Hasarlı Araç Tespiti

Esat Berat Uzunca
221307044

İbrahim Buğra San

221307059

I. GİRİŞ

Bu projede, Python programlama dili kullanılarak, web üzerindeki görsel içeriklerin otomatik olarak taranması ve indirilmesi için bir web tarayıcı (web crawler) geliştirilmiştir. Geliştirilen sistem, Selenium kütüphanesi ile tarayıcı otomasyonu sağlayarak hedef web sayfalarını ziyaret eder ve BeautifulSoup kütüphanesi yardımıyla **HTML** yapısını analiz ederek görselleri belirler. Belirlenen görseller, **Pillow (PIL)** kütüphanesi aracılığıyla işlenerek belirtilen boyutlara **(512x512)** yeniden boyutlandırılır ve yerel bir dizine kaydedilir. Kod, hasarlı araçların görüntülerinin toplanması gibi belirli kullanım alanlarına yönelik olarak özelleştirilmiş, genişletilebilir ve otomatikleştirilmiş bir çözüm sunmaktadır. Bu raporda, kullanılan teknikler, karşılaşılan zorluklar ve çözüm yolları ayrıntılı olarak ele alınacaktır Ease of Use

II. YÖNTEMLER ARAÇLAR VE TEKNOLOJİLER

Bu projede, web sayfalarından görsel veri toplama ve işleme süreçlerinin etkin bir şekilde gerçekleştirilmesi için bir dizi ileri düzey teknik kullanılmıştır. Dinamik içeriklerin yüklenmesi ve web sayfalarına programatik erişim sağlanması için **Selenium WebDriver** kullanılmıştır. HTML yapısının ayrıştırılması ve hedef görsellerin bulunması işlemleri, esnek ve hassas veri çıkarmaya olanak sağlayan **BeautifulSoup** kütüphanesiyle gerçekleştirilmiştir. Görsellerin URL üzerinden indirilmesi için **Requests** kütüphanesi kullanılmış ve indirilen görsellerin yeniden boyutlandırılması, format dönüşümü ve yerel dosya sistemine kaydedilmesi süreçlerinde **Pillow (PIL)** kütüphanesinden yararlanılmıştır. Performansı artırmak amacıyla sistem, **headless tarayıcı modu** ile çalıştırılmış ve dinamik URL yapıları kullanılarak çok sayıda sayfa arasında sorunsuz geçiş sağlanmıştır. Bu yöntemler, büyük ölçekli veri toplama ve işleme ihtiyaçlarını karşılamak üzere tasarlanmıştır.

A. Web Tarayıcı Otomasyonu

Web tarayıcı otomasyonu için Selenium kullanılmıştır. Dinamik web sayfalarına erişim sağlamak, bu sayfalarla etkileşim kurmak ve içeriği ayrıştırmak için etkili bir araç olan Selenium, projede tarayıcıyı başlatma, belirli bir URL'ye erişme ve sayfanın tamamen yüklenmesini bekleme gibi görevleri yerine getirmiştir. İşlemlerin daha hızlı ve verimli bir şekilde gerçekleştirilmesi için headless tarayıcı modu tercih edilmiş, böylece tarayıcı arayüzü görüntülenmeden işlemler arka planda yürütülmüştür. Çoklu sayfalardan veri toplayabilmek için URL yapısı dinamik olarak oluşturulmuş ve sayfa numaraları döngüye alınmıştır. Bu yapı, her sayfadan veri toplanmasını mümkün kılmıştır.

B. HTML Ayrıştırma Ve Veri Çıkarma

HTML analizi ve veri çıkarma süreçlerinde **BeautifulSoup** kütüphanesi kullanılmıştır. Selenium tarafından yüklenen sayfaların **HTML** yapısı, BeautifulSoup ile ayrıştırılarak görsellerin bulunduğu `<img>` etiketleri ve ilgili sınıf adları tespit edilmiştir. Görsellerin kaynak **URL**'leri, bu etiketlerin `src` öznitelikleri üzerinden çıkarılmıştır. Geçersiz **URL'ler** ve boş öznitelikler kontrol edilerek yalnızca doğrulanmış verilerin işlenmesine izin verilmiştir. Bu yöntem, veri doğruluğunu artırmış ve işleme alınacak görsellerin belirlenmesini kolaylaştırmıştır.

