanacaktır.

Son olarak da verilerimizi **x.npy** ve **y.npy** formatındaki dosyalarda saklıyoruz ve verimiz artık eğitime hazır.

Derin öğrenmesi için oldukça önemli olan verilerin boyutunu (öznitelik) azaltma (indirgeme) yöntemi uygulanacaktır. Bu projede boyut azaltma yönteminde Temel Bileşen Analizi (Principle Components Analysis, PCA) ve Doğrusal Diskriminant Analizi (Linear Discriminant Analysis, LDA) algoritmaları kullanılacaktır. Seçilen bu algoritmalar LCDS verileri üzerinde denenecektir. Oluşan yeni BraTS verilerinin etiketli verileri tekrar sınıflandırma algoritmaları ile eğitilip ardından etiketsiz verileri ile test edilecektir.

Projenin sonucu olarak; derin öğrenmesi tabanlı beyin tümörü tespitinde, bir öngörüye sahip olabilmek için derin öğrenmesi için en iyi hangi algoritmanın olacağına karar verilecektir.

4.Programlama Bilgileri

Derin öğrenmesi tabanlı beyin tümörü segmentasyonu tespit sistemi, derin öğrenmesi alanında son zamanlarda artan bir şekilde kullanılan Python programlama dili ile geliştirilecektir. İşlemler Windows 8 işletim sisteminde, Python'unun 3.7 sürümü kullanılarak gerçekleştirilecektir. Burada Jupyter ve Colab üzerinden çalışmalarımı yürüttüm Ayrıca Python üzerinden veri işleme ve raporlama işlemleri için ise Anaconda Enterprise kullanılacaktır.

5.Görev Aşamaları

Proje başlangıç tarihi: 01.10.2020

Proje bitiş tarihi: 02.06.2018

6.Sonuç

Gerçekleştirilecek bu proje sayesinde hızlı ilerleyen kanser türleri arasında beyin tümörünün erken teşhisi için model çıkararak erken teşhisin yapılabilirliği artacaktır. Böylece erken müdahale için bir fırsat doğmuş olacaktır. Proje sonucunda elde edilecek başarı performans değerleri, bu konuda derin öğrenme yöntemlerinden en etkin olanlarının bulunması ve problemin teşhisine yönelik kullanılması planlanmaktadır. Bu nedenle proje sonucunda beyin tümörü tespiti alanında derin öğrenmesi algoritmaları ile başarılı sonuçlar elde edildiği takdirde hem doktorlara hem de hastalara fayda sağlayacağı düşünülmektedir. Proje sonucunda; beyin

tümörü teşhisinde, erken tanı ve teşhis imkanı sağlayarak çok daha önemli ve başarılı bir konuma ulaşılması hedeflenmektedir.

7.Maliyet Raporu

ALINAN DONANIM:	FİYAT
Kingston 8GB 1600mhz DDR3 Notebook Sodimm Ram	400
Ram Takım Ücreti	50
Udemy Eğitim Seti	36
Sandisk 1 TB Extream PRO V2 Harici SSD(2000MB okuma /2000MB Yazma)	2.800
SanDisk 64gb ixpand Mini usb 3.0 Flash Bellek	190
TOPLAM	3.476

9.KODLAR VE AÇIKLAMASI

Zipli dosyalara ulaşabilmemiz için şu kod parçacığını yazmamız gerekiyor. Buradaki gibi yazdığımızda zipli dosyalarımıza erişmiş olacağız.

```
In [ ]: import glob

In [2]: files=glob.glob('C:\\Users\\user1\\Desktop\\deneme1\\*.*' + '**\\*flair.nii.gz',recursive=True)

In [3]: files
```

Zipli dosyalarımıza ulaştığımıza da şu şekilde anlayabiliriz;


```
In [3]: files
Out[3]: ['C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_10_1\\BraTS19_2013_10_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_11_1\\BraTS19_2013_11_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_12_1\\BraTS19_2013_12_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_13_1\\BraTS19_2013_13_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_14_1\\BraTS19_2013_14_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_17_1\\BraTS19_2013_17_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_18_1\\BraTS19_2013_18_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_19_1\\BraTS19_2013_19_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_20_1\\BraTS19_2013_20_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_21_1\\BraTS19_2013_21_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_22_1\\BraTS19_2013_22_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_23_1\\BraTS19_2013_23_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_25_1\\BraTS19_2013_25_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_26_1\\BraTS19_2013_26_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_27_1\\BraTS19_2013_27_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_2_1\\BraTS19_2013_2_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_3_1\\BraTS19_2013_3_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_4_1\\BraTS19_2013_4_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_5_1\\BraTS19_2013_5_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_7_1\\BraTS19_2013_7_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_AAB_1\\BraTS19_CBICA_AAB_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_AAG_1\\BraTS19_CBICA_AAG_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_AAL_1\\BraTS19_CBICA_AAL_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_AAP_1\\BraTS19_CBICA_AAP_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ABB_1\\BraTS19_CBICA_ABB_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ABE_1\\BraTS19_CBICA_ABE_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ABM_1\\BraTS19_CBICA_ABM_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ABN_1\\BraTS19_CBICA_ABN_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ABO_1\\BraTS19_CBICA_ABO_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ABY_1\\BraTS19_CBICA_ABY_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ALN_1\\BraTS19_CBICA_ALN_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ALU_1\\BraTS19_CBICA_ALU_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ALX_1\\BraTS19_CBICA_ALX_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_AME_1\\BraTS19_CBICA_AME_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_AMH_1\\BraTS19_CBICA_AMH_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ANG_1\\BraTS19_CBICA_ANG_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ANI_1\\BraTS19_CBICA_ANI_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ANP_1\\BraTS19_CBICA_ANP_1_flair.nii.gz',
'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_CBICA_ANV_1\\BraTS19_CBICA_ANV_1_flair.nii.gz']
```

Burada kullanacağımız veri setinin yollarına ulaşmış olduk.

```
In [4]: len(files)
Out[4]: 102

In [5]: import skimage.io as io

In [6]:

In [7]: files[1]
Out[7]: 'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_11_1\\BraTS19_2013_11_1_flair.nii.gz'

In [8]: pip install SimpleITK

Requirement already satisfied: SimpleITK in c:\\users\\user1\\anaconda3\\envs\\tensor_gpu\\lib\\site-packages (2.0.2)
Note: you may need to restart the kernel to use updated packages.

In [9]: img= io.imread(örnek,plugin='simpleitk')
```

Dosyalarımıza ulaştıktan sonra dosyamızın uzunluğuna baktık. Sonuç olarak ta bize 102 olarak bir değer döndürdü.

Elimizdeki emar görüntülerini numpy'e çevirmek için skimage kütüphanesini kullanarak numpy hale getirmiş olduk.

Burada numpy olarak oluşturduğumuz dosyanın yoluna ulaşabiliyor muyuz diye kontrolünü sağladık. Bunun içinde örnek nesnesi ile çağırdık.

```
In [9]: img= io.imread(örnek,plugin='simpleitk')
```

```
In [10]: img.shape
```

```
Out[10]: (155, 240, 240)
```

```
In [11]: img.dtype
```

```
Out[11]: dtype('int16')
```

```
In [12]: import matplotlib.pyplot as plt
```

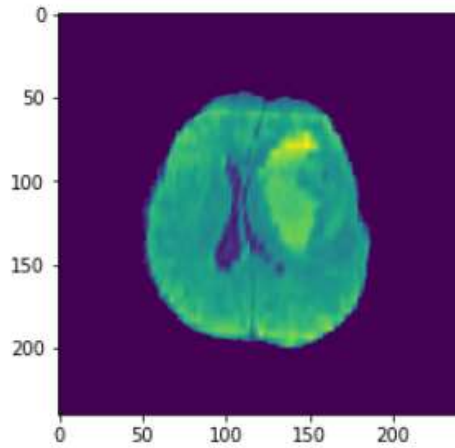
Burada dosyayı hangi plugin ile okumamız gerektiğini belirledik . Biz burada simpleitk ile okumamız gerekiyor çünkü tıbbi, emar gibi görüntüleri incelemek için özel olarak hazırlanmış bir plugindir.

Daha sonrasında elimizdeki veri setinin boyutlarını inceledik. Bizim görüntülerimiz burada 3 boyutludur. Burada en, boy ve derinlik olarak düşünmemiz gerekiyor.

Görselleştirme yapmak için de matplotlib kütüphanesini import etmemiz gerekiyor.

```
In [13]: plt.imshow(img[90,:,:])
```

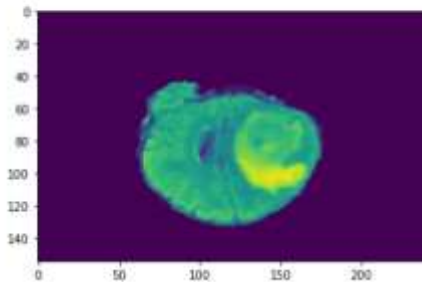
```
Out[13]: <matplotlib.image.AxesImage at 0xd6267db648>
```



Buradaki sarı olan bölge tümörlü alandır “Yukarıdan Bakış” aslında diğer bir deyişle AXIAL(TRANSVERS) kesittir.

```
In [14]: plt.imshow(img[:,90,:]) #farklı bir kesitten emar görüntüsü; burası ARKADAN yani "CORONAL KESİT"
```

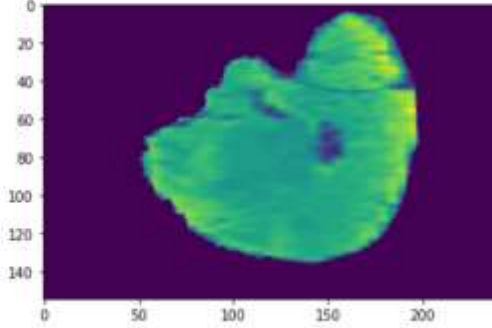
```
Out[14]: <matplotlib.image.AxesImage at 0x266ec5c978>
```



Farklı bir kesitten emar görüntüsü: Burası “arkadan” bakış yani "CORONAL KESİT" olarak adlandırılır.

```
In [15]: plt.imshow(img[:, :, 90]) #Burası da kafanın yanından bakışdır "SAGİTTAL KESİT"
```

```
Out[15]: <matplotlib.image.AxesImage at 0x266eccd518>
```



Burası da kafanın yanından bakışdır "SAGİTTAL KESİT" olarak adlandırılır.

