

BLM4540 Görüntü İşleme - Ödev 2

Toygar Tanyel
18011094

Yöntem

`class helper_functions` ve `class train`:

Bu classlar, görev gerçekleştirirken resim okuma dışında kütüphane kullanılamayacağı için, ihtiyacımız olacak olan fonksiyonları içermektedir.

```
class helper_functions:

    def __init__(self): return

    def read_image(self, img_path)
    # Resmi YCbCr'ye çevir ve Y (grayscale) değerlerini al

    def read_original_image(self, img_path)
    # Resmi RGB olarak oku

    def show_img(self, img_path)
    # Gerekliyse resmi göster (kodu yazarken kontrol için koymuştum)

    def get_pixel(self, img, center, x, y)
    # Pixel-wise karşılaştırma, LBP orta değerine etrafındaki 8
    değerin büyük küçük olma durumunu kontrol eden fonksiyon

    def calc_lbp(self, img, x, y)
    # Saat yönünde lbp değerlerini hesapla
    Burada derste gördüğümüz sıralama ile denediğimde sonuçlar çok
    iyi gelmedi.
    Buradaki makaledeki gibi yaptığımızda sonuçlar çok daha anlamlı:
https://www.researchgate.net/publication/305152373\_Texture\_Feature\_Extraction\_by\_Using\_Local\_Binary\_Pattern/figures

    def flatten(self, l)
    # LBP sonucunu uniform dağılıma indirgerken sonucu tek array
    olarak tutmak istiyorum. Dolayısıyla .ravel() kullanılması yasak
    olduğundan bu görevi yapan bir flatten fonksiyonu yazdım.
```

```

def uniform_lbp_array(self, image, uniform)
# uniform table üzerinden eldeki resmi [0,58] arasına indirgeme

def hist(self, lbp)
# histogram hesaplama

def manhattan_dist(self, a, b)
# manhattan distance hesabı

def get_three_min(self, manhattan_arr, n)
# En düşük uzaklığa sahip 3 resmin index'ini ve uzaklık değerini
dictionary olarak döndüren fonksiyon

def get_best_matches(self, train, result_dict, train_path)
# get_three_min ile bulunan dict verildiğinde bize en yakın 3
resmi orijinal halleriyle ekranda gösteren fonksiyon

def calculate_best_matches(self, train_dict, test_dict,
test_path)
# verilen test seti için get_three_min, get_best_matches ve
manhattan_dist fonksiyonlarını kullanarak en yakın matchleri hesaplayan
ve ekranda birleştiren fonksiyon

class train:
def __init__(self, dataset, training_path):
# Hangi veri setini kullanacak ismi ve path'i ile initilaze
ediyoruz. Daha sonra özelleşmiş train işlemlerini gerçekleştirebiliriz.
# Örnek:
# train_data = train("train", train_path)
# test_data = train("test", test_path)
# test_report = train("test_for_report", test_report_path)

def train(self):
if self.dataset == "train":
elif self.dataset == "test" or self.dataset ==
"test_for_report":
# Veri olarak ne gönderdiğimiz önemli çünkü train dosyaları alt
alta farklı dosyalarda bulunuyor. Test ve Rapor Test için bu geçerli
değil.
# Burada gerekli lbp dönüşümü ve histogram çıkarma işlemleri
yapılıyor.

```

```

def save_dictionary(self, dictionary, save_path)
    # Gönderdiğimiz train/test class ile elde ettiğimiz dictionary'i
save_path ile vereceğimiz yere save yapabiliriz.

def get_dictionary(self, save_path)
    # Aynı şekilde save_path'te bulunan hazır dictionary'leri direkt
geri yükleyebiliriz.

##### TRAIN-TEST VERİSİ İÇİN ÖRNEK İŞLEMLER: #####
train_data = train("train", train_path)
test_data = train("test", test_path)

train_dict= dict(sorted(train_dict.items(),key=lambda x: x[0].lower()))
train_data.save_dictionary(train_dict, save_path)

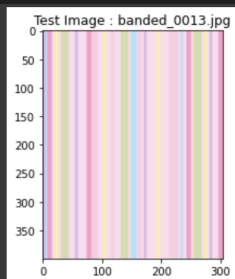
test_dict = test_data.train()

test_dict = dict(sorted(test_dict.items(), key=lambda x: x[0].lower()))
test_data.save_dictionary(test_dict, save_path)

helper = helper_functions()
helper.calculate_best_matches(train_dict, test_dict, test_path)

## Bu işlemler sonucunda train histogramları ile test histogramlarını
karşılaştırarak en alakalı sonuçları getirmektedir. Örnek:

```

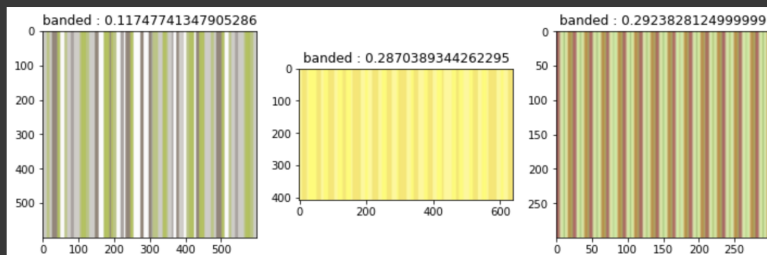


---- BEST THREE MATCHES ----:

```

--> 1. banded_0006.jpg : 0.11747741347905286
--> 2. banded_0012.jpg : 0.2870389344262295
--> 3. banded_0002.jpg : 0.2923828124999999

```



Uygulama

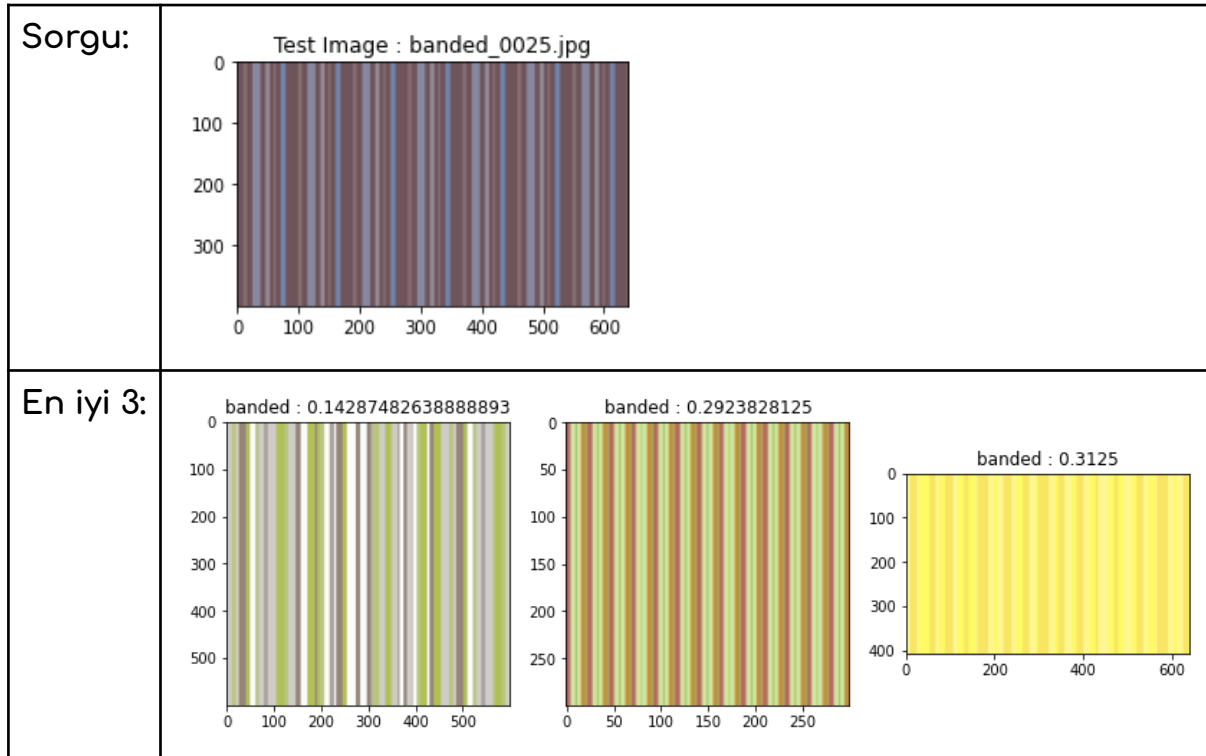
- 1) **test directory'sindeki** her test resmi için en benzer 3 resmi bulunuz. Eğer 3 resimden en az 1'i test resmi ile aynı sınıfta ise doğru sonuç, hiç benzer yok ise yanlış olarak hesaplayınız. Her sınıf için doğru bulma oranı ve ortalama doğru bulma oranını veriniz.