C. Görsellerin İndirilmesi İşlenmesi

İndirilen görsellerin kaynak URL'lerinden alınması için Requests kütüphanesi kullanılmıştır. **HTTP** isteklerinin durum kodları kontrol edilerek yalnızca başarılı isteklerde veri işleme süreci devam ettirilmiştir. İndirilen görseller, **Pillow (PIL)** kütüphanesi ile **512x512** piksel boyutunda yeniden boyutlandırılmış ve **JPEG** formatında kaydedilmiştir. Bu işlem, görsellerin standart bir boyuta getirilmesini ve depolama alanının daha verimli kullanılmasını sağlamıştır. Hatalı veya indirilemeyen görseller için hata yönetimi mekanizmaları devreye alınarak süreçte oluşabilecek aksaklıklar en aza indirilmiştir..

III. KARŞILAŞILAN ZORLUKLAR VE ÇÖZÜMLER

Proje sürecinde karşılaşılan bazı zorluklar, özellikle web sayfalarından veri çekme ve görsellerin işlenmesiyle ilgili oldu. İlk zorluk, dinamik web sayfalarının içeriklerini doğru bir şekilde almak için gerekli olan sayfa yükleme sürelerinin değişkenlik göstermesiydi. Bu durumu aşmak için **Selenium'un** bekleme fonksiyonları ve sayfanın tamamen yüklenmesini bekleme yöntemleri kullanıldı. Ayrıca, bazı web sitelerindeki görsellerin **URL'lerinin** doğruluğu sorun yaratıyordu, çünkü geçersiz veya bozuk bağlantılar olabiliyordu. Bu sorunun üstesinden gelmek için Requests kütüphanesiyle **HTTP** isteklerinin durum kodları kontrol edilerek yalnızca geçerli veriler işleme alındı. Diğer bir zorluk ise, görsellerin boyutlarının standart hale getirilmesiydi; çünkü her görsel farklı boyutlarda ve formatlarda olabiliyordu. Bu durumda, **Pillow** kütüphanesi ile görsellerin boyutları yeniden ayarlandı ve **JPEG** formatına dönüştürüldü. Son olarak, büyük miktarda veri işlenirken bilgisayarın kaynaklarını verimli kullanmak ve hızla sonuç almak adına **headless** modda çalışma ve tarayıcı işlemlerini arka planda tutma gibi optimizasyonlar yapıldı. Bu çözüm önerileri, proje sürecindeki verimliliği artırarak, karşılaşılan zorlukların üstesinden gelmemize yardımcı oldu.

A. Dinamik İçeriklerin Yüklenmesi

Bazı web sayfalarında dinamik içeriklerin yüklenmesi için ek zaman gereklidir, bu da sayfa tam olarak yüklenmeden veri çekmeye çalışıldığında hatalara yol açabilmektedir. Bu durumu aşmak için, **Selenium'da time.sleep()** fonksiyonu kullanılarak sayfanın yüklenmesi için sabit bekleme süreleri tanımlanmıştır. Ancak daha iyi bir çözüm sağlamak için, **Selenium'un WebDriverWait** ve **ExpectedConditions** fonksiyonlarının entegrasyonu önerilmektedir. Bu yöntemle, sayfanın belirli elementleri yüklenene kadar bekleme süreci dinamik olarak yönetilebilir

B. Büyük Veri İşleme ve Performans Sorunları

Çok sayıda sayfadan veri toplama ve görsellerin indirilmesi işlemi, sistem kaynakları üzerinde ciddi performans sorunlarına yol açtı. Özellikle yüksek işlemci ve bellek kullanımı, işlemlerin verimli bir şekilde yapılmasını

engellemiştir. Performans sorunlarını çözmek amacıyla, **headless** tarayıcı modu kullanılarak görsel işleme işlemi arka planda gerçekleştirilmiştir. Ayrıca, indirme sırasında görsellerin boyutları optimize edilerek daha az bant genişliği kullanımı sağlanmıştır. Gelecekte, çok iş parçacıklı (multi-threading) bir yapı ile işlem süresi daha da kısaltılabilir ve performans artırılabilir..