Buradaki görüntüler emar görüntülerinin ham halleridir. Segmente edilmemiş ham halleridir. Segmente edilmiş görüntüler için şu kodlar yazılır:

```
In [16]: Örnek:
```

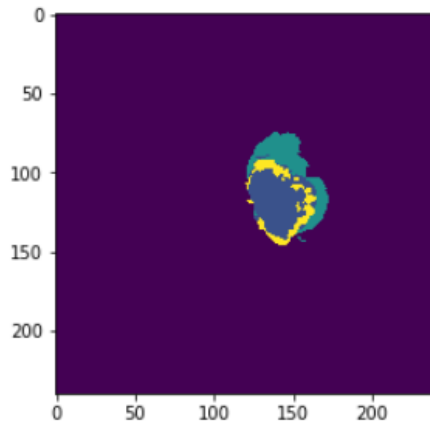
```
Out[16]: 'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_11_1\\BraTS19_2013_11_1_flair.nii.gz'
```

```
In [17]: img_seg10.imread('C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_11_1\\BraTS19_2013_11_1_seg.nii.gz', plugin='simpleitk')
```

İlk önce dosyamızın yolunu aldık . daha sonrasında flair uzantılı olan dosyadan segmente edilmiş kısımları incelemek için .flair yazan yere “seg” yazıp imread ile okuduktan sonra artık görüntüleri pyplot ile görselleştirmesini sağlamış oluruz.

```
In [18]: plt.imshow(img_seg[90, :, :])
```

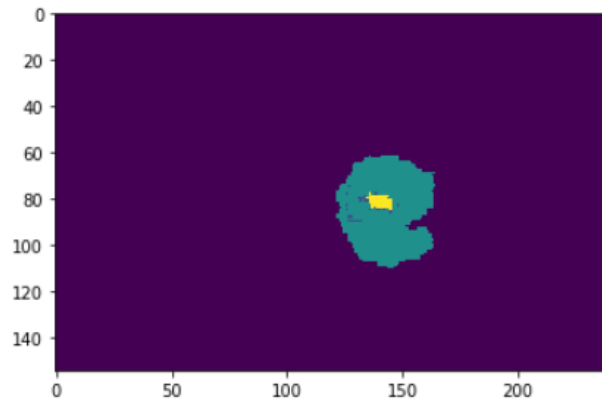
```
Out[18]: <matplotlib.image.AxesImage at 0xd62694cb48>
```



Bu tümörün segmente edilmiş halidir.

```
In [19]: plt.imshow(img_seg[:,90,:])
```

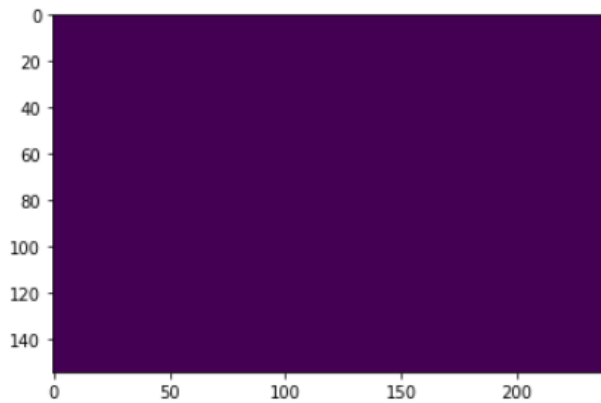
```
Out[19]: <matplotlib.image.AxesImage at 0xd6269a9c88>
```



Yukarıdaki ham olarak elde edilen görüntülerin tek tek segmente hallerini elde ediyoruz burada.

```
In [20]: plt.imshow(img_seg[:, :, 90])
```

```
Out[20]: <matplotlib.image.AxesImage at 0xd626a14488>
```



```

In [22]: plt.figure(figsize=(15,10))

plt.subplot(3,4,1) #buradaki deęirler grafik boyutunu ayarlamaya yardımcı oluyor buradaki 1'de grafięimizin sırasını belirler
plt.title('ham görüntüsü')
plt.axis('off')
plt.imshow(img[:, :, 90])

plt.subplot(3,4,2)
plt.title('segmentasyon görüntüsü')
plt.axis('off')
plt.imshow(img_seg[:, :, 90])

plt.subplot(3,4,3)
plt.title('ham görüntüsü')
plt.axis('off')
plt.imshow(img[:, 90, :])

plt.subplot(3,4,4)
plt.title('segmentasyon görüntüsü')
plt.axis('off')
plt.imshow(img_seg[:, 90, :])

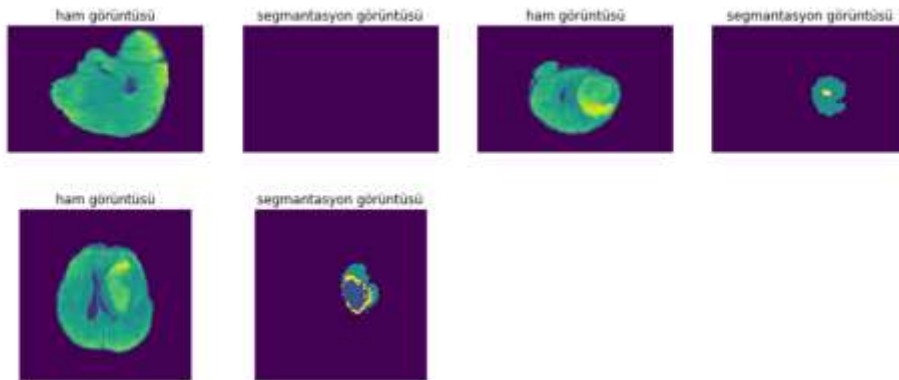
plt.subplot(3,4,5)
plt.title('ham görüntüsü')
plt.axis('off')
plt.imshow(img[90, :, :])

plt.subplot(3,4,6)
plt.title('segmentasyon görüntüsü')
plt.axis('off')
plt.imshow(img_seg[90, :, :])

```

Burada segmente olarak elde ettięim görüntüler ve ham olarak bulunan görüntülerimi görselleştirmeye çalışacağım. Bu şekilde oluşturduğum grafikler sayesinde modelimizin eğitimi ve tahmin işlemlerindeki başarı oranını daha iyi elde edebiliriz.

Out[22]: <matplotlib.image.AxesImage at 0x266ef47cc0>



Definasyon işlemi:

```
In [25]: import skimage.io as io
import matplotlib.pyplot as plt
import numpy as np
import random as r

def seg_array(path,end,label):
    files = glob.glob(path+end,recursive=True)
    img_liste = []
    r.seed(9)
    r.shuffle(files)
    for file in files:
        img = io.imread(file,plugin='simpleitk')

        if label == 1:
            img[img != 0] = 1 # tam tümör
        if label == 2:
            img[img != 1] = 0 # nekroz
        if label == 3:
            img[img == 2] = 0 # ödemli tümör
            img[img != 0] = 1
        if label == 4:
            img[img != 4] = 0 # genişleyen tümör
            img[img == 4] = 1

        img.astype('float32')

        for slice in range(60,130):
            img_s = img[slice,:,:]
            img_s = np.expand_dims(img_s,axis=0)
            img_liste.append(img_s)

    return np.array(img_liste,np.float32) #!!!!!!!
```

```
        img_s = img[slice,:,:]
        img_s = np.expand_dims(img_s,axis=0)
        img_liste.append(img_s)

    return np.array(img_liste,np.float32) #!!!!!!!

def train_array(path,end):
    files = glob.glob(path+end,recursive=True)
    img_liste = []
    r.seed(9)
    r.shuffle(files)
    for file in files:
        img = io.imread(file,plugin='simpleitk')
        img = (img-img.mean())/ img.std()
        img.astype('float32')

        for slice in range(60,130):
            img_s = img[slice,:,:]
            img_s = np.expand_dims(img_s,axis=0)
            img_liste.append(img_s)

    return np.array(img_liste,np.float32) #!!!!!!!
```

Glob modeli sayesinde emar görüntülerin bulunduğu dosyanın yolunu string bir şekilde files dosyasına atama işlemi gerçekleştirdik. Daha sonrasında da boş liste tanımladık bu boş liste en sonunda elimizdeki bütün arrayleri ekleme imkanı sağlayacak.

`r.seed(9)` bu komutla dosyaları karıştırma işlemini sağladık.

Karıştırma işlemini yaptıktan sonra dosya konumları için teker teker bütün dosyaların konumlarını bulamamızı sağlayacak.

`img = io.imread(file,plugin='simpleitk')` teker teker bütün dosyaların konumlarını saykıt image konumlandırarak emar görüntülerini elde edeceğiz.

