**FİNAL PROJE(RAPOR)**

**Elde Edilen Başarı Oranı : %56 tir. Bu çalışmada aşağıdaki adımlar sırasıyla uygulanmıştır.**

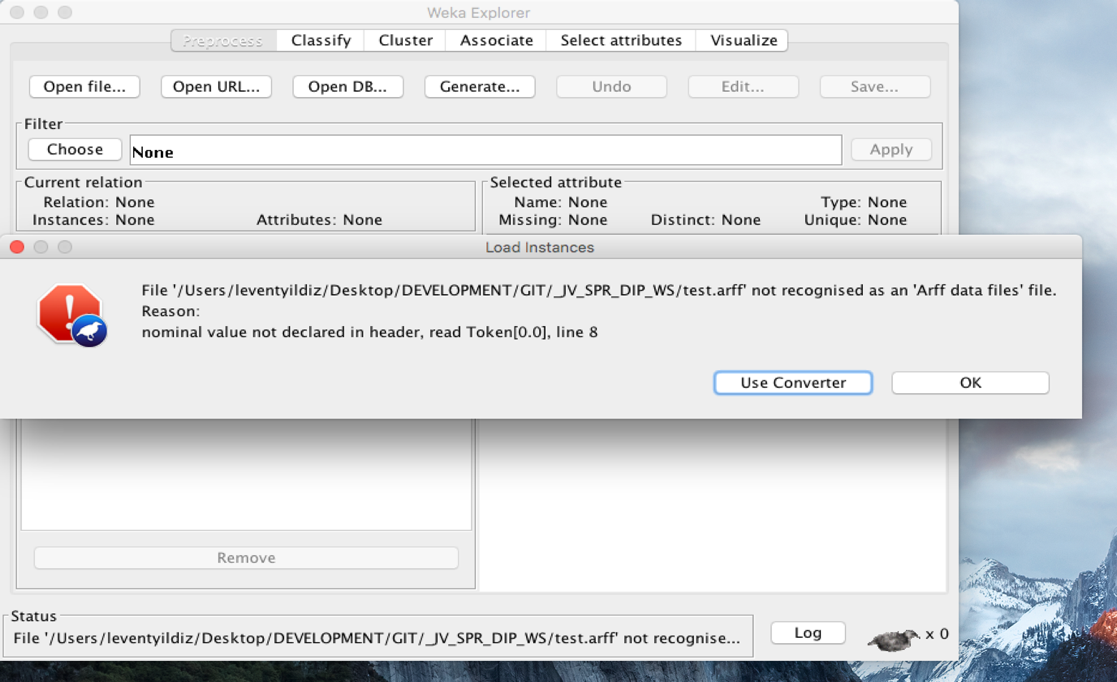
1. İlk olarak internet üzerinden görüntülerin benzerliklerinin nasıl bulunacağı araştırıldı. Hemen hemen ödev için çalışılan tüm günlerde bu çalışma kısa süreler şeklinde tekrarlandı.
2. Bu araştırma sonucunda görüntülerdeki benzerliklerin bulunması için özellik çıkarmak gerektiği kanısına varıldı.
3. Geçen sene ki final ödevinin de aynı konudan olduğu anlaşıldı. Bu noktada kendi ödev projemi inceledim.
4. Bu işlemden sonra VPT kütüphanesi içerisinde bulunan yöntemlerle öznitelik çıkarma işlemi yapıldı.
5. İlk öznitelik çıkarma olarak histogram seçildi.

double[] feature = vpt.algorithms.histogram.Histogram.*invoke*(img,false);

oluşan ARFF dosyaları weka ile incelendi.

1. Wekada ARff dosyası analiz edilirken hata alındı. Bu hatanın giderilmesi için internet üzerinden araştırmalar yapıldı.