C. Eksik veya Geçersiz URL Sorunları

Bazı durumlarda, sayfalardan elde edilen görsel URL'leri eksik, hatalı veya erişilemez olabiliyordu. Bu tür sorunlar, veri bütünlüğünü bozmakta ve bazı içeriklerin işlenmesini engellemektedir. Bu sorunları çözmek için, her bir URL'nin doğruluğu kontrol edilmiştir. Geçersiz URL'ler, hata yönetimi mekanizmalarıyla atlanarak işleme alınmamıştır.

D. Dosya Adlandırma ve Çakışma Sorunları

Görsellerin dosya sistemine kaydedilmesi sırasında, aynı isimde dosyaların oluşması ve bunun sonucunda dosya çakışmalarının yaşanması söz konusu olmuştur. Bu durumda, bazı dosyalar üzerine yazılabilir, bu da veri kaybına yol açabilirdi. Çakışma sorunlarını önlemek için, her görsele sayfa numarası ve sıra numarası eklenerek dosya adlandırma süreci yapılmıştır. Bu yöntem, her görselin benzersiz bir isme sahip olmasını sağlayarak çakışmaları etkili bir şekilde önlemiştir.

E. Siteye Erişim Engelleri

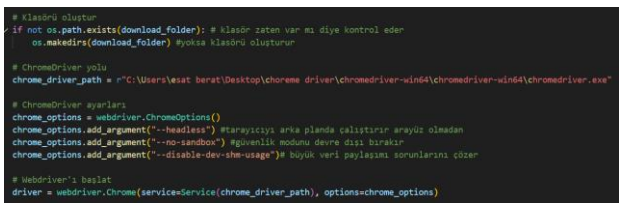
Bazı durumlarda, web siteleri çoklu isteklerin yapılmasını engelleyerek, sürekli erişim sağlayan IP adreslerini engellemiş veya web tarayıcısını bloke etmiştir. Bu, veri çekme sürecinde kesintilere yol açabilir. Bu sorunu aşmak için, sisteme rastgele bekleme süreleri eklenerek isteklerin daha doğal bir şekilde yapılması sağlanmıştır. Ayrıca, **proxy** kullanımı ile **IP** değişimi gerçekleştirilerek engellemelerden kaçınılmıştır. Uzun süreli kullanım için daha güvenilir ve sürdürülebilir veri erişim yöntemleri planlanmış ve uygulanması düşünülmüştür.

IV. KODUN İŞLEYİŞ YAPISI

Bu bölümde, hazırlanan kodun ana akışı ve işleyiş yapısı detaylı bir şekilde ele alınmıştır. Kod, veri toplama ve işleme süreçlerini bir araya getirerek organize bir yapı oluşturur.

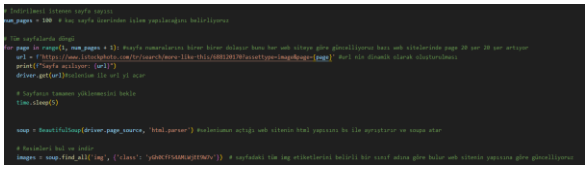
A. Klasör Ve Çevresel Ayarlar

Kodun başlangıç aşamasında, indirilen görsellerin kaydedileceği bir klasör oluşturulmuştur. Eğer belirtilen klasör sistemde önceden mevcut değilse, **os.makedirs()** fonksiyonu kullanılarak klasör otomatik olarak yaratılmıştır. Tarayıcı otomasyonu için ise **Selenium WebDriver** tercih edilmiştir. **Google Chrome** tarayıcısı, **headless** modda çalıştırılarak görsel arayüz olmadan işlemlerin arka planda hızlı ve düşük kaynak tüketimiyle gerçekleştirilmesi sağlanmıştır. Ayrıca, **no-sandbox** ve **disable-dev-shm** ayarları, güvenlik sorunlarını ve bellek paylaşımı kaynaklı hataları önlemek için kullanılmıştır.