`img = (img-img.mean())/ img.std()` #standart hale getirmemize yardımcı olur.

```

In [24]: files[0]
Out[24]: 'C:\\Users\\user1\\Desktop\\deneme1\\hggbir\\BraTS19_2013_10_1\\BraTS19_2013_10_1_flair.nii.gz'

In [25]: img.shape
Out[25]: (155, 240, 240)

In [26]: a= img[61,:,:]

In [27]: a.shape #bak mesela burada 2 boyutlu çıkar ama şimdi expand ile 3 boyutlu yaparız
Out[27]: (240, 240)

In [28]: örnek_a = np.expand_dims(a,axis=0)

In [29]: örnek_a.shape #ve bu şekilde 240,240 olan a'ya "1" olarak boyut ataması sağladık
Out[29]: (1, 240, 240)

```

U_net modeli 3 boyuttan oluşan görseller beklemektedir. Bu yüzden bizde expand ile boyut eklemesi gerçekleştirdik.

```

In [30]: yol='C:\\Users\\user1\\Desktop\\deneme1\\'

In [31]: train = train_array( yol, '**\\*flair.nii.gz')

```

Burada klasörümüzün içerisine girip verisetine ulaşmamızı sağlar.

```

In [28]: yol='C:\\Users\\user1\\Desktop\\deneme1\\'

In [29]: train = train_array( yol, '**\\*flair.nii.gz')

In [30]: train.shape
Out[30]: (7140, 1, 240, 240)

In [31]: train_re= train[0].reshape(240,240)

In [32]: train[0].shape
Out[32]: (1, 240, 240)

In [33]: train_re.shape
Out[33]: (240, 240)

```

Burada 4 boyutlu bunun mantığı model for lopta olduğu gibi bir bir alır ve tekrar üç boyutlu hale gelir.

axise 0 'a eşitleyerek bir model düşürmüş oluruz.

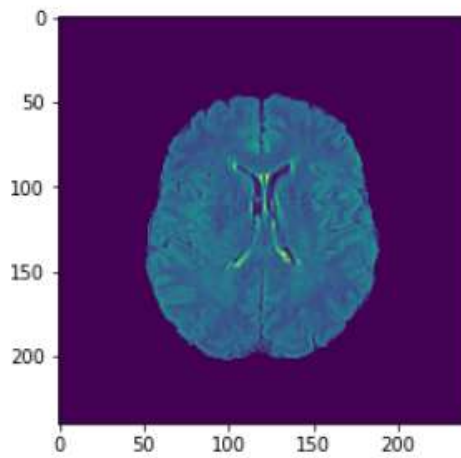
train_re= train[0].reshape(240,240) bu kodla yeniden şeklimizi boyutlandırma işlemi yapıyoruz .

train_re.shape burada bir katmanı silip sadece 240 ve 240 kalmış olur.

En sonunda pyplot ile görselleştirme işlemi gerçekleştirdim.


```
In [36]: plt.imshow(train[1000].reshape(240,240))
```

```
Out[36]: <matplotlib.image.AxesImage at 0x266f314f98>
```



Şimdi ise segmentasyon kısımlarını yapmaya çalışacağız.

Ham görüntüler 4 bölümden oluşur segmentasyon kısımlar ise de kendi içerisinde rakamlarla 3 bölümden oluşmaktadır.

1=nekroz

2=ödem

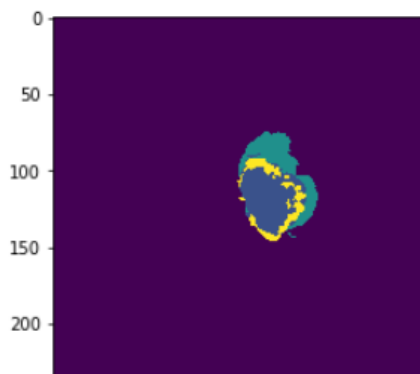
4=genişleyen tümör

```
In [38]: img_seg.shape #segmentasyon kısım
```

```
Out[38]: (155, 240, 240)
```

```
In [39]: plt.imshow(img_seg[90,:,:])
```

```
Out[39]: <matplotlib.image.AxesImage at 0x2600fe13c8>
```



İlk önce segmente olarak elde edilen görüntüye ulaşmamız gerekiyor bunun içinde img_seg ile segmente edilmiş görüntüye ulaşmış oluruz. Daha sonrasında bu segmente olarak elde edilmiş görüntüyü görselleştirmek için pyplot ile görselleştirme işlemini yaptık.

Çıkan çıktıdaki sonuçları şu şekilde okuyabiliriz;

turkuaz alan =ödem

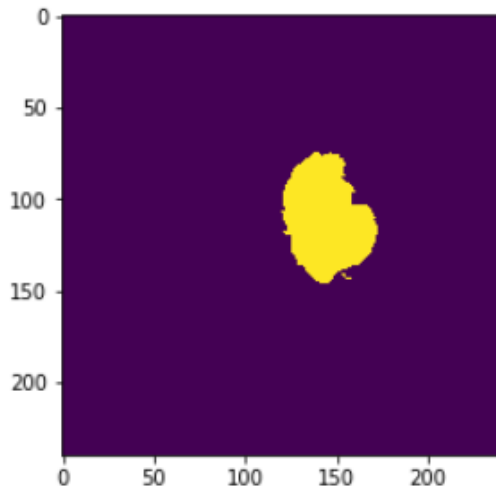
sarı renk=genişleyen tümör

merkezdeki alan ise nekroz kısım

```
In [37]: seg_tam = img_seg.copy()
```

```
In [38]: seg_tam[seg_tam != 0] = 1  
plt.imshow(seg_tam[90,:,:])
```

```
Out[38]: <matplotlib.image.AxesImage at 0xd6000a7dc8>
```



segmentasyonun tam kısmını kopyalama işlemi yaparak yapmamız gerekiyor

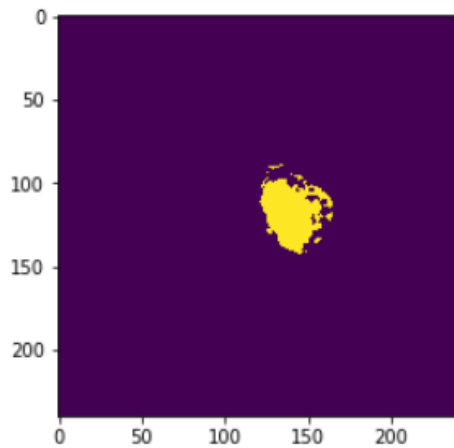
0'a eşit olmayan her alanı 1 yap diyoruz kodda

En son da tümörlü alanı gösteriyoruz.

```
In [39]: seg_nekroz = img_seg.copy()
```

```
In [40]: seg_nekroz[seg_nekroz != 1] = 0  
plt.imshow(seg_nekroz[90,:,:])
```

```
Out[40]: <matplotlib.image.AxesImage at 0xd602338548>
```

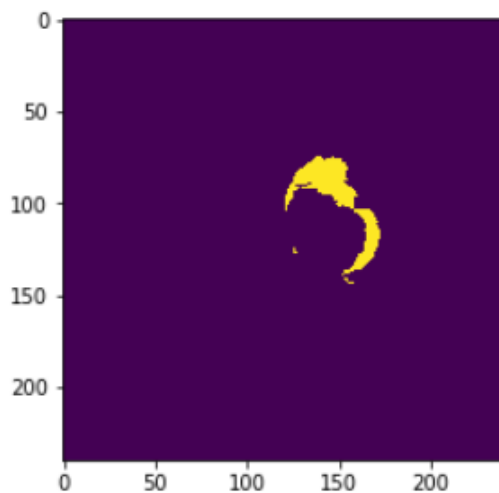


ödemle kıyasla çok küçük kalır ve aslında biraz dezavantaj çünkü modelin öğrenmesini zorlaştırıyor.

```
In [41]: seg_ödem = img_seg.copy()
```

```
In [42]: seg_ödem[seg_ödem == 1] = 0  
seg_ödem[seg_ödem == 4] = 0  
seg_ödem[seg_ödem != 0] = 1  
plt.imshow(seg_ödem[90,:,:])
```

```
Out[42]: <matplotlib.image.AxesImage at 0xd602393f48>
```



sadece ödemli kısmı alabilmek için 1 ve 4 numaralı alanı iptal etmemizi gerekiyor

ve ödemli kısmı ayırtmış oluyoruz bu şekilde.

```
In [43]: seg_tam =seg_array('C:\\Users\\user1\\Desktop\\deneme1\\', '**\\*seg.nii.gz',1)
seg_nekroz =seg_array('C:\\Users\\user1\\Desktop\\deneme1\\', '**\\*seg.nii.gz',2)
seg_ödemsiz =seg_array('C:\\Users\\user1\\Desktop\\deneme1\\', '**\\*seg.nii.gz',3)
seg_geniş =seg_array('C:\\Users\\user1\\Desktop\\deneme1\\', '**\\*seg.nii.gz',4)
seg_orj =seg_array('C:\\Users\\user1\\Desktop\\deneme1\\', '**\\*seg.nii.gz',0)
```

Burada segmentasyon işlemi yaparken elde ettiğimiz segmentasyon görüntülerin yollarını almamızı gerekiyor çünkü modeli eğitime sokarken ihtiyacımız olacak bu görsellere.

```
In [44]: x= 250 #biz burada rastgele bir sayı verdik burada başka bir değer de verebiliriz

plt.figure(figsize=(15,10))

plt.subplot(3,4,1) #tam tümör
plt.imshow(seg_tam[x].reshape(240,240))

plt.subplot(3,4,2) #nekroz
plt.imshow(seg_nekroz[x].reshape(240,240))

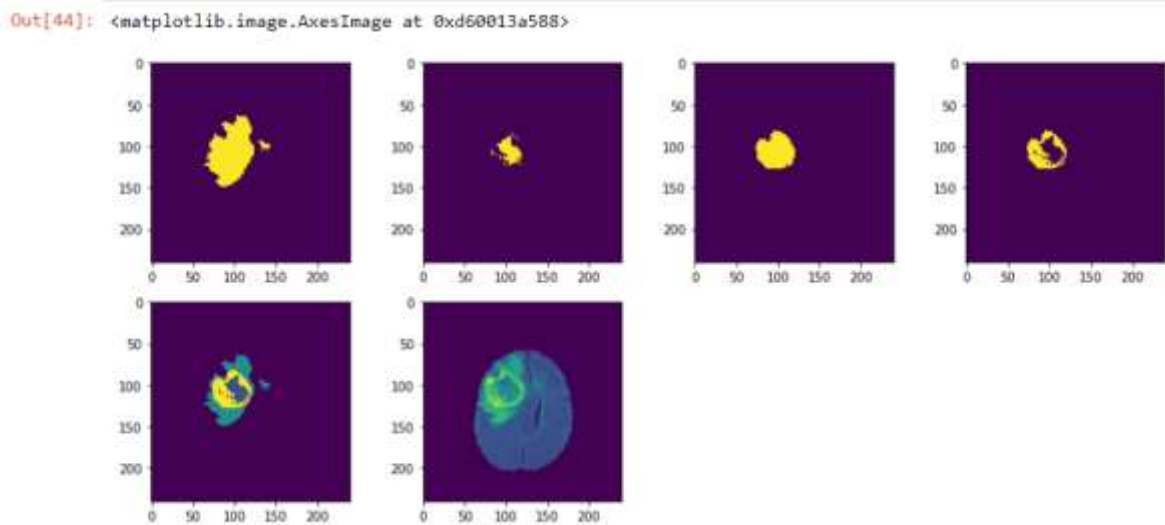
plt.subplot(3,4,3) #nekrozla genişleyen tümörün hali
plt.imshow(seg_ödemsiz[x].reshape(240,240))

plt.subplot(3,4,4) #genişleyen tümör
plt.imshow(seg_geniş[x].reshape(240,240))

plt.subplot(3,4,5) #
plt.imshow(seg_orj[x].reshape(240,240))

plt.subplot(3,4,6)
plt.imshow(train[x].reshape(240,240))
```