**“Reason: nominal value not declared in header, read Token”**



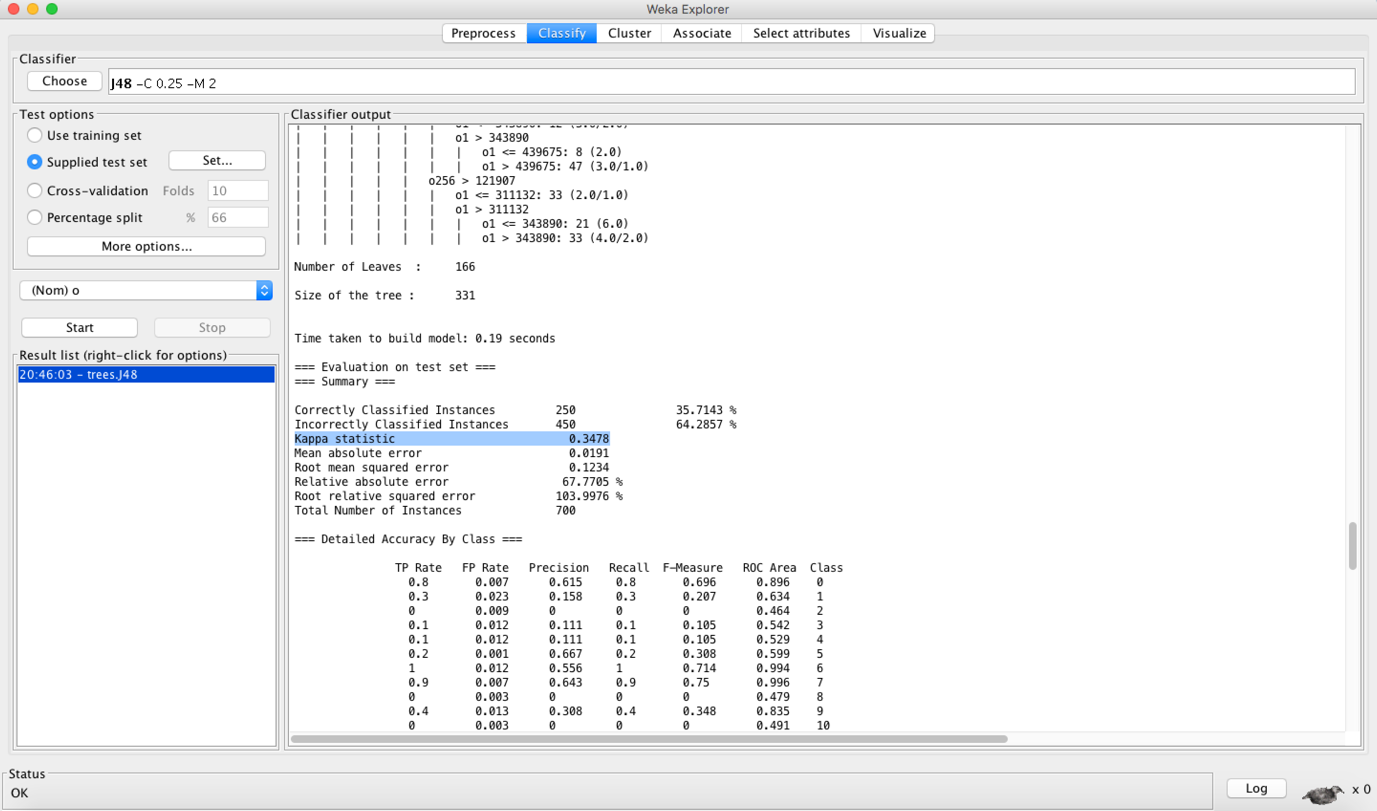
1. ARFF dosyalarında deklare edilmiş öznitelik sayısıyla öznitelik(değer) sayılarının eşleşmediği görüldü. Öz nitelik sayısı güncellendi.

// *TODO oznitelik yoneyi uzunlugunu güncelleyin*//int featureVectorLength = 4;  
int featureVectorLength = 256;

1. Histogramlar ile çıkan öznitelikler IB1 sınıflandırılıp doğrulama yapıldı. Başarı oranın %40 larda kappa statistic değerinin de yine yaklaşık olarak 0,4 lerde olduğu görüldü.

**NOT:** Weka/IB1 en yakın komşu algoritmasını kullandığı internetten araştırıldı. Oluşturduğumuz öz niteliklerin tamamı sayısal değerlerden oluştuğu için bu yöntemle sınıflara ayırmak ve daha sonra teste tabi tutmak yerinde olacaktır.

1. Elimizdeki veriler sayısal olduğu için c4.5 algoritmasıyla training veri setini kullanarak karar ağacı oluşturup , bu karar ağacında test verilerini kullanarak ne kadar doğru tespit yapabildiğimizi denedim. Ancak bu veri setinde başarı oranı çok düşük çıktı.