Doğruluk oranları tablodaki gibidir:

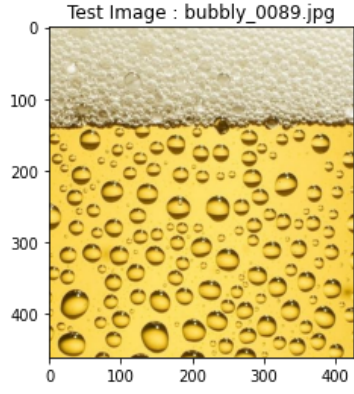
test	<i>banded</i>	<i>bubbly</i>	<i>chequered</i>	<i>cobwebbed</i>	<i>crystalline</i>	<i>dotted</i>	<i>honeycombed</i>	<i>striped</i>	<i>zigzagged</i>
ort: 16/27	2/3	1/3	1/3	2/3	3/3	0/3	2/3	3/3	2/3
test_rapor	<i>banded</i>	<i>bubbly</i>	<i>chequered</i>	<i>cobwebbed</i>	<i>crystalline</i>	<i>dotted</i>	<i>honeycombed</i>	<i>striped</i>	<i>zigzagged</i>
ort: 8/9	1/1	1/1	0/1	1/1	1/1	1/1	1/1	1/1	1/1
overall:24/36	3/4	2/4	1/4	3/4	4/4	1/4	3/4	4/4	3/4

- 2) **testRaporaEklenecek directory'sindeki** her resim için en benzer 3'er resmi bularak **raporunuza ekleyiniz**. Resimleri küçülterek kullanınız. Her sorgu resminin yanına 3 benzeri ve her benzer resmin, sorgu resmine mesafesini de altına yazarak veriniz.

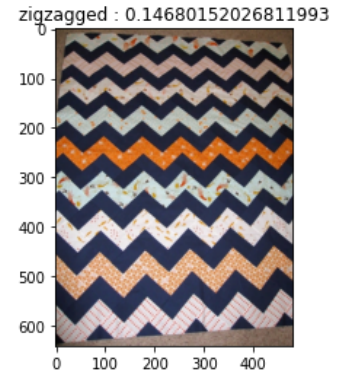
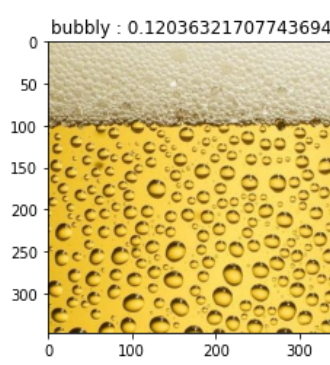
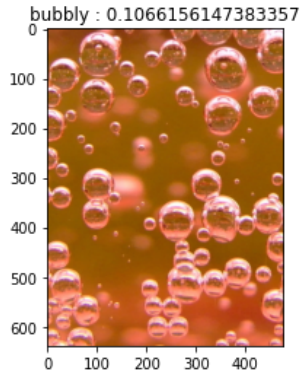
Sorgular:



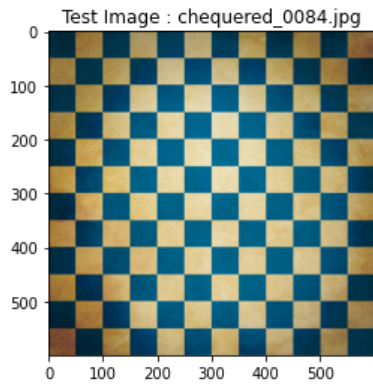
Sorgu:



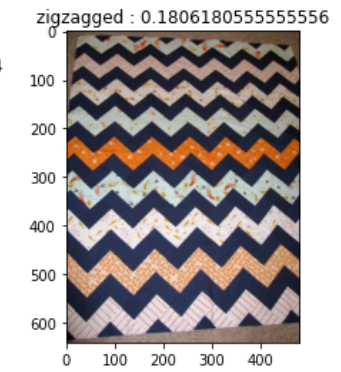
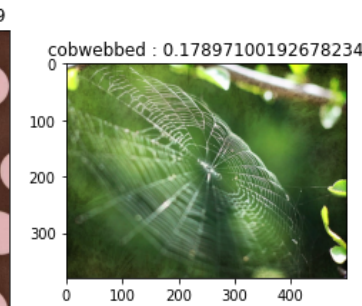
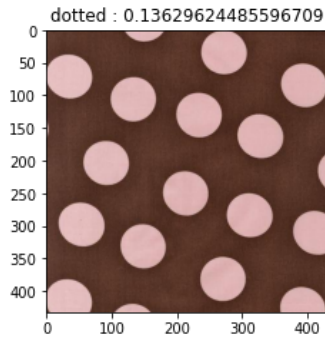
En iyi 3:



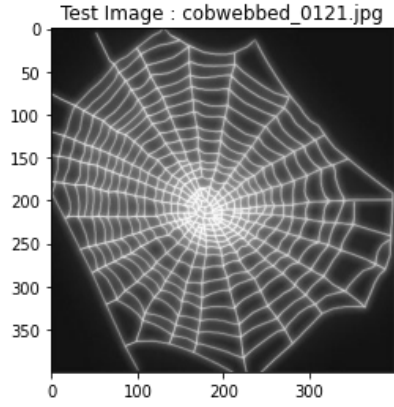
Sorgu:



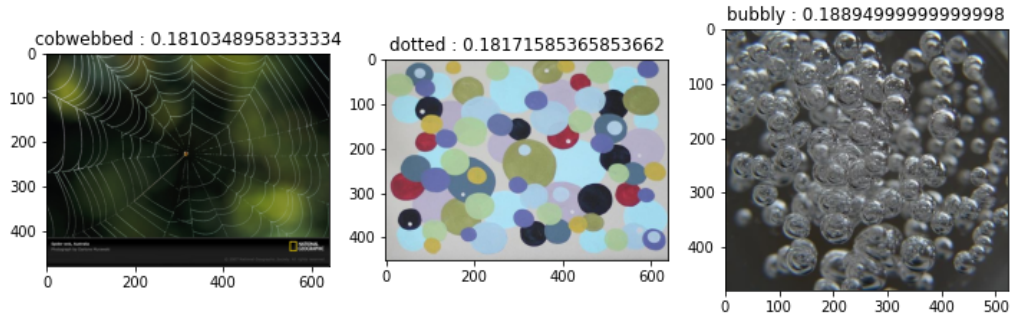
En iyi 3:



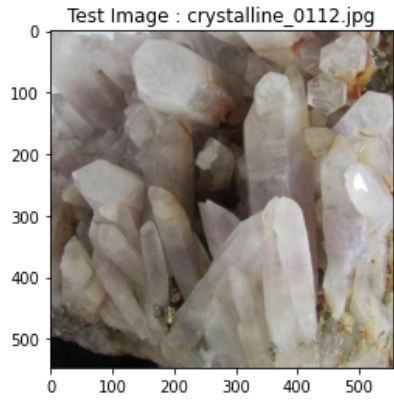
Sorgu:



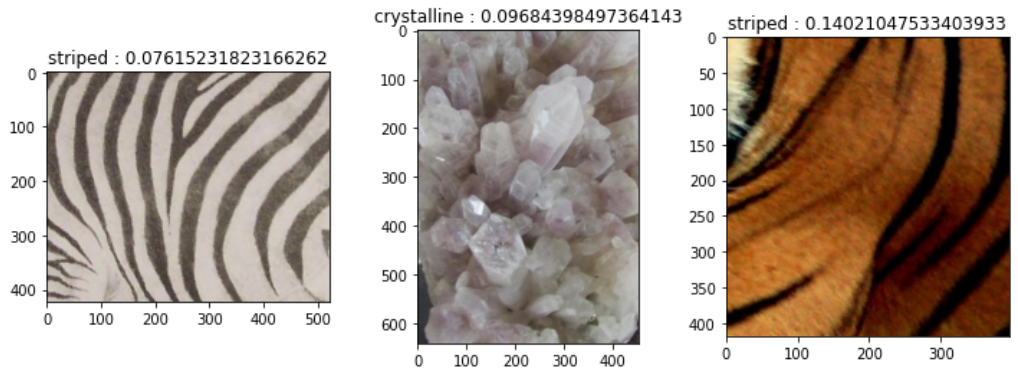
En iyi 3:



