

Bir Araç Plakasını Bilgisayar Nasıl Okur?

. . .

YOLO11n ve OCR ile Uçtan Uca Plaka Tanıma: İki Farklı Yaklaşımın Karşılaştırmalı Analizi

Bir aracın fotoğrafına baktığımızda plakanın nerede olduğunu bulmak ve üzerindeki karakterleri okumak bizim için oldukça basit bir işlem.

Peki aynı görüntüyü bir bilgisayara verdiğimizde ne oluyor?

Bilgisayar için burada aslında tek bir problem yok.

Önce görüntü içerisindeki **plakanın nerede olduğunu bulması**, ardından bulduğu bölgedeki **harf ve rakamları doğru şekilde okuyarak metne dönüştürmesi** gerekiyor.

Tam olarak bu noktada görüntü işleme, nesne tespiti ve optik karakter tanıma (OCR) birlikte devreye giriyor.

Bu projede temel olarak şu sorunun peşinden gittim:

Bir araç görüntüsünden plakayı otomatik olarak tespit edip üzerindeki karakterleri ne kadar doğru okuyabiliriz?

Bu soruya cevap ararken yalnızca tek bir model ve tek bir yöntem kullanmak yerine, farklı veri setleri, eğitim stratejileri ve OCR yaklaşımlarıyla oluşturduğum **iki ayrı plaka tanıma sistemini** deneysel olarak karşılaştırdım.

İlk sistemde **1.653 görüntülük** daha büyük ve çeşitli bir veri seti kullandım. Plaka tespiti için **YOLO11n**, karakter tanıma için ise **EasyOCR** kullandım.

İkinci sistemde ise **432 görüntülük** daha küçük bir veri setiyle çalıştım. Yine YOLO11n kullandım; ancak karakter tanıma aşamasını

genişleterek EasyOCR ve PaddleOCR sonuçlarını birlikte değerlendirdim.

Bu iki yaklaşımı karşılaştırırken yalnızca “hangi model daha iyi?” sorusuna odaklanmadım.

Aynı zamanda;

- veri seti büyüklüğünün,
- veri çeşitliliğinin,
- eğitim stratejisinin,
- görüntü ön işlemenin,
- OCR motorunun,
- birden fazla OCR sonucunu birleştirmenin

sistem performansını nasıl değiştirdiğini incelemeye çalıştım.

Son aşamada ise geliştirdiğim modelleri Tkinter tabanlı masaüstü uygulamalarına entegre ederek sistemi yalnızca eğitim ve test aşamasında çalışan bir model olmaktan çıkarıp kullanılabilir bir uygulamaya dönüştürdüm.

. . .

Problemi İkiye Ayıralım: Plaka Tespiti + OCR

Bir araç görüntüsünden plaka bilgisini elde etmek, aslında iki temel bilgisayarlı görü probleminin ardışık olarak çözülmesini gerektiriyor.

1. Plaka nerede?

→ Nesne Tespiti

2. Plakanın üzerinde ne yazıyor?

→ Optik Karakter Tanıma (OCR)

Sistemin genel akışını şu şekilde düşünebiliriz:



İlk aşamada YOLO11n görüntü içerisindeki plaka bölgesini tespit ediyor.

Ardından bu bölge görüntüden kırpılıyor.

Kırılan plaka görüntüsü çeşitli ön işleme adımlarından geçirildikten sonra OCR motorlarına gönderiliyor.

Son aşamada ise OCR tarafından üretilen metinler değerlendirilerek nihai plaka sonucu oluşturuluyor.

Buradaki kritik nokta, bu aşamaların birbirinden bağımsız düşünülmemesi.

YOLO yanlış bölgeyi tespit ederse OCR'a yanlış görüntü gönderiliyor.

YOLO doğru plakayı bulsa bile plaka görüntüsü düşük kaliteli ise OCR karakterleri yanlış okuyabiliyor.

Dolayısıyla ortaya çıkan sistemin başarısı tek bir modelin performansından değil, **uçtan uca pipeline'ın bütününden** etkileniyor.

. . .



İki Farklı Yaklaşım

Projede aynı temel problemi iki farklı şekilde ele aldım.

Özellik	Proje A	Proje B
Veri seti	1.653 görüntü	432 görüntü
Veri kaynağı	Video kareleri + internet	Ağırlıklı internet görselleri ▾
Tespit modeli	YOLO11n	YOLO11n
Eğitim stratejisi	5 eğitim/fine-tuning denemesi	Tek eğitim, 50 epoch
OCR	EasyOCR	<u>EasyOCR + PaddleOCR</u>
Ön işleme	5 varyant × 2 ölçek	9 varyant × 2 OCR motoru
OCR adayları	10	18
Karar mekanizması	Çoğunluk oylaması + Regex	Ağırlıklı skor oylaması
Arayüz	Tkinter	Tkinter + toplu test arayüzü

Bu iki yaklaşım arasındaki en önemli farklardan biri veri miktarı.