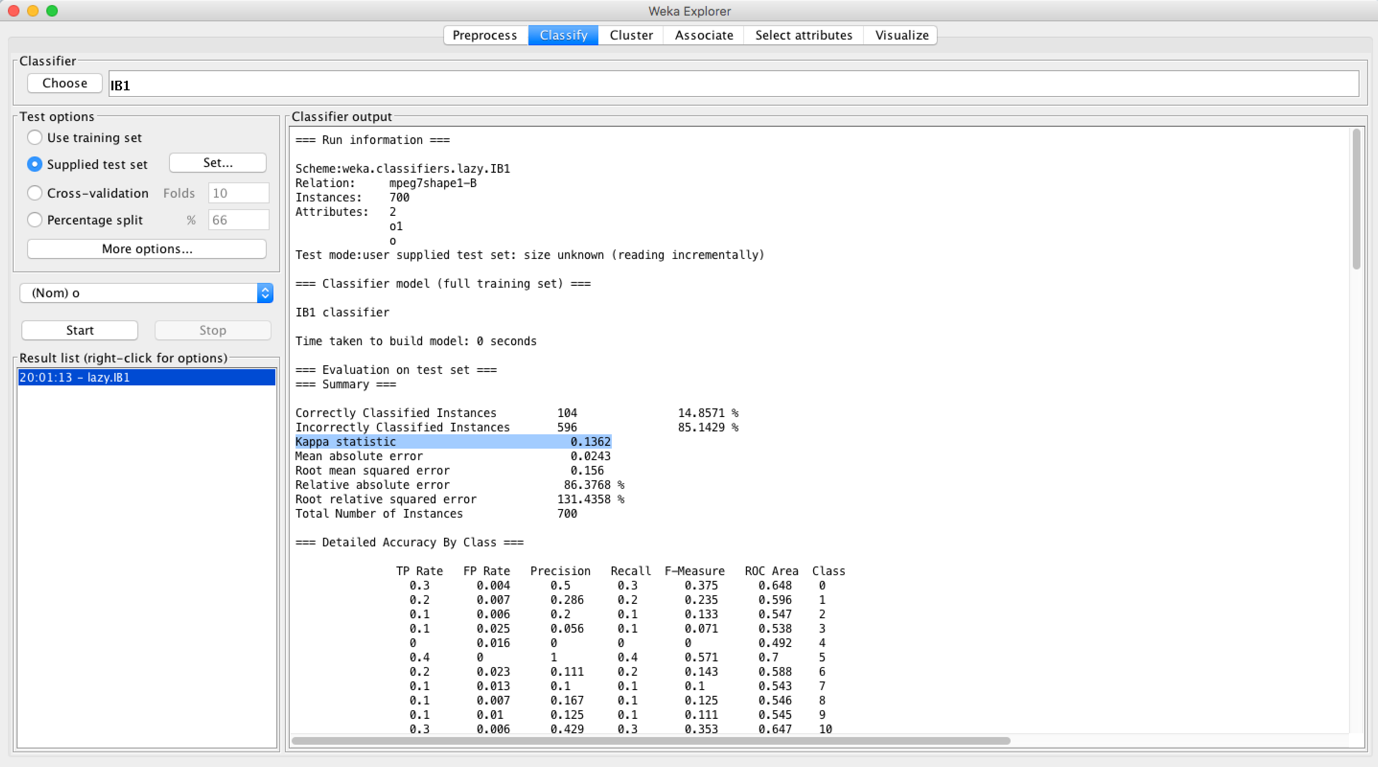


1. Öznitelik Vektör Genişliğini 1 olarak set edip Invariant moment ile öznitelik hesabı yapıldı.