segmentasyon görüntülerinden elde ettiğim verileri bir görsel üzerinde görebilmemiz için bu kodları yazdım.



Bunlar da 3 bölümden oluşan segmentasyon görüntülerimin analizi.

```
In [45]: seg_tam.shape
```

```
Out[45]: (7140, 1, 240, 240)
```

```
In [46]: train.shape
```

```
Out[46]: (7140, 1, 240, 240)
```

```
In [47]: flair=train_array(yol,'**/*flair.nii.gz')
t2=train_array(yol,'**/*t2.nii.gz')
seg=seg_array(yol,'**/*seg.nii.gz',1)
```

```
In [48]: flair.shape, seg.shape, t2.shape
```

```
Out[48]: ((7140, 1, 240, 240), (7140, 1, 240, 240), (7140, 1, 240, 240))
```

```
In [49]: flair.dtype, seg.dtype, t2.dtype
```

```
Out[49]: (dtype('float32'), dtype('float32'), dtype('float32'))
```

Bu kodlarda da verilerimin boyutlarını inceledim. Burada hepsinin aynı olması gerekiyor çünkü aynı segmentasyona sahip bir görsel üzerinden inceledim .

```
Out[57]: (dtype('float32'), (7140, 2, 240, 240))
```

```
In [1]: pip install --user --upgrade tensorflow-gpu
```

```
Requirement already satisfied: tensorflow-gpu in c:\users\user1\appdata\roaming\python\python37\site-packages (1.14.0)
Collecting tensorflow-gpu
  Using cached tensorflow_gpu-2.3.1-cp37-cp37m-win_and64.whl (344.2 MB)
Requirement already satisfied: wrapt>=1.11.1 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (1.11.2)
Requirement already satisfied: gast>=0.3.3 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (0.3.3)
Requirement already satisfied: tensorboard<3,>=2.3.0 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (2.4.0)
Requirement already satisfied: absl-py>=0.7.0 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (0.11.0)
Requirement already satisfied: wheel>=0.26 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (0.33.6)
Requirement already satisfied: tensorflow-gpu-estimator<2.4.0,>=2.3.0 in c:\users\user1\appdata\roaming\python\python37\site-packages (from tensorflow-gpu) (2.3.0)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (3.1.0)
Requirement already satisfied: keras-preprocessing<1.2,>=1.1.1 in c:\users\user1\appdata\roaming\python\python37\site-packages (from tensorflow-gpu) (1.1.2)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (1.1.0)
Requirement already satisfied: grpcio>=1.8.6 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (1.27.2)
Requirement already satisfied: numpy<1.19.0,>=1.16.0 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (1.16.5)
Requirement already satisfied: astunparse==1.6.3 in c:\users\user1\appdata\roaming\python\python37\site-packages (from tensorflow-gpu) (1.6.3)
Requirement already satisfied: h5py<2.11.0,>=2.10.0 in c:\users\user1\appdata\roaming\python\python37\site-packages (from tensorflow-gpu) (2.10.0)
Requirement already satisfied: six>=1.12.0 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (1.12.0)
Requirement already satisfied: google-pasta>=0.1.8 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (0.2.0)
Requirement already satisfied: protobuf>=3.9.2 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from tensorflow-gpu) (3.13.0)
```

```
In [64]: pip install keras

Requirement already satisfied: keras in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (2.4.3)
Requirement already satisfied: numpy>=1.9.1 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from keras) (1.16.5)
Requirement already satisfied: scipy>=0.14 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from keras) (1.4.1)
Requirement already satisfied: h5py in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from keras) (2.9.0)
Requirement already satisfied: pyyaml in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from keras) (5.1.2)
Requirement already satisfied: six in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from h5py->keras) (1.12.0)
Requirement already satisfied: numpy>=1.9.1 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from keras) (1.16.5)
Requirement already satisfied: numpy>=1.9.1 in c:\users\user1\anaconda3\envs\tensor_gpu\lib\site-packages (from keras) (1.16.5)
Note: you may need to restart the kernel to use updated packages.
```

Şimdi sırada makine eğitime kısmına geçiyorum. Makinayı eğitmeden önce tensorflow ve keras kütüphanelerini import etmem gerekiyor.

```
[ ] from keras.models import Model
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import concatenate, Conv2D, MaxPooling2D, Conv2DTranspose
from keras.layers import Input, merge, UpSampling2D, BatchNormalization
from keras.callbacks import ModelCheckpoint
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras import backend as K
import tensorflow as tf

K.set_image_data_format('channels_first')

def dice_coef(y_true, y_pred):
    smooth = 0.005
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)

def dice_coef_loss(y_true, y_pred):
    return 1-dice_coef(y_true, y_pred)

def unet_model():

    inputs = Input((2, 240, 240))

    conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
```

```
[ ] conv1 = Conv2D(64, (3, 3), activation='relu', padding='same') (inputs)
batch1 = BatchNormalization(axis=1)(conv1)
conv1 = Conv2D(64, (3, 3), activation='relu', padding='same') (batch1)
batch1 = BatchNormalization(axis=1)(conv1)
pool1 = MaxPooling2D((2, 2)) (batch1)

conv2 = Conv2D(128, (3, 3), activation='relu', padding='same') (pool1)
batch2 = BatchNormalization(axis=1)(conv2)
conv2 = Conv2D(128, (3, 3), activation='relu', padding='same') (batch2)
batch2 = BatchNormalization(axis=1)(conv2)
pool2 = MaxPooling2D((2, 2)) (batch2)

conv3 = Conv2D(256, (3, 3), activation='relu', padding='same') (pool2)
batch3 = BatchNormalization(axis=1)(conv3)
conv3 = Conv2D(256, (3, 3), activation='relu', padding='same') (batch3)
batch3 = BatchNormalization(axis=1)(conv3)
pool3 = MaxPooling2D((2, 2)) (batch3)

conv4 = Conv2D(512, (3, 3), activation='relu', padding='same') (pool3)
batch4 = BatchNormalization(axis=1)(conv4)
conv4 = Conv2D(512, (3, 3), activation='relu', padding='same') (batch4)
batch4 = BatchNormalization(axis=1)(conv4)
pool4 = MaxPooling2D(pool_size=(2, 2)) (batch4)

conv5 = Conv2D(1024, (3, 3), activation='relu', padding='same') (pool4)
batch5 = BatchNormalization(axis=1)(conv5)
conv5 = Conv2D(1024, (3, 3), activation='relu', padding='same') (batch5)
batch5 = BatchNormalization(axis=1)(conv5)

up6 = Conv2DTranspose(512, (2, 2), strides=(2, 2), padding='same') (batch5)
up6 = concatenate([up6, conv4], axis=1)
```

```
[ ] up8 = concatenate([up8, conv2], axis=1)
conv8 = Conv2D(128, (3, 3), activation='relu', padding='same') (up8)
batch8 = BatchNormalization(axis=1)(conv8)
conv8 = Conv2D(128, (3, 3), activation='relu', padding='same') (batch8)
batch8 = BatchNormalization(axis=1)(conv8)

up9 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (batch8)
up9 = concatenate([up9, conv1], axis=1)
conv9 = Conv2D(64, (3, 3), activation='relu', padding='same') (up9)
batch9 = BatchNormalization(axis=1)(conv9)
conv9 = Conv2D(64, (3, 3), activation='relu', padding='same') (batch9)
batch9 = BatchNormalization(axis=1)(conv9)

conv10 = Conv2D(1, (1, 1), activation='sigmoid')(batch9)

model = Model(inputs=[inputs], outputs=[conv10])

model.compile(optimizer=Adam(lr=1e-4), loss=dice_coef_loss, metrics=[dice_coef])

return model

model = unet_model()
```

```
[ ] model.fit(x_train,seg,validation_split=0.05,batch_size=1,epochs=3,shuffle=True,verbose=1)
```

Burada flair ve t2 bölgelerinden elde edilen verileri birleştiriyoruz sebebi ise; birbirlerindeki eksikleri tamamlıyor. Mesela flair de ödemler daha iyi gözükebilirken t2' de ise merkezdeki tümörler daha net elde edilebilir. Bu şekilde de birbirlerindeki eksikleri bu şekilde

tamamlamış oluruz. Bu nedenle eğitime soktuğumuz takdirde daha iyi sonuçlar elde edebiliriz. Birleştirme işlemini de numpy ile yapacağız.