B. Sayfa Gezinme ve Veri Çekme

Dinamik veri toplama süreci için birden fazla sayfanın işlenmesine olanak tanıyan bir **URL** yapısı oluşturulmuştur. Bu amaçla for döngüsü kullanılarak her sayfa için dinamik URL'ler oluşturulmuş ve **Selenium** aracılığıyla ilgili sayfalara erişim sağlanmıştır. Sayfa içeriklerinin tamamen yüklenmesi için **time.sleep()** fonksiyonu kullanılmıştır. Bu bekleme süresi, veri eksikliğini önleyerek işlem sırasında hata oluşmasını engellemiştir.

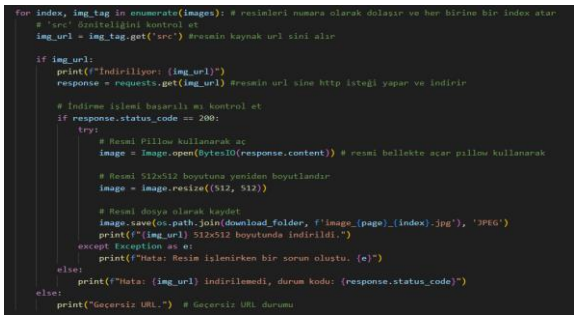


C. HTML Analizi ve Görsel Tespiti

Sayfalardan veri toplamak için **Selenium** tarafından yüklenen **HTML** içerikleri **BeautifulSoup** kütüphanesi ile analiz edilmiştir. Sayfalardaki **<img>** etiketleri ve bu etiketlere bağlı belirli sınıflar yardımıyla görseller tespit edilmiştir. Her bir **<img>** etiketinin **src** özelliği kontrol edilerek geçersiz veya boş **URL'ler** işleme alınmamıştır. Yalnızca geçerli **URL'ler** doğrulanarak indirme sürecine dahil edilmiştir.

D. Görsellerin İndirilmesi ve İşlenmesi

Görsellerin kaynak **URL'lerden** indirilebilmesi için Requests kütüphanesi kullanılmıştır. Her bir **HTTP** isteğinin durum kodu kontrol edilerek yalnızca başarılı istekler doğrultusunda işlem devam ettirilmiştir. İndirilen görseller, **Pillow** kütüphanesi yardımıyla **512x512** piksel boyutuna yeniden boyutlandırılmış ve **JPEG** formatında kaydedilmiştir. Görsel işleme sırasında meydana gelebilecek olası hatalar için **try-except** blokları ile hata yönetimi uygulanmıştır



E. Dosya Adlandırma ve Kaydetme

Her görsel, benzersiz bir şekilde adlandırılarak kaydedilmiştir. Dosya çakışmalarını önlemek amacıyla adlandırma sırasında her görsele sayfa numarası ve görsel sıra numarasını içeren bir isim verilmiştir. Bu yöntem, verilerin düzenli ve organize bir şekilde belirtilen klasör içerisine kaydedilmesini sağlamıştır.

F. Tarayıcının Kapatılması

Kod çalışmasının sonunda, tarayıcı oturumu **driver.quit()** komutuyla sonlandırılmıştır. Bu işlem, kaynakların verimli kullanılması için açık kalan süreçlerin kapatılmasını ve belleğin serbest bırakılmasını sağlamıştır. Bu sayede, sonraki işlemler için sistem kaynaklarının kullanılabilirliği artırılmıştır.

```
# Tarayıcıyı kapat
driver.quit()
```