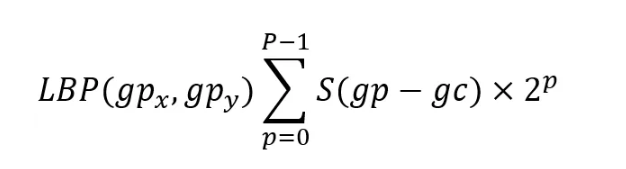
**Hazırlayan**

**Esra ŞAVIRDİ**

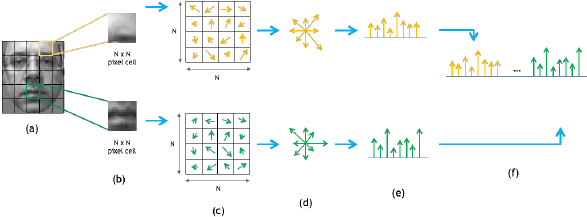
**190303076**

**Görüntü Üzerinden Öznitelik Çıkaran Programı**

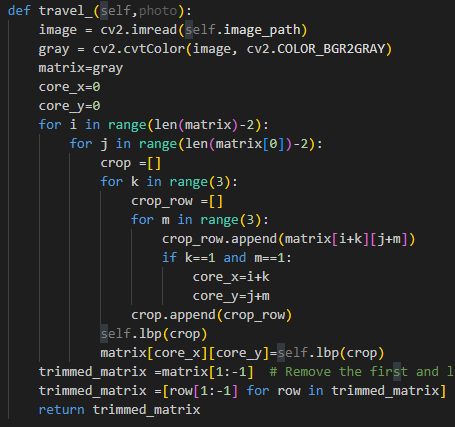
* Proje Hakkında
  + Görğüntü işlemeye giriş dersinde anlatılmış olan öznitelik çıkarma algoritmalarından Hog, LBP ile bir görüntü öznitelik çıkarmaya amaçlamaktadır. Bu proje kapsamında kullanıcı istediği Görüntüyü seçip yada canlı bir şekilde görüntüyü çekip ve bu görüntüden öznitelikleri çıkarmak için hog yada lbp yöntemleriyle gerçekleştirmek ve histogramı görüntelemek için basit bir arayüzü tasalrladık.
  + LBP((Local Binary patterns), Bu yöntem, yüz tanıma, yüz ifadesi tanıma ve nesne algılama sistemlerinde gibi çok uygulamalarda kullanılmıştır.
    - Derste işlenmiş yaklaşıma göre görüntüdeki her 3x3 pencerenin bir lbp kodunu çıkarmak için işlenmesini gerektirir. aşağıdaki göstermiş olduğumuz lbp matematiksel formuludur.

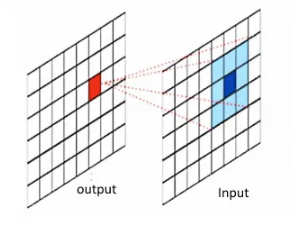


* + HOG (Histogram of Oriented Gradients),bir görüntünün lokal bölgelerdeki kenar ve köşe bilgilerini özetleyen bir özellik çıkarma yöntemidir. Bir görüntünün her bir bölgesinde kenar yönelimlerinin histogramlarını hesaplar. Bu yöntem insan tanıma için ilk bulunmuştur daha sonra diğer nesneler tanıması için kullanıldı.
    - Bu yaklaşımın esası, bir görüntünün her bölgesinde 36 boyutlu vektör, bu şekilde bir görüntü bu vektörlerinden 3. 780 boyutlu bir vektör ile temsil ediliyor.

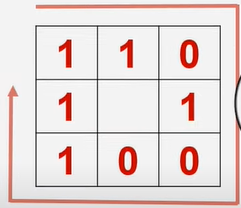
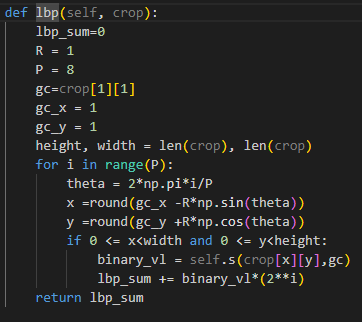


* Kod inceleme
  + LBP
    - travel metodu ile görüntüyü okunması ve convolution işlemiyle gri görüntye döndürmesini daha sonra görüntünün matrisinde gezmemizi ve kernel’e denk gelen pikselleri işlenmesi için lbp metoduna gönderilir.