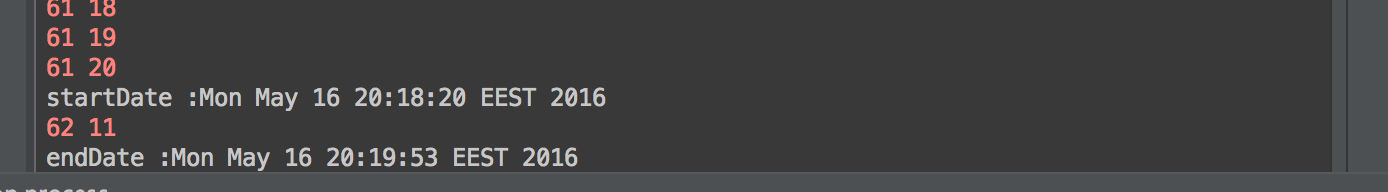
int featureVectorLength = 1;

double[] feature = InvariantMoment.*invoke*(img, Integer.*valueOf*(1));

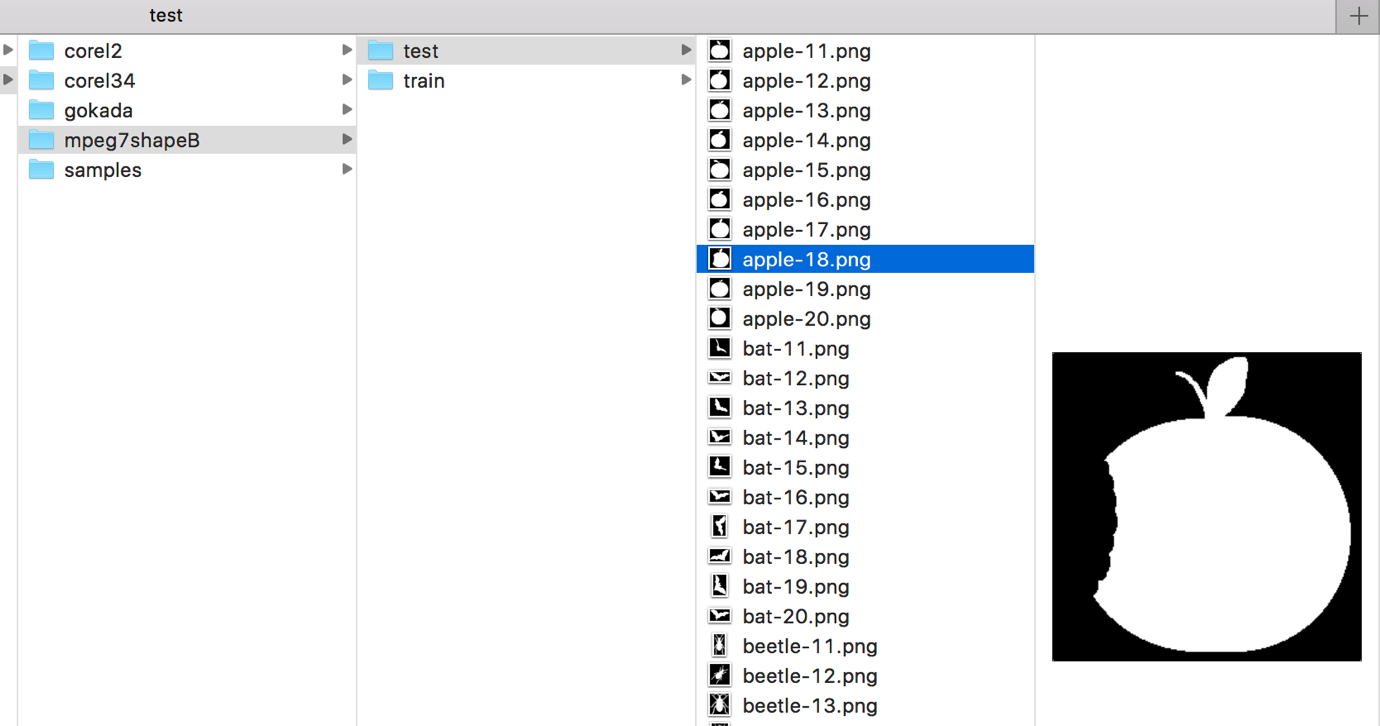
Sonuç:



Invariant moment : <https://en.wikipedia.org/wiki/Image_moment>



1. VPT algoritmasının kaynak kodlarıı incelendi. Fikir üretmesi ve yeni sonuçlar bulmak adına kaynak kodlardan işimize yarayan/yaramayan kodlar incelendi.
2. Resimler incelendi. Elimizdeki resimlerin siyah beyaz(grayscale) resimlerden oluştuğu ve bu uygun bir çözüm üretmek gerektiği görüldü.

