Hem görüntünün mevcut halinin moment’i alınıp hemde kenarları çıkarılarak moment alınırsa. Ve her iki öz nitelik Arff Dosyasına eklenirse başarı oranının artacağını düşündüm. Çünkü daha önce iki ayrı şekilde öznitelik çıkarımı yapmıştım ve başarı oranları farklı çıkmıştı. Zaten kenarları çıkarılan görüntü yeni ve özgün görüntüden %90 oranında farklılaşmış bir görüntü oluyor.

Bundan sonra Diğer öz nitelik çıkarma yöntemlerini de kullanarak tamamını kapsayan ARFF dosyaları oluşturacağım. Bu sayede başarı oranının daha da artacağını düşünüyorum. Bunun hız yönünden büyük bir handikap oluşturacağının farkındayım.

|  |  |
| --- | --- |
| Başarı Oranı(Moment) | %14 |
| Yöntem /Açıklama | Moment Alarak Öz nitelik Çıkarma |
| Kaynak Kod | private static double[] oznitelikHesapla(Image img){  // top sizde..  double[] feature = null;  feature = Moment.*invoke*(img, 1, 1);    return feature; } |
| Ekran Görüntüsü |  |

|  |  |
| --- | --- |
| Başarı Oranı(Sobel+Moment) | %8 |
| Açıklama | Sobel İle kenar çıkarıp moment alınarak öznitelik çıkarımı yapıldı. |
| Kaynak Kod | private static double[] oznitelikHesapla(Image img) {  // top sizde..  double[] feature = null;  img = Sobel.*invoke*(img, Sobel.*GRAD*, true);  feature = Moment.*invoke*(img, 1, 1);    return feature; } |
| Ekran Görüntüsü |  |

|  |  |
| --- | --- |
| Başarı Oranı(Moment + [Sobel+Moment]) | %29 |
| Açıklama | Hem Özgün görüntünün momentini al ARFF dosyasına ekle. Hem de sobel ile kenarlarını çıkararak görüntünün momentini al ve yine ARFF dosyasına ekle. |
| Kaynak Kod | double[] featureMoment = *oznitelikHesapla*(img, "moment");//moment uygula  for (int k = 0; k < featureMoment.length; k++)  pw2.print(featureMoment[k] + ",");  double[] featureMomentWithSobel = *oznitelikHesapla*(img, "momentwithsobel");//sobel ile kenar çıkar sonra moment al  for (int k = 0; k < featureMomentWithSobel.length; k++)  pw2.print(featureMomentWithSobel[k] + ",");    if (type.equalsIgnoreCase("moment")) {  feature = Moment.*invoke*(img, 1, 1); } else if (type.equalsIgnoreCase("momentwithsobel")) {  img = Sobel.*invoke*(img, Sobel.*GRAD*, true);  feature = Moment.*invoke*(img, 1, 1); } |
| Ekran Görüntüsü |  |

|  |  |
| --- | --- |
| Başarı Oranı(Histogram) | %37 |
| Açıklama | Histogram Alarak Öznitelik Oluşturma |
| Kaynak Kod | private static double[] oznitelikHesapla(Image img){  // top sizde..  double[] feature = null;   feature = Histogram.*invoke*(img, false);   return feature; } |
| Ekran Görüntüsü |  |

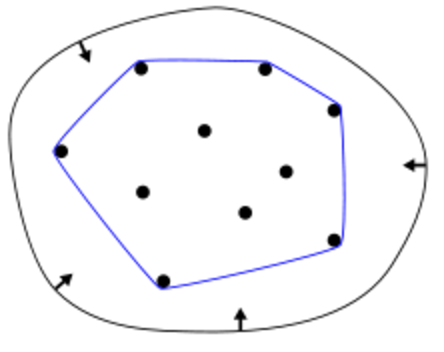
|  |  |
| --- | --- |
| Başarı Oranı(Sobel+Hstogram) | %43 |
| Açıklama | Histogram almadan önce sobel ile görüntünün kenarları çıkarıldı. Ve bu yeni görüntünün histogramı alındı. |
| Kaynak Kod | private static double[] oznitelikHesapla(Image img) {  // top sizde..  double[] feature = null;  img = Sobel.*invoke*(img, Sobel.*GRAD*, true);  feature = Histogram.*invoke*(img, false);    return feature; } |
| Ekran Görüntüsü |  |

|  |  |
| --- | --- |
| Başarı Oranı(Sobel+Moment + Histogram) | %55 |
| Açıklama | Özgün görüntünün momenti ve histogramı alındır. Bir de özgün görüntüye sobel uygulanarak moment ve histogram alındı. Bu dört öz nitelik ile ARFF dosyaları oluşturuldu. |
| Kaynak Kod | if (type.equalsIgnoreCase("moment")) {  feature = Moment.*invoke*(img, 1, 1); } else if (type.equalsIgnoreCase("momentwithsobel")) {  img = Sobel.*invoke*(img, Sobel.*GRAD*, true);  feature = Moment.*invoke*(img, 1, 1); } else if (type.equalsIgnoreCase("histogram")) {  feature = Histogram.*invoke*(img, false); } else if (type.equalsIgnoreCase("histogramwithsobel")) {  img = Sobel.*invoke*(img, Sobel.*GRAD*, true);  feature = Histogram.*invoke*(img, false); } |
| Ekran Görüntüsü |  |

Geçen yılkı görüntü işleme final ödevinde convexHull ve HOG birlikte kullanarak Yaprak tanıma ve sınıflandırma işleminde yaptığım çalışma ile %17 başarı oranı elde etmiştim. Bu yetersiz bir orandı ancak şuan bu yöntemi tek başına kullanmak yerine dier yöntemlerle birlikte kullanmak başarı oranını artırmak için mümkün görünmektedir. Bu kez ise convexHull yöntemini üstte kullandığım diğer algoritmalrla bilrikte kullanarak başarı oranını daha artıracağımı düşünüyorum.

Convexhull:

Minimum dış bükey oluncaya kadar sıkıştırma işlemi.



|  |  |
| --- | --- |
| Başarı Oranı | %11 |
| Açıklama | Convexhull özgün görüntüler üzerinde denendiğinde |
| Kaynak Kod | private static double[] oznitelikHesapla(Image img){  // top sizde..  double[] feature = null;   Image imgBinary = Threshold.*invoke*(img, 1 \* DoubleImage.*byteToDouble*);  Image borderImage = BInternGradient.*invoke*(imgBinary, FlatSE.*square*(3));  feature = *computeConvex*(borderImage);   return feature; } |
| Ekran Görüntüsü |  |

|  |  |
| --- | --- |
| Başarı Oranı(Sobel + ConvexHull) | %10 |
| Açıklama | Sobel ile kenarlar çıkarılarak ConvexHull uygunadı |
| Kaynak Kod | private static double[] oznitelikHesapla(Image img) {  // top sizde..  double[] feature = null;  img = Sobel.*invoke*(img, Sobel.*GRAD*, true);   Image imgBinary = Threshold.*invoke*(img, 1 \* DoubleImage.*byteToDouble*);  Image borderImage = BInternGradient.*invoke*(imgBinary, FlatSE.*square*(3));  feature = *computeConvex*(borderImage);    return feature; } |
| Ekran Görüntüsü |  |

|  |  |
| --- | --- |
| Başarı Oranı(Sobel+Moment + Histogram + ConvexHull) |  |
| Açıklama |  |
| Kaynak Kod |  |
| Ekran Görüntüsü |  |

|  |  |
| --- | --- |
| Başarı Oranı |  |
| Açıklama |  |
| Kaynak Kod |  |
| Ekran Görüntüsü |  |