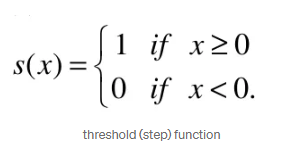


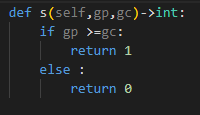


* + Lbp, Kernel metoduna gelen kernelin boyutu , komşu sayısını(p), kernelin ortasındaki pikselin şiddeti (gc) ve konumunu (gc\_x,gc\_y) alıyor ve komşu piksellerden bir 1D bir vektör oluşturuyoruz bu vector oluşturmak için üst-sol köşesinden başlayarak dairesel bir geçiş ile sıralı bir şekilde alınıyor bu hareketi sağlamak için komşuların koordinatları bu formulları kullanarak hesaplanıyor, x= (gc\_x-Rsin(2πp/P),y = (gc\_y + Rcos(2πp/P) ve her komşunun orta piksel ile karşılaştırılıyor eğer komşu pixel ortadaki pixel daha büyük yada eşitse 1 döndürür, aksi durumda 0 döndürür. En sonunda her gelen değer öncekileriyle toplanır ve sonucu döndürüyor.

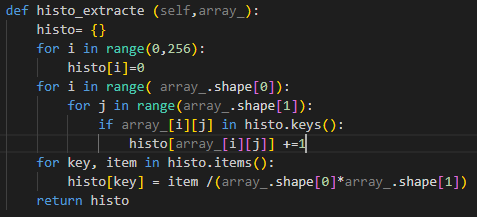


|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

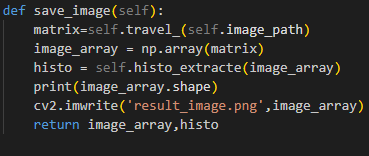




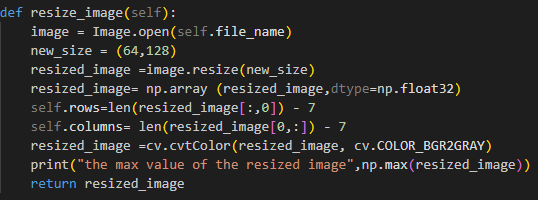
* + Histogram çıkarma, histogramı oluşturmak için sözlük yapısını kulladıldı sözlük anahtarları (key) 0dan 255’e kadar olarak ayarlandı ardından elde ettiğimiz siyah beyaz görüntüyü convolution yaparak bu piksellerin şiddetleri teker teker alıp ve eşit olduğu anahtarının değeri 1 artırıldı ve ardından görüntüdeki pixel sayısına sözlükteki tüm anahtarının değerleri bölerek normalize edildi.



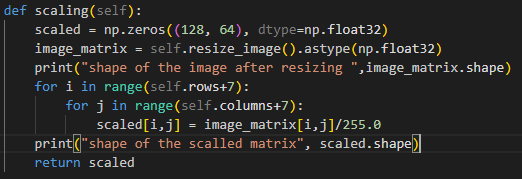
* + Sonuçları kaydetmek, LBP ‘den elde edilen binary görüntüyü openCv kütüphanedeki fonksiyonları kullanarak projenin klasöründe kaydedildi.



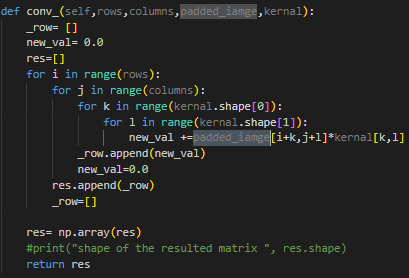
* HOG
  + Görüntü okuma ve yeniden boyutlandırma, pillow modul’unun Image sınıfındaki open metodu ile görüntün okuması sağlıyor ve ardından yeni boyutu 64x128 olarak resize metodu ile ayarlandı ilk convolution işlemi için görüntüyü 8x8 pencerelere ayırmak için self.row, self.column ayarlanıyor ve ondan sonra gri görüntüye çeviriyoruz.