Unet_model için;

İlk önce “channels_first” ile katman sayısını belirledim. 0. İndis olarak belirledik.

Modelimde accursy’si yerine dice_coef_loss kullandık. Çünkü eğer accury kullandığımızda gerçekçi değerler vermiyordu. Modelimizi eğitime soktuğumuzda modelimizin eğitilip eğitilmediğini anlayamıyorduk.


Dice coefficient’in accuracy’dan farkı şudur:


2 tane görseli üst üste koyduğumuzda arasındaki piksel farkını bize verir. Eğer piksel farkı ne kadar az ise Dice coefficient bir o kadar artar. Bizim inputumuzda outputumuzda da girişler görsel olduğu için ikisinin arasındaki piksel farkını almamız model eğitiminde daha yararlı bilgiler vermektedir.

```
[ ] model.fit(x_train,seg,validation_split=0.05,batch_size=1,epochs=1,shuffle=True,verbose=1)

Epoch 1/3
1662/1662 [=====] - 134s 79ms/step - loss: 0.7161 - dice_coef: 0.2839 - val_loss: 0.2788 - val_dice_coef: 0.7212
Epoch 2/3
1662/1662 [=====] - 136s 82ms/step - loss: 0.4619 - dice_coef: 0.5381 - val_loss: 0.1906 - val_dice_coef: 0.8094
Epoch 3/3
1662/1662 [=====] - 137s 82ms/step - loss: 0.4025 - dice_coef: 0.5975 - val_loss: 0.2079 - val_dice_coef: 0.7921
ctensorflow.python.keras.callbacks.history at 0x7f905d135ef0>
```


Burada modelimiz için verilerimizle eğitime soktuk. Batch size ile verilerimizi parça parça bölerek öğrenmesini sağladık. Burada verilerimizin karışık olması her zaman daha iyidir bundan dolayı True yaptık.

 x_train.shape

 (1750, 2, 240, 240)

[] x_train[460].shape

(2, 240, 240)

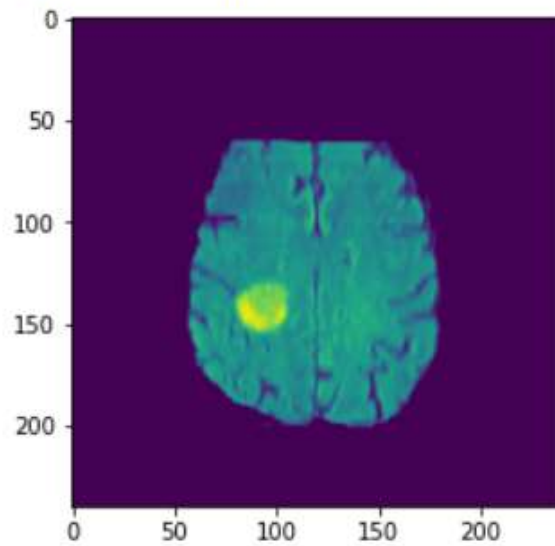
 x_train[460][0].shape

(240, 240)

Bu koddan ise x_train’ni indis şeklinde yazmamın sebebi 4 boyuttan iki boyuta düşürmek istediğim için.

```
[ ] plt.imshow(x_train[460][0])
```

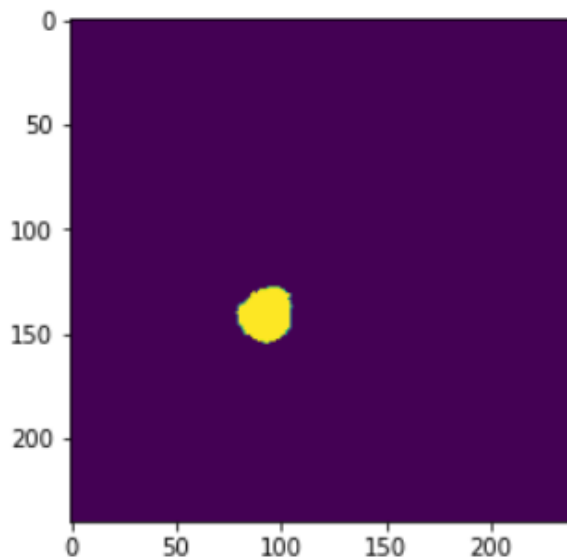
<matplotlib.image.AxesImage at 0x7f933d577358>



Burada x_train[460][0]) olan ham görüntümü görselleştirdim..

```
[ ] plt.imshow(seg[460][0])
```

<matplotlib.image.AxesImage at 0x7f933d4f9b00>

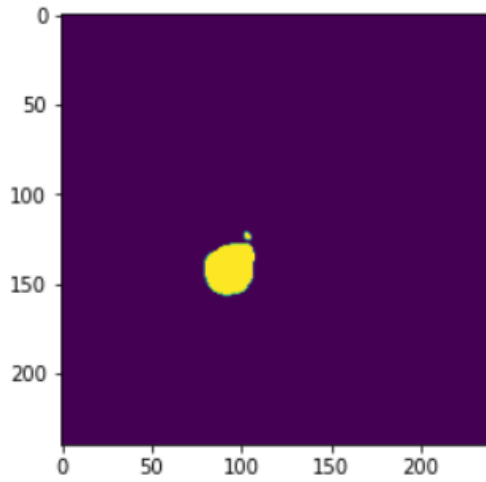


Burada da x_train[460][0]) olan segmentasyon görüntümü görselleştirdim.


```
[ ] pred=model.predict(örnek)
```

```
[ ] plt.imshow(pred[0][0])
```

<matplotlib.image.AxesImage at 0x7f933d062dd8>



Burada da bilgisayarımızın segmentasyon tahmini görselleştirdim.

Şimdi veri gözlemini yapalım bunu grafiğe dökerek yapabiliriz.

```
[ ] x      = 110
    renk = {0:'magma',
            1:'viridis',
            2:'gray',
            3:'inferno',
            4:'cividis',
            5:'hot', }

    a      = 4

örnek = np.expand_dims(x_train[x],axis=0)
pred = model.predict(örnek)

fig = plt.figure(figsize=(15,10))

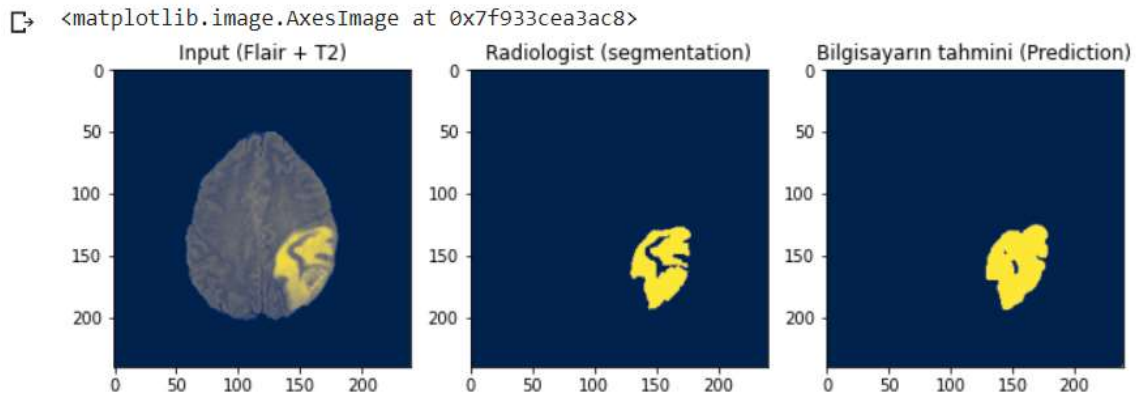
plt.subplot(141)
plt.title('Input (Flair + T2)')
plt.imshow(x_train[x][0],cmap = renk[a])

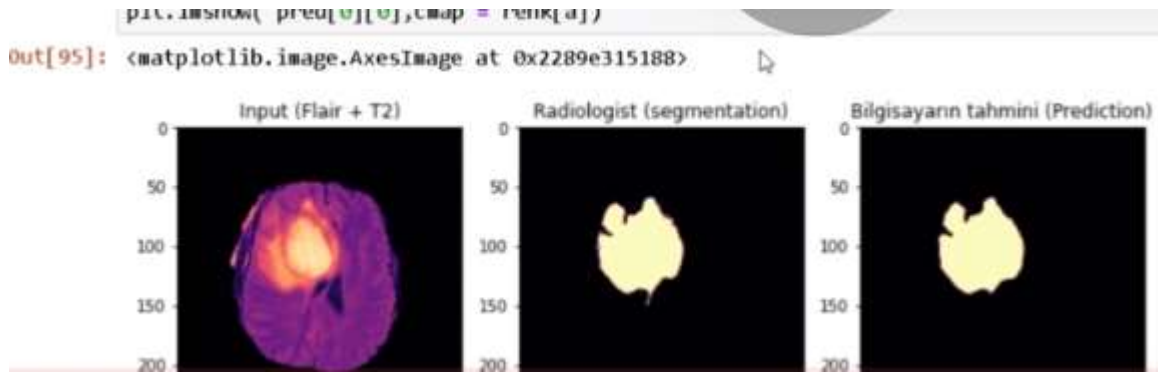
plt.subplot(142)
plt.title('Radiologist (segmentation)')
plt.imshow( seg[x][0],cmap = renk[a])

plt.subplot(143)
plt.title('Bilgisayarın tahmini (Prediction)')
plt.imshow( pred[0][0],cmap = renk[a])
```

Elimdeki verilerin indeksini x diye bir değişken tanımladım. İkinci kısımda ise renklendirme işlemi yaptım.

İkinci kısımdaki renklendirme işleminde bir liste biçiminde listelendirme işlemi yaptım. Buradaki a değişkenini hangi grafikte çalışmak isterse o rakamı yazıp istediği gibi renklendirme işlemi ile inceleyebilir.





Renkli bir şekilde incelememizin sebebi verileri daha net bir şekilde görebilmek için kullanırız.