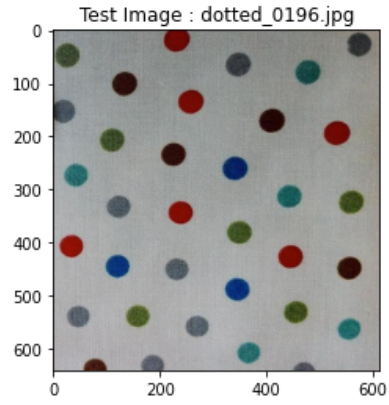
Sorgu:



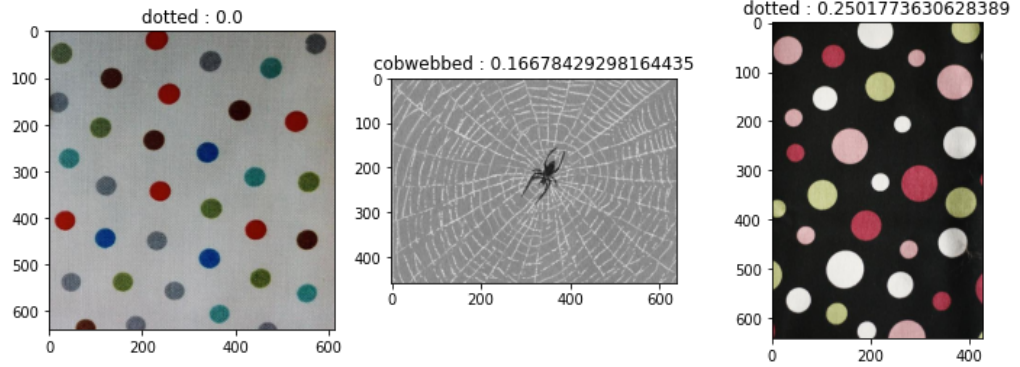
En iyi 3:



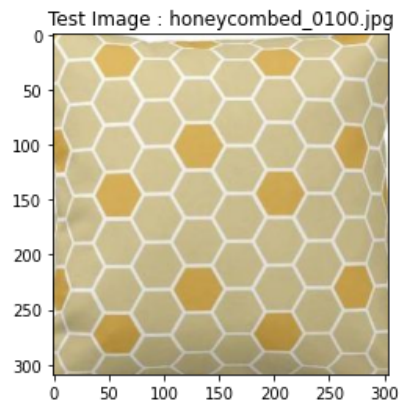
Sorgu:



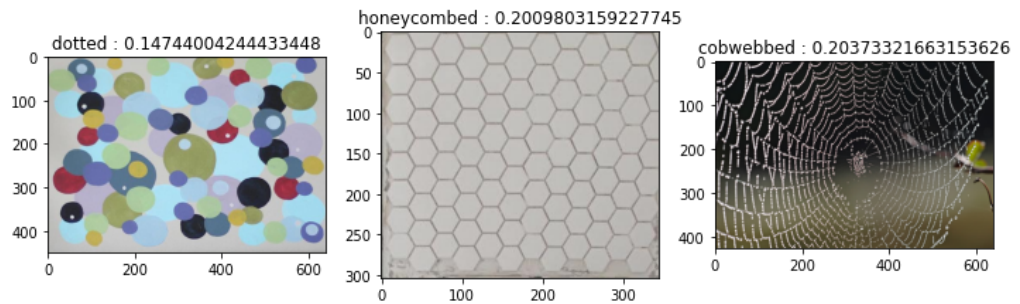
En iyi 3:


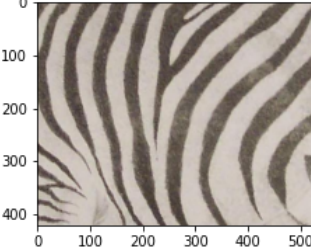
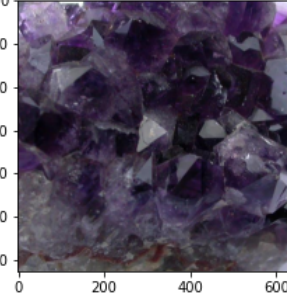
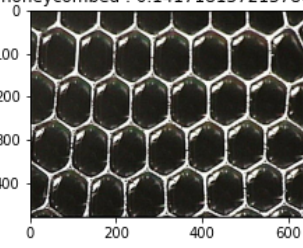
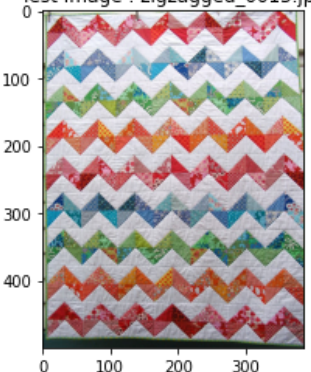
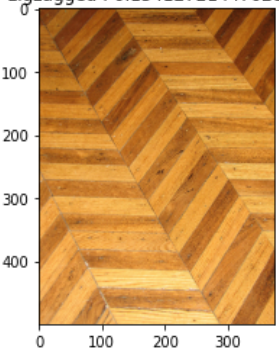
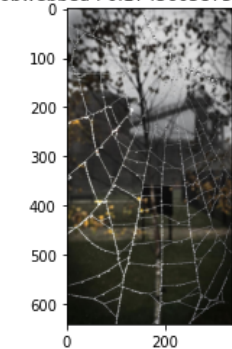
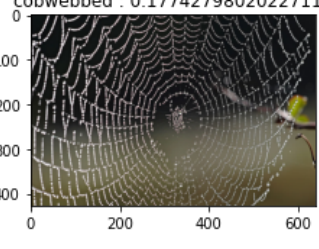


Sorgu:



En iyi 3:

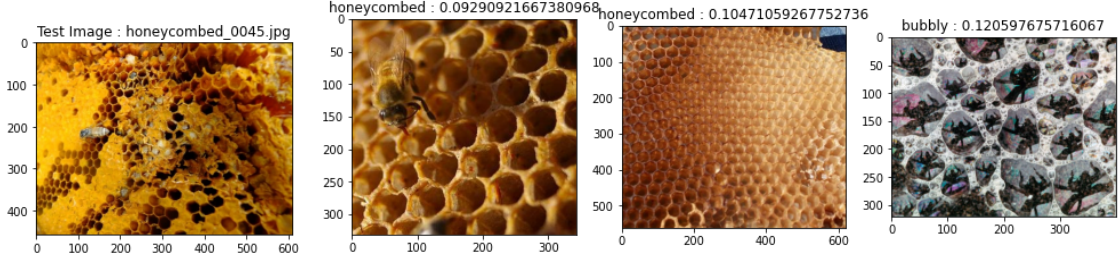


Sorgu:	<p>Test Image : striped_0002.jpg</p> 
En iyi 3:	<div> <div> <p>striped : 0.11473672511657276</p>  </div> <div> <p>crystalline : 0.14084985800694666</p>  </div> <div> <p>honeycombed : 0.14171813721378693</p>  </div> </div>
Sorgu:	<p>Test Image : zigzagged_0015.jpg</p> 
En iyi 3:	<div> <div> <p>zigzagged : 0.1541272144702842</p>  </div> <div> <p>cobwebbed : 0.17456058751529996</p>  </div> <div> <p>cobwebbed : 0.17742798020227113</p>  </div> </div>

Sonuç

LBP yöntemi pek çok pattern için oldukça iyi sonuçlar vermektedir. Çoğu zaman dokusu karmaşık olan resimler için class'ı yanlış tahmin etse de yaptığı yanlışlar anlaşılabilir.

Örnek olarak aşağıdaki örnekte bubbly_0046.jpg için yakın bir benzerlik bulmuş. Fakat doku anlamında düşünüldüğünde bubbly'nin bu örneği de honeycombed yapısına çok benzemektedir.



Yukarıdakinin aksine tamamen bilemediği örneklerde de bu tarz faktörler etkisini göstermektedir.

Bu yöntemi kullanmanın en büyük avantajı, diğerlerine göre çok daha küçük bir vektör tutması [0,58] ve hesaplama için çok düşük gereksinimleri olmasıdır. Diğer yöntemlere oranla hem yer hem hız avantajı vardır. Işık değişimlerine duyarlı değildir dolayısıyla farklı parlaklıklardaki resimlerin dokularını da yakalayabilmektedir.

Ayrıca bize verilen bazı test örneklerinin arasında ya resmin kaydırılmış hali ya da birebir train görselleri içerisinde olduğu durumlar da vardı. Bunlara örnek olarak, (bubbly_0089.jpg, bubbly_0061.jpg) ve dotted_0196.jpg verilebilir.

Doğal olarak birebir aynı resmi 0 fark ile en yakın getirmektedir. Benzer olanları da kayma vb. durumlar olmasına rağmen kaçırmadan en benzerler arasına almaktadır.