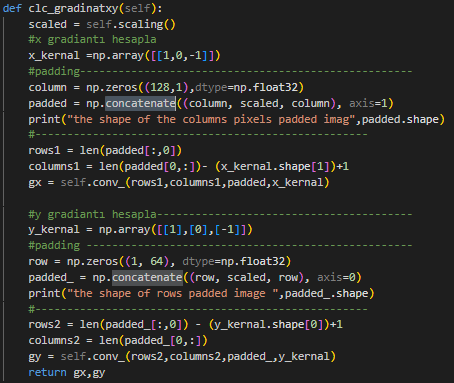
* + Pixellerin ölçeklendirme, hespalamalar daha kolay olması için ve pixellerin aralığı 0,255 olması yerine 0,1 arasında olarak ayarlıyoruz, dolayısıyla görüntüdeki tüm pixelleri 255’ye bölündü.



* + Convolution, convolution algoritmasını uygulanarak görüntüyü 8X8 boyutta parçalara ayırmasını sağlıyor ilk 2 for döngüsü görüntünün pixllerin tek tek geçmesi , 2. Iki for döngüsü maskenin uygulaması sağlıyor .



* + Gradiant hespalamak,bu işlemi iki aşamada gerçekleşildi;
    - X yönünde gradiant bulması, x yönünde türev almak için [1,0,-1] kernel’i kullanıldı ve convolution uygulamadan önce görüntüye paddig yapıldı her satırdan ilk ve son pixelleri sıfırlar ile doldururuz bu şekilde görüntünün boyutu 64x164 ise dogu yaptıktan sonra 64x166 olacaktır ve convolution uyguladıktan sonra tekrar orjinal boyuta geri dönüyor.
    - Y yönünde gradiant bulması, burada bu maskeyi kullandık , convolution uygulamadan önce görüntüye paddig yapıldı her sütünün ilk ve son pixelleri sıfırlar ile doldururuz bu şekilde görüntünün boyutu 64x164 ise dogu yaptıktan sonra 66x164 olacaktır ve convolution uyguladıktan sonra tekrar orjinal boyuta geri dönüyor.

Tüm 8x8 pencerelerin gradiyantı hesapladıkTan

sonra değişme miktarı (mignatude) ve Değişm

yönü (direction) hesaplayabiliriz bu şekilde Her

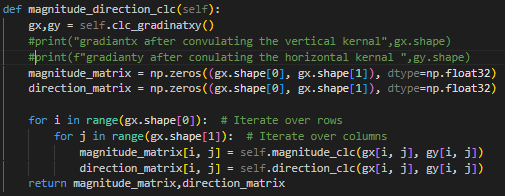
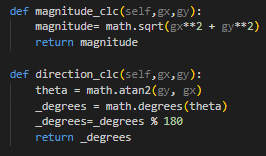
bir 8x8 pencere için 2 tane matris Mignatude

ve direction matris’i olur.

* + Değişme miktarı ve yönü hesaplama, pencindeki pixellerin değişme miktarı(magnitude) ve yönü hesaplaması bu iki formula göre gerçekleşti .

Magnitude Direction

 ,



elde ettiğimiz açılar 0 ile 360 arasındadır, bazı elde edilmiş açılar negative olabilir fakat bu yöntemde 0 ile 180 arasında olmasını gerekiyor direction\_clc fonksiyonu bunu sağladı.

* + Histogramı oluşturması, bunu yapmak için oylama yapılaması gerekiyor 0dan 160 kadar bir vektör oluşturulur bu vektör 9 bindan oluşmaktadır 0,20,40,60,,,,160 ardından her iki direction ve magnitude matrisi