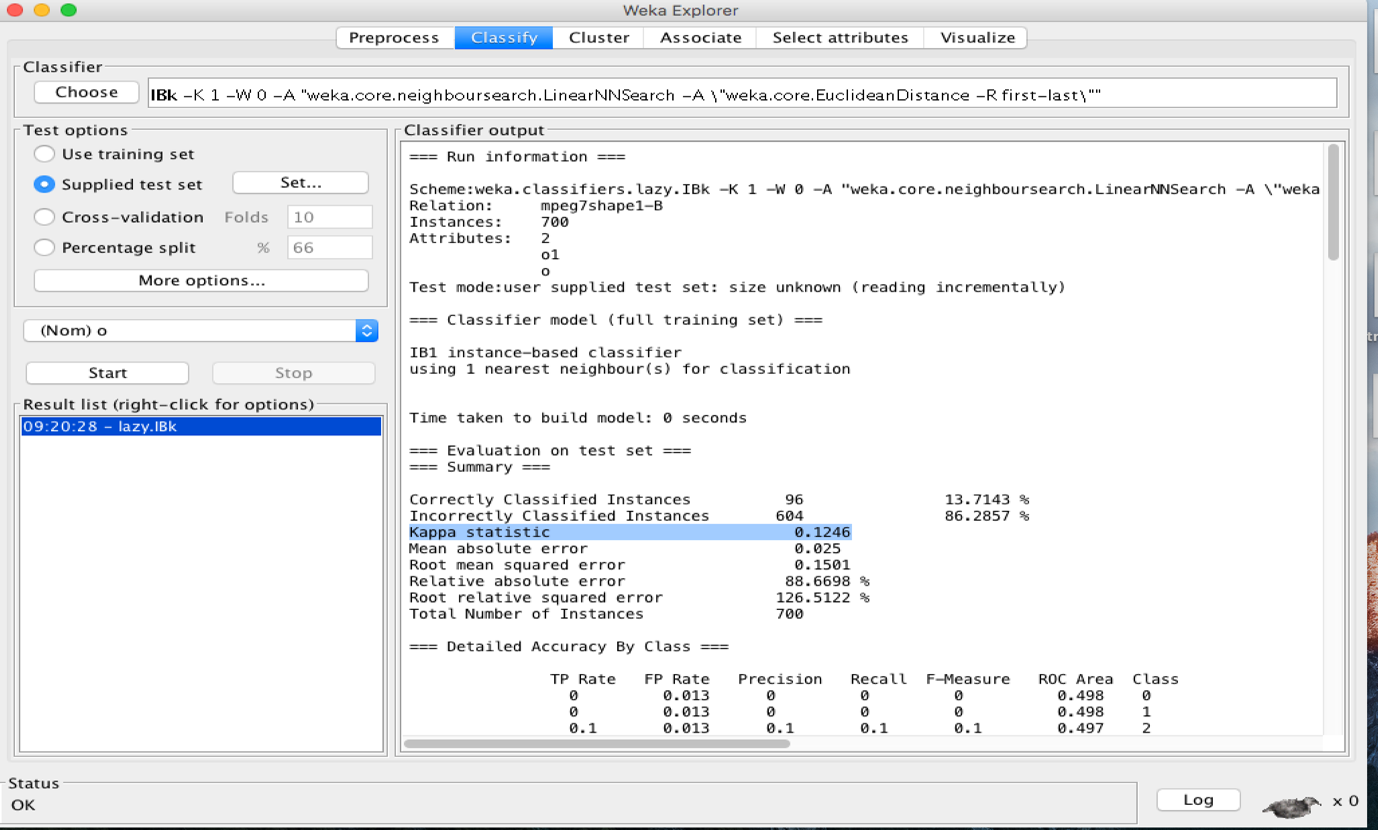


1. Öznitelik çıkarma yöntemleri ile ilgili tekrar internet üzerinde araştırma yapıldı.