Veri gözlemi yaparken renk haritalarının değiştirebildiğimizi gördük.

Burada bilgisayarımızın tahmini ile radiologist tümörü karşılaştırma işlemi yaptık.

```
[ ] tmp[tmp>0.2] = 1  
    tmp[tmp!=1] = 0
```

```
[ ] index_xy = np.where(tmp==1)
```

```
[ ] index_xy[0]
```

Şimdi sırada tümör kırpma işlemini yapacağız bunu da şu şekilde yapacağız.

Görüntü işleme ile ilgilendiğimiz için thresholding kavramı ile bir dizide (görüntü matrisinde) belirli bir eşik değeri altında olan kısımları 0; üstünde olan kısımları 1 yapmak suretiyle ikili (binary) bir görüntüyü elde etmemiz gerekiyor. Tmp de eğer 0.2 değerinden büyük ise 1 yap küçükse de 0 yap.

Daha sonrasında ise 1 olarak bulduğu değerleri numpy olarak listeleme işleme yaptık. Bunu yapmamızın sebebi ise ekrandaki x ve y değerlerini öğrenerek merkez konumuna erişmek.

Bu sayede tümörlü alanı adeta bir makas kesiği gibi oradan almak ve buradan aldığımız kesit ile makinaı eğitime sokmak.

```
index_xy[0]
array([[131, 131, 131, 131, 131, 131, 131, 131, 131, 131, 132, 132, 132,
       132, 132, 132, 132, 132, 132, 132, 132, 132, 132, 132, 132, 132,
       132, 133, 133, 133, 133, 133, 133, 133, 133, 133, 133, 133, 133,
       133, 133, 133, 133, 133, 133, 133, 133, 133, 134, 134, 134,
       134, 134, 134, 134, 134, 134, 134, 134, 134, 134, 134, 134, 134,
       134, 134, 134, 134, 134, 134, 134, 134, 134, 134, 135, 135,
       135, 135, 135, 135, 135, 135, 135, 135, 135, 135, 135, 135,
       135, 135, 135, 135, 135, 135, 135, 135, 135, 135, 135, 135,
       136, 136, 136, 136, 136, 136, 136, 136, 136, 136, 136, 136, 136,
       136, 136, 136, 136, 136, 136, 136, 136, 136, 136, 136, 136,
       136, 136, 136, 136, 137, 137, 137, 137, 137, 137, 137, 137, 137,
       137, 137, 137, 137, 137, 137, 137, 137, 137, 137, 137, 137,
       137, 137, 137, 137, 137, 137, 137, 137, 137, 138, 138, 138, 138,
       138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138,
       138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138, 138,
       138, 138, 138, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139,
       139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 139,
       139, 139, 139, 139, 139, 139, 139, 139, 139, 139, 140, 140, 140,
       140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140,
       140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140, 140,
       140, 140, 140, 140, 141, 141, 141, 141, 141, 141, 141, 141,
       141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141,
       141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 141,
       141, 141, 141, 141, 141, 141, 141, 141, 141, 141, 142, 142,
       142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142,
       142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142, 142,
       142, 142, 142, 142, 142, 143, 143, 143, 143, 143, 143, 143,
       143, 143, 143, 143, 143, 143, 143, 143, 143, 143, 143, 143,
       143, 143, 143, 143, 143, 143, 143, 143, 143, 143, 144, 144,
       144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
       144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144, 144,
       144, 144, 144, 144, 144, 145, 145, 145, 145, 145, 145, 145,
       145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 145,
       145, 145, 145, 145, 145, 145, 145, 145, 145, 145, 146, 146,
```

Bu index değerleri bize tümörün hangi alanda olduğunu öğrenmemize yarar.

```
[ ] merkez_y = (max(index_xy[0]) + (min(index_xy[0]))) / 2
    merkez_x = (max(index_xy[1]) + (min(index_xy[1]))) / 2
```

```
[ ] merkez_y,merkez_x

(147.0, 132.5)
```

```
[ ] img_x = np.zeros((144,144), np.float32)
```

```
[ ] img_x

array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

```
▶ x = t1ce[110,0,:,:]
x.shape
```

Biz burada ilk başta merkezdeki y ve x değerlerini öğrendik. Daha sonrasında da elimizdeki bu koordinatları kullanarak kırpma işlemi yapacağız. Bu arada unet_modeli sadece 4'ün katlarını kabul ettiği için tümörümüzün görüntüsünü y eksenine göre yani 147.0 ile yani biz bunu 4'ün katı biçiminde kullanacaksak 144 şeklinde kullanmamız gerekiyor. X ve y eksenine göre yapacak olursak da 72 72 şeklinde yapmamız gerekiyor. Bu oluşturduğumuz eksen numpy zeros kütüphanesi ile 0 şeklinde göstereceğiz.

Kod + SMS

```
def tümör_kırpma(mr, seg):  
    mr = mr[0]  
    liste = []  
    tmp = seg[0,:,:]  
    tmp[tmp>0.2] = 1  
    tmp[tmp!= 1] = 0  
    index_xy = np.where(tmp==1)  
    if index_xy[0] != []:  
        merkez_y = (max(index_xy[0]) + (min(index_xy[0])) ) / 2  
        merkez_x = (max(index_xy[1]) + (min(index_xy[1])) ) / 2  
        img_x = np.zeros((64,64), np.float32)  
        img_x[:, :] = mr[int(merkez_y - 64/2):int(merkez_y + 64/2),int(merkez_x - 64/2):int(merkez_x + 64/2) ]  
        liste.append(img_x)  
    return np.array(liste)  
  
def tumortoarray(tumor, segmentasyon):  
    liste_boş= []  
    for i in range(len(segmentasyon)):  
        img = tümör_kırpma(tumor[i], segmentasyon[i])  
        if len(img.shape) > 2:  
            liste_boş.append(img)  
            print(f'{i}')  
    return np.array(liste_boş)  
  
ödemsiz = tumortoarray(seg_ödemsiz, seg_ödemsiz)  
geniş = tumortoarray(seg_geniş, seg_ödemsiz)  
tlce_kırp = tumortoarray(tlce, seg_ödemsiz)
```

Burada tümör kırpma işlemi yapmaktayız. Biz buradaki bütün verilerimizdeki tümörlü olan verileri bulacak.

```

def tümör_kırpma(mr, seg):
    mr = mr[0]
    liste = []
    tmp = seg[0,:,:]
    tmp[tmp>0.2] = 1
    tmp[tmp!= 1] = 0
    index_xy = np.where(tmp==1)
    if index_xy[0] != []:
        merkez_y = (max(index_xy[0]) + (min(index_xy[0]))) / 2
        merkez_x = (max(index_xy[1]) + (min(index_xy[1]))) / 2
        img_x = np.zeros((64,64), np.float32)
        img_x[:, :] = mr[int(merkez_y - 64/2):int(merkez_y + 64/2),int(merkez_x - 64/2):int(merkez_x + 64/2) ]
        liste.append(img_x)
    return np.array(liste)

def tumortoarray(tumor, segmentasyon):
    liste_boş= []
    for i in range(len(segmentasyon)):
        img = tümör_kırpma(tumor[i], segmentasyon[i])
        if len(img.shape) > 2:
            liste_boş.append(img)
            print(f'{i}')
    return np.array(liste_boş)

ödemsiz = tumortoarray(seg_ödemsiz, seg_ödemsiz)
geniş = tumortoarray(seg_geniş, seg_ödemsiz)
tice_kırp = tumortoarray(tice, seg_ödemsiz)

```

Burada da verilerimde ilk önce tümörlü olan görselleri bulacağız. Bulduğum tümörlü bölgelerimi şu şekilde listeletme işlemi yaptık.



```
/usr/local/lib/p.  
# Remove the Cl
```

```
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37
```

	case_k1 p	case_k1 p
43		
44		
45		
46		
47		
48		
49		
50		
51		
52		
53		
54		
55		
56		
57		
58		
59		
60		
61		
62		
63		
64		
70		
71		
72		
73		
74		
75		
76		
77		
78		
79		

Aralardaki veri kaybının sebebi ise tümörlü olmayan bölgeleri ayırma işlemi yaptık.

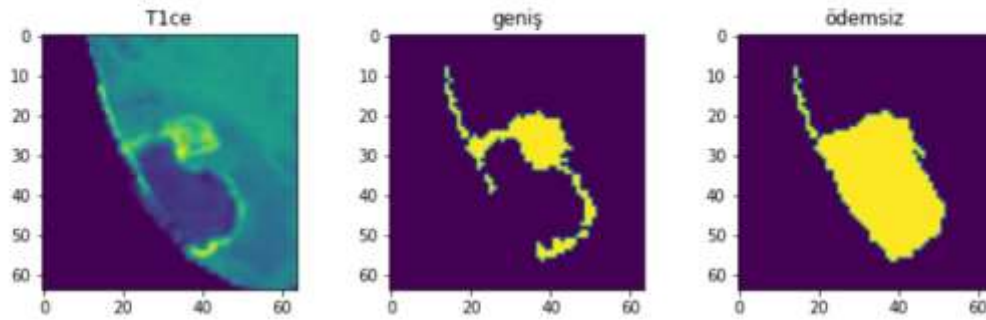

```
[ ] plt.figure(figsize=(15,10))

plt.subplot(3,4,1)
plt.title('Tıce')
plt.imshow(tıce_kırp[405,0,:,:])

plt.subplot(3,4,2)
plt.title('geniş')
plt.imshow(geniş[405,0,:,:])

plt.subplot(3,4,3)
plt.title('ödemsiz')
plt.imshow(ödemsiz[405,0,:,:])
```

<matplotlib.image.AxesImage at 0x7f3104f62ba8>



Burada direk tümörlü olan yerleri daha net bir şekilde görebilmek için görselleştirme işlemi yaptık.