Moment alarak hesaplama yapılmasının işimize yarayacağı düşünülerek bu yöntemler öznitelik çıkarımı yapıldı.

double[] feature = null;  
  
 feature = Moment.*invoke*(img,1,1);

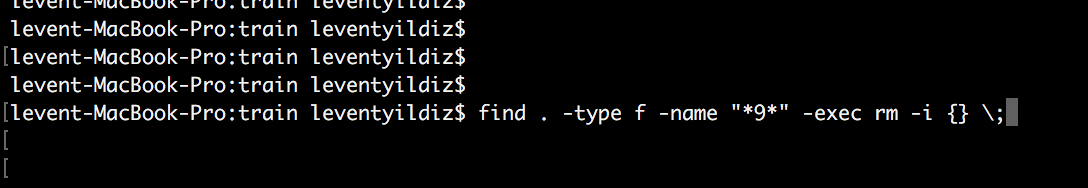
@RELATION mpeg7shape1-B  
@ATTRIBUTE o1 REAL  
@ATTRIBUTE o {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69}  
@DATA  
6.11150445E8,0  
5.69306756E8,0  
6.70233357E8,0  
4.58914266E8,0  
7.02001434E8,0



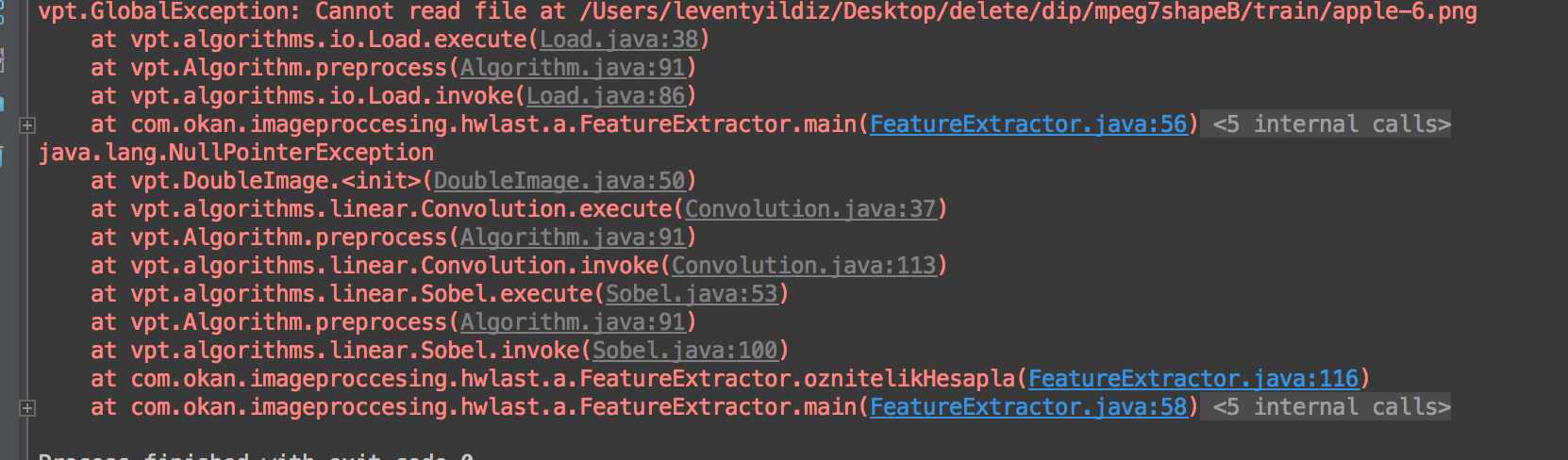
1. Sınıflandırma algoritmaları araştırıldı.

Sınıflandırma algoritmaları:

* ► [Artificial neural networks](https://en.wikipedia.org/wiki/Category:Artificial_neural_networks)‎
* ► [Decision trees](https://en.wikipedia.org/wiki/Category:Decision_trees)‎
* ► [Ensemble learning](https://en.wikipedia.org/wiki/Category:Ensemble_learning)‎

1. Tüm resmin değil sadece kenarları baz alarak bir öznitelik çıkarma yöntemi denemenin faydalı olacağını düşünerek kenar çıkararak bir çalışma yapılması gerektiği kanısına varıldı.
2. Ayrı öznitelik çıkarma yöntemlerinin tek tek denendiğinde her birinde ortalama %15 oranında tespit yapılabildiği görüldü. Bunun için bu yöntemlerin hepsini aynı anda kullanılarak daha da başarılı bir sonuç elde edileceği kanısına varıldı.
3. Tüm öznitelik yöntemleri aynı anda kullanılacağı için bu işlem uzun sürecektir. Bundan sonra test ve training verilerinin bir kısmını silerek deney sonuçları hızlandırıldı.

Görüntülerden bir kısmının silinmesi sonucu oluşan hatalar giderildi.



Test veri setindeki resimlerin bir kısmı silindiği için kodlar kaynaklara göre güncellendi.

for(int j = 11; j <= 20; j++){  
 Image img = Load.*invoke*(pathTest + classNames[i] + "-" + j + ".png");

11 den 20. Resime kadar olan döngü 11 den 15’e kadar’a düşürüldü.

for(int j = 11; j <= 15; j++){  
 Image img = Load.*invoke*(pathTest + classNames[i] + "-" + j + ".png");

Aynı şekilde. Train veri setindeki resimlerin de bir kısmı silindi kod yine kaynaklara göre güncellendi.

for(int j = 1; j <= 10; j++){  
 Image img = Load.*invoke*(pathTrain + classNames[i] + "-" + j + ".png");

1 den 10’a kadar Resim yerine 1 den 5 e kadar döngünün dönmesi sağlandı.

for(int j = 1; j <= 5; j++){  
 Image img = Load.*invoke*(pathTrain + classNames[i] + "-" + j + ".png");

1. 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.
2. 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 |  |

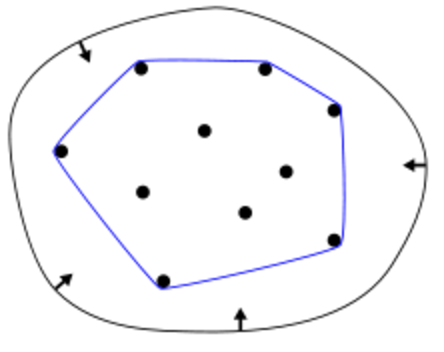
|  |  |
| --- | --- |
| Başarı Oranı(Sobel+Hstogram) | %43 |
| Açıklama | Histogram almadan önce sobel ile görüntünün kenarları çıkarıldı. Ve bu yeni görüntünün histogramı alındı. |
| Kaynak Kod | private static double[] oznitelikHesapla(Image img) {  // top sizde..  double[] feature = null;  img = Sobel.*invoke*(img,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

1. 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) | %55 |
| Açıklama | Convex Hull ve Hog diğer yöntemlerle birlikte denenince başarı oranı yine %55 olarak tespit edilid. Bir değişiklik yok |
| Kaynak Kod | } else if (type.equalsIgnoreCase("convexHog")) {   Image imgBinary = Threshold.*invoke*(img, 1 \* DoubleImage.*byteToDouble*);  Image borderImage = BInternGradient.*invoke*(imgBinary, FlatSE.*square*(3));  feature = *computeConvex*(borderImage);  } else if (type.equalsIgnoreCase("convexHogwithsobel")) {  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);  } |
| Ekran Görüntüsü |  |

|  |  |
| --- | --- |
| Başarı Oranı (BIC ) | %26 |
| Açıklama | Görüntüyü renklendirerek BIC çözümünü uygulama |
| Kaynak Kod | private static double[] oznitelikHesapla(Image img) {  // top sizde..  double[] feature = null;  feature = BIC.*invoke*(Gray2Color.*invoke*(img));  return feature; } |
| Ekran Görüntüsü |  |

|  |  |
| --- | --- |
| Başarı Oranı(Sobel + BIC ) | %19 |
| Açıklama | Görüntünün kenarları çıkartılıp renklendirerek BIC çözümünü uygulama |
| Kaynak Kod | private static double[] oznitelikHesapla(Image img) {  // top sizde..  double[] feature = null;  img = Sobel.*invoke*(img, Sobel.*GRAD*, true);   feature = BIC.*invoke*(Gray2Color.*invoke*(img));   return feature; } |
| Ekran Görüntüsü |  |

|  |  |
| --- | --- |
| Başarı Oranı | %56 |
| Açıklama | BIC + Sobel + Moment + ConvexHull + Histogram birlikte kullanılarak oluşturulan ARFF kayıtları ile başarı oranı %56 olarak tespiet edilmiştir. |
| Kaynak Kod | } else if (type.equalsIgnoreCase("bic")) {  feature = BIC.*invoke*(Gray2Color.*invoke*(img)); } else if (type.equalsIgnoreCase("bicwithsobel")) {  img = Sobel.*invoke*(img, Sobel.*GRAD*, true);  feature = BIC.*invoke*(Gray2Color.*invoke*(img)); |
| Ekran Görüntüsü |  |