```
[ ] def unet_model_7():

    inputs = Input((1, 64, 64))
    conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
    batch1 = BatchNormalization(axis=1)(conv1)
    conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(batch1)
    batch1 = BatchNormalization(axis=1)(conv1)
    pool1 = MaxPooling2D((2, 2))(batch1)

    conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool1)
    batch2 = BatchNormalization(axis=1)(conv2)
    conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(batch2)
    batch2 = BatchNormalization(axis=1)(conv2)
    pool2 = MaxPooling2D((2, 2))(batch2)

    conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(pool2)
    batch3 = BatchNormalization(axis=1)(conv3)
    conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(batch3)
    batch3 = BatchNormalization(axis=1)(conv3)
    pool3 = MaxPooling2D((2, 2))(batch3)

    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(pool3)
    batch5 = BatchNormalization(axis=1)(conv5)
    conv5 = Conv2D(512, (3, 3), activation='relu', padding='same')(batch5)
    batch5 = BatchNormalization(axis=1)(conv5)

    up7 = Conv2DTranspose(256, (2, 2), strides=(2, 2), padding='same')(batch5)
    up7 = concatenate([up7, conv3], axis=1)
    conv7 = Conv2D(256, (3, 3), activation='relu', padding='same')(up7)
    batch7 = BatchNormalization(axis=1)(conv7)
```

```
[ ]
up8 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same') (batch7)
up8 = concatenate([up8, conv2], axis=1)
conv8 = Conv2D(128, (3, 3), activation='relu', padding='same') (up8)
batch8 = BatchNormalization(axis=1)(conv8)
conv8 = Conv2D(128, (3, 3), activation='relu', padding='same') (batch8)
batch8 = BatchNormalization(axis=1)(conv8)

up9 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same') (batch8)
up9 = concatenate([up9, conv1], axis=1)
conv9 = Conv2D(64, (3, 3), activation='relu', padding='same') (up9)
batch9 = BatchNormalization(axis=1)(conv9)
conv9 = Conv2D(64, (3, 3), activation='relu', padding='same') (batch9)
batch9 = BatchNormalization(axis=1)(conv9)

conv10 = Conv2D(1, (1, 1), activation='sigmoid')(batch9)

model = Model(inputs=[inputs], outputs=[conv10])

model.compile(optimizer=Adam(lr=1e-4), loss=dice_coef_loss, metrics=[dice_coef])

return model
```

Burada kırpılan verileri modele sokarak eğitimini sağlamış ve tamamlamış olacağız.

```
[ ] model_ödemsiz = unet_model_7()

history = model_ödemsiz.fit(tice_kirp, ödemsiz,
                           validation_split= 0.20,
                           batch_size = 10,
                           epochs= 5,
                           shuffle=True,
                           verbose=1)

Epoch 1/5
93/93 [=====] - 7s 50ms/step - loss: 0.4362 - dice_coef: 0.5638 - val_loss: 0.9004 - val_dice_coef: 0.0997
Epoch 2/5
93/93 [=====] - 8s 38ms/step - loss: 0.1637 - dice_coef: 0.8163 - val_loss: 0.8167 - val_dice_coef: 0.1963
Epoch 3/5
93/93 [=====] - 3s 38ms/step - loss: 0.1396 - dice_coef: 0.8604 - val_loss: 0.4203 - val_dice_coef: 0.5731
Epoch 4/5
93/93 [=====] - 4s 38ms/step - loss: 0.1145 - dice_coef: 0.8855 - val_loss: 0.3292 - val_dice_coef: 0.6709
Epoch 5/5
93/93 [=====] - 4s 38ms/step - loss: 0.0842 - dice_coef: 0.9158 - val_loss: 0.3046 - val_dice_coef: 0.6953
```

Modelimizin ödemsiz kısmı için eğitimini tamamlamış olduk.

```
[ ] model_geniş = unet_model_7()

[ ] history = model_geniş.fit(tice_kirp, geniş,
                             validation_split= 0.20,
                             batch_size = 10,
                             epochs= 5,
                             shuffle=True,
                             verbose=1)

Epoch 1/5
93/93 [=====] - 5s 42ms/step - loss: 0.5784 - dice_coef: 0.4216 - val_loss: 0.7873 - val_dice_coef: 0.2142
Epoch 2/5
93/93 [=====] - 4s 38ms/step - loss: 0.3647 - dice_coef: 0.6353 - val_loss: 0.5835 - val_dice_coef: 0.4225
Epoch 3/5
93/93 [=====] - 4s 39ms/step - loss: 0.2980 - dice_coef: 0.7020 - val_loss: 0.4918 - val_dice_coef: 0.5140
Epoch 4/5
93/93 [=====] - 4s 39ms/step - loss: 0.2381 - dice_coef: 0.7619 - val_loss: 0.3551 - val_dice_coef: 0.6451
Epoch 5/5
93/93 [=====] - 4s 39ms/step - loss: 0.2088 - dice_coef: 0.7912 - val_loss: 0.2980 - val_dice_coef: 0.7002
```

Modelimizin genişleyen kısmı için eğitimini tamamlamış olduk.

```
[ ] x = 300

plt.figure(figsize=(15,10))

plt.subplot(3,4,1)
plt.title('T1ce')
plt.imshow(t1ce_kırp[x,0,:,:])

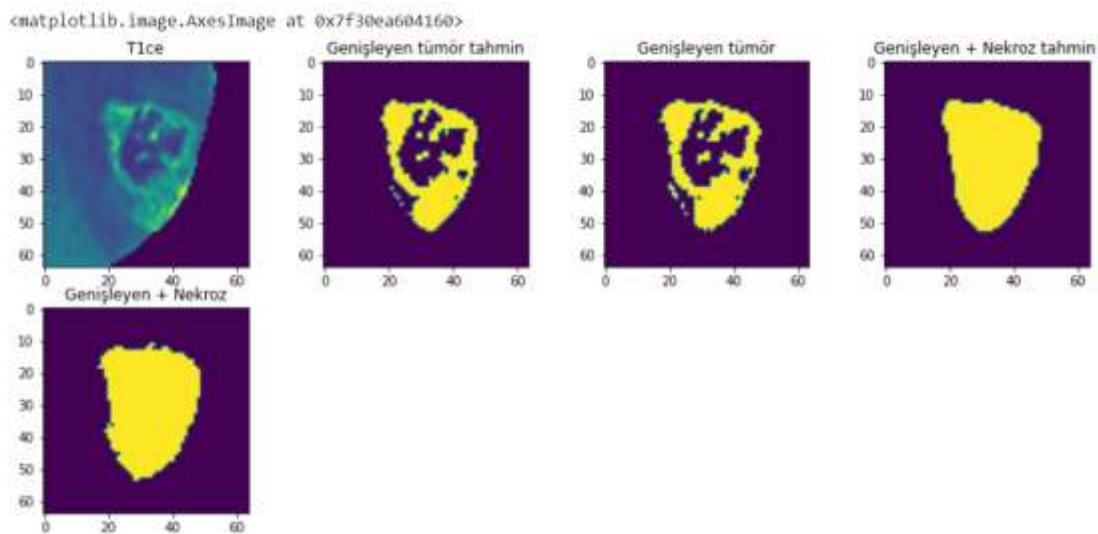
pred_geniş = model_geniş.predict(t1ce_kırp[x:x+1,:,:,:])
plt.subplot(3,4,2)
plt.title('Genişleyen tümör tahmin')
plt.imshow(pred_geniş[0,0,:,:] )

plt.subplot(3,4,3)
plt.title('Genişleyen tümör')
plt.imshow(geniş[x,0,:,:])

pred_ödemsiz = model_ödemsiz.predict(t1ce_kırp[x:x+1,:,:,:])
plt.subplot(3,4,4)
plt.title('Genişleyen + Nekroz tahmin ')
plt.imshow(pred_ödemsiz[0,0,:,:] )

plt.subplot(3,4,5)
plt.title('Genişleyen + Nekroz')
plt.imshow(ödemsiz[x,0,:,:])
```

Şimdi oluşturduğumuz bu modellerin tahmin işlemini grafikte inceleyeceğiz.



Burada radyoloğun manuel olarak yaptığı segmentasyon görüntüsü ve bilgisayarın yaptığı tahminleri görselleştirme işlemi yaptık.

```
def tümör_kırp(x, pred, size):
    crop_x = []
    list_xy = []
    p_tmp = pred[0,:,:]
    p_tmp[p_tmp>0.2] = 1
    p_tmp[p_tmp!=1] = 0
    index_xy = np.where(p_tmp==1)

    if index_xy[0].shape[0] == 0:
        return [],[]

    center_x = (max(index_xy[0]) + min(index_xy[0])) / 2
    center_y = (max(index_xy[1]) + min(index_xy[1])) / 2

    if center_x >= 176:
        center_x = center_x-8

    length = max(index_xy[0]) - min(index_xy[0])
    width = max(index_xy[1]) - min(index_xy[1])

    if width <= 64 and length <= 64: #64x64
        img_x = np.zeros((1,size,size),np.float32)
        img_x[:, :, :] = x[:,int(center_x - size/2) : int(center_x + size/2),int(center_y - size/2) : int(center_y + size/2)]
        crop_x.append(img_x)
        list_xy.append((int(center_x - size/2),int(center_y - size/2)))

    if width > 64 and length <= 64: #64x128
        img_x = np.zeros((1,size,size),np.float32)
        img_x[:, :, :] = x[:,int(center_x - size/2) : int(center_x + size/2),int(center_y - size) : int(center_y)]

    if width <= 64 and length > 64: #128x64
        img_x = np.zeros((1,size,size),np.float32)
        img_x[:, :, :] = x[:,int(center_x - size) : int(center_x),int(center_y - size/2) : int(center_y + size/2)]
        crop_x.append(img_x)
        list_xy.append((int(center_x - size),int(center_y - size/2)))

        img_x = np.zeros((1,size,size),np.float32)
        img_x[:, :, :] = x[:,int(center_x + size + 1) : int(center_x + size + 1),int(center_y - size/2) : int(center_y + size/2)]
        crop_x.append(img_x)
        list_xy.append((int(center_x),int(center_y - size/2)))

    if width > 64 and length > 64: #128x128
        img_x = np.zeros((1,size,size),np.float32)
        img_x[:, :, :] = x[:,int(center_x - size) : int(center_x),int(center_y - size) : int(center_y)]
```



```

img_x = np.zeros((1,size,size),np.float32)
img_x[:, :, :] = x[:,int(center_x - size) : int(center_x),int(center_y - size) : int(center_y)]
crop_x.append(img_x)
list_xy.append((int(center_x - size),int(center_y - size)))

img_x = np.zeros((1,size,size),np.float32)
img_x[:, :, :] = x[:,int(center_x + 1) : int(center_x + size + 1),int(center_y - size) : int(center_y)]
crop_x.append(img_x)
list_xy.append((int(center_x),int(center_y - size)))

img_x = np.zeros((1,size,size),np.float32)
img_x[:, :, :] = x[:,int(center_x - size) : int(center_x),int(center_y + 1) : int(center_y + size + 1)]
crop_x.append(img_x)
list_xy.append((int(center_x - size),int(center_y)))

img_x = np.zeros((1,size,size),np.float32)
img_x[:, :, :] = x[:,int(center_x + 1) : int(center_x + size + 1),int(center_y + 1) : int(center_y + size + 1)]
crop_x.append(img_x)
list_xy.append((int(center_x),int(center_y)))

return np.array(crop_x) , list_xy

```

Biz buradaki kırpma işlemini hem t1ce'ye göre hem de izi modelimizi eğitirken segmentasyon görüntülerimiz için de kullanacağız. Biz burada mesela emar görüntümüzü 4 parçaya bölme işlemi yaparak kırpma işlemini tamamlıyorsak segmentasyon işlemi için de otomatik olarak yapmamız gerekiyor aksi takdirde boyut hatası verir. 2 parçaya bölme işlemi yapıyor isek aynı şekilde segmentasyon işleminin de aynı şekilde parçalanması gerekiyor.

```

[ ] def tumor2array(tumor,segmentasyon):

    liste = []

    for i in range(len(tumor)):
        crop , kordinat = tümör_kırp(tumor[i,:,:,:],segmentasyon[i,:,:,:],64)

        if crop == []:
            pass
        elif crop.shape[0] ==1:
            liste.append(crop[0])

        elif crop.shape[0] ==2:

            liste.append(crop[0])
            liste.append(crop[1])

        elif crop.shape[0] ==4:

            liste.append(crop[0])
            liste.append(crop[1])
            liste.append(crop[2])
            liste.append(crop[3])

    return np.array(liste)

[ ] t1ce_array = tumor2array(t1ce,seg)
    geniş_array = tumor2array(seg_geniş,seg)
    ödemsiz_array = tumor2array(seg_ödemsiz,seg)
    t1ce_array.shape, ödemsiz_array.shape, geniş_array.shape

```

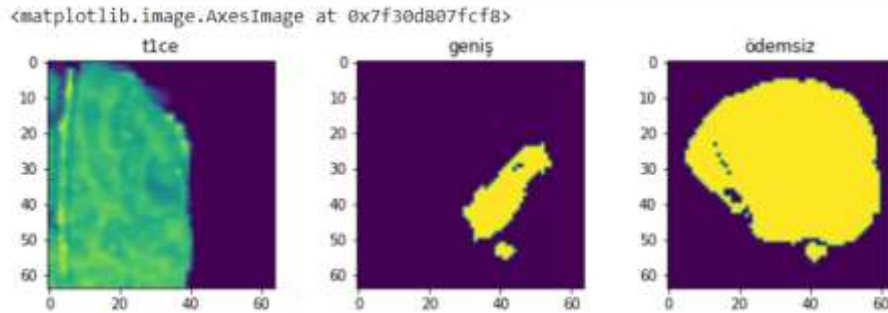
Burada tümörlerin koordinatlarına göre ayırma işlemi yapıyor.

```
[ ] plt.figure(figsize=(15,10))

plt.subplot(3,4,1)
plt.title('t1ce')
plt.imshow(t1ce_array[70,0,:,:])

plt.subplot(3,4,2)
plt.title('geniş')
plt.imshow(geniş_array[40,0,:,:])

plt.subplot(3,4,3)
plt.title('ödemsiz')
plt.imshow(ödemsiz_array[40,0,:,:])
```



Burada da t1ce bölgesindeki tümörlü alanları buluyoruz.

```
[ ] model_ödemsiz_2 = unet_model_7()

[ ] history = model_ödemsiz_2.fit(t1ce_array, ödemsiz_array,
                                validation_split= 0.20,
                                batch_size = 10,
                                epochs= 5,
                                shuffle=True,
                                verbose=1)

Epoch 1/5
238/238 [=====] - 11s 41ms/step - loss: 0.6192 - dice_coef: 0.3808 - val_loss: 0.9018 - val_dice_coef: 0.0974
Epoch 2/5
238/238 [=====] - 9s 36ms/step - loss: 0.3187 - dice_coef: 0.6813 - val_loss: 0.6412 - val_dice_coef: 0.3558
Epoch 3/5
238/238 [=====] - 9s 37ms/step - loss: 0.2442 - dice_coef: 0.7558 - val_loss: 0.5324 - val_dice_coef: 0.4637
Epoch 4/5
238/238 [=====] - 9s 37ms/step - loss: 0.1712 - dice_coef: 0.8288 - val_loss: 0.5421 - val_dice_coef: 0.4541
Epoch 5/5
238/238 [=====] - 9s 37ms/step - loss: 0.1272 - dice_coef: 0.8728 - val_loss: 0.5378 - val_dice_coef: 0.4584
```

Burada da bulduğumuz bu tümörlü parçaları eğitime sokuyoruz.

```
[ ] model_geniş_2 = unet_model_7()

[ ] history = model_geniş_2.fit(t1ce_array, geniş_array,
                                validation_split= 0.20,
                                batch_size = 10,
                                epochs= 5,
                                shuffle=True,
                                verbose=1)

Epoch 1/5
238/238 [=====] - 11s 39ms/step - loss: 0.7712 - dice_coef: 0.2288 - val_loss: 0.7675 - val_dice_coef: 0.2306
Epoch 2/5
238/238 [=====] - 9s 37ms/step - loss: 0.5782 - dice_coef: 0.4218 - val_loss: 0.5467 - val_dice_coef: 0.4496
Epoch 3/5
238/238 [=====] - 9s 37ms/step - loss: 0.4132 - dice_coef: 0.5868 - val_loss: 0.5315 - val_dice_coef: 0.4648
Epoch 4/5
238/238 [=====] - 9s 37ms/step - loss: 0.3240 - dice_coef: 0.6768 - val_loss: 0.4844 - val_dice_coef: 0.5113
Epoch 5/5
238/238 [=====] - 9s 38ms/step - loss: 0.2836 - dice_coef: 0.7164 - val_loss: 0.5023 - val_dice_coef: 0.4935
```

```
[ ] görüntü , koordinat = tümör_kırp(tıce[1020,:,:,:],seg[1020,:,:,:],64)
```

```
[ ] görüntü.shape
```

```
(4, 1, 64, 64)
```

```
[ ] pred_ödemisiz = model_ödemisiz.predict(görüntü)
    pred_geniş = model_geniş.predict(görüntü)
    pred_tam = model.predict(x_train[1020:1021,:,:,:])
```

```
[ ] pred_tam[pred_tam > 0.2] = 2
    pred_tam[pred_tam != 2 ] = 0

    pred_ödemisiz[pred_ödemisiz > 0.2] = 1
    pred_ödemisiz[pred_ödemisiz != 1 ] = 0

    pred_geniş[pred_geniş > 0.2] = 4
    pred_geniş[pred_geniş != 4 ] = 0
```

Burada şimdi oluşturduğumuz modelleri eğitime soktuktan sonra tahmin ile karşılaştırma işlemi yapacağız bu kodlarla.

```
[ ] def üstüne_ekle(pred_tam, pred_ödemisiz , pred_geniş , koordinat):

    total = np.zeros((1,240,240),np.float32)
    total[:,:,:] = pred_tam[:,:,:]

    for i in range(pred_ödemisiz.shape[0]):
        for j in range(64):
            for k in range(64):

                if pred_ödemisiz[i,0,j,k] != 0 and pred_tam[0,koordinat[i][0]+j,koordinat[i][1]+k] !=0:
                    total[0,koordinat[i][0]+j,koordinat[i][1]+k] = pred_ödemisiz[i,0,j,k]

                if pred_geniş[i,0,j,k] != 0 and pred_tam[0,koordinat[i][0]+j,koordinat[i][1]+k] !=0:
                    total[0,koordinat[i][0]+j,koordinat[i][1]+k] = pred_geniş[i,0,j,k]

    return total

[ ] deneme = üstüne_ekle(pred_tam[0,:,:,:], pred_ödemisiz, pred_geniş, koordinat)

[ ] deneme.shape

(1, 240, 240)
```

9.Kaynaklar

<https://ayyucekizrak.medium.com/g%C3%B6r%C3%BCnt%C3%BCb%C3%B6l%C3%BCtleme-segmentasyon-i%C3%A7in-derin-%C3%B6l%C4%9Frenme-u-net-3340be23096b>

<https://dergipark.org.tr/en/download/article-file/587409>

<https://ab.org.tr/ab16/bildiri/83.pdf>

https://www.emo.org.tr/ekler/430a7300689d591_ek.pdf

<https://medium.com/deep-learning-turkiye/mri-g%C3%B6r%C3%BCnt%C3%BCleri-%C3%BCzerinden-beyin-t%C3%BCm%C3%B6r%C3%BC-tespiti-ec644a2ff0c9>