Proje A'da **1.653 görüntü**, Proje B'de ise **432 görüntü** bulunuyor.

Dolayısıyla yalnızca OCR mimarisini değil, aynı zamanda **veri miktarı ve eğitim stratejisinin model performansına etkisini** de gözlemleme fırsatım oldu.

. . .



Neden YOLO11n?

Plaka tespiti için kullanılabilecek birçok nesne tespit modeli bulunuyor. Bu projede ise **YOLO11n (nano)** modelini tercih ettim.

Buradaki amaç yalnızca mümkün olan en yüksek doğruluk değerine ulaşmak değildi.

Aynı zamanda modelin:

- hızlı çalışması,
- düşük hesaplama maliyetine sahip olması,
- masaüstü uygulamasına entegre edilebilmesi,
- inference aşamasında pratik olması

benim için önemliydi.

Plaka tespiti özelinde modelin onlarca farklı nesneyi birbirinden ayırması gerekmiyor. Temel olarak tek bir sınıfla, yani **plate** sınıfıyla çalışıyoruz.

Bu nedenle daha büyük bir model yerine daha hafif bir YOLO mimarisinin yeterli olup olmayacağını test etmek istedim.

Burada model seçimindeki temel düşüncem:

Doğruluk + hız + hesaplama maliyeti arasında dengeli bir çözüm bulmak.

YOLO11n'in görevi ise yalnızca plakayı bulmak.

Karakterleri okumak modelin görevi değil.

Bu nedenle pipeline'ı iki farklı uzmanlık alanına ayırdım:

YOLO11n
"Plaka nerede?"

+

OCR
"Plakada ne yazıyor?"

Bu ayırım, projenin geri kalan mimarisinin de temelini oluşturdu.

. . .



Veri Setlerini Nasıl Hazırladım?

Model performansını belirleyen en önemli faktörlerden biri kullanılan verinin niteliği.

Bu nedenle projede iki farklı ölçekte veri seti kullandım.

Proje A—1.653 Görüntü

İlk yaklaşımda toplam 1.653 görüntü kullandım.

Bunun:

- 1.486 görüntüsü training
- 167 görüntüsü validation

olarak ayrıldı.

Veri setinin yaklaşık %30,3'ü video karelerinden, %69,7'si ise internet kaynaklı görüntülerden oluşuyordu.

Burada özellikle veri sızıntısını önlemeye dikkat ettim.

Validation setindeki görüntülerin eğitim görüntüleriyle aynı video kaynaklarından gelmemesine dikkat ederek doğrulama sürecini daha güvenilir hale getirmeye çalıştım.

Validation tarafında Video 5, 6, 8 ve 9 gibi eğitim setinden farklı kaynaklar kullanıldı.

Bu önemli çünkü aynı videodan alınmış birbirine çok benzeyen karelerin hem training hem validation içerisinde bulunması model performansının gerçekte olduğundan daha yüksek görünmesine neden olabilir.

. . .



Proje B—432 Görüntü

İkinci yaklaşımda ise 432 görüntülük daha küçük bir veri seti kullandım. Bu veri setindeki etiketler başlangıçta Pascal VOC XML formatındaydı. Etiketleri YOLO formatına dönüştürdükten sonra veri setini:

- %80 training
- %20 validation

şeklinde böldüm.

Bölme işlemini `seed=42` kullanarak gerçekleştirdim. Burada iki yaklaşım arasındaki önemli fark ortaya çıkıyor:

Proje A:

```
1.653 görüntü  
↓  
5 eğitim/fine-tuning aşaması
```

Proje B:

```
432 görüntü  
↓  
Tek eğitim  
↓  
50 epoch
```

Dolayısıyla sonuçları değerlendirirken yalnızca OCR farkına değil, veri miktarı ve eğitim stratejisine de dikkat etmek gerekiyor.

. . .



YOLO11n ile Plaka Tespiti

Model eğitildikten sonra YOLO11n, görüntü içerisindeki plaka bölgesini bounding box ile işaretliyor.

Örneğin modelin ürettiği bir çıktı şu şekilde olabilir:

```
Class: 0 (plate)
```

```
x1 = 245  
y1 = 318  
x2 = 492  
y2 = 386
```

```
Confidence = 0.933
```

Buradaki koordinatlar plakanın görüntü içerisindeki konumunu belirliyor. Daha sonra bu koordinatlar kullanılarak plaka görüntüden kırılıyor. Bu noktada crop işleminin nasıl yapıldığı da önemli. Çünkü bounding box karakterlerin hemen kenarında kalırsa OCR için gerekli bazı pikseller kaybolabilir. Bu nedenle iki projede farklı crop stratejileri kullandım.

Proje A

Plaka crop'una sabit 20 piksel kenar payı ekledim.

Proje B

Kenar payını görüntünün boyutuna göre oransal olarak genişlettim:

- yatayda %15
- dikeyde %25

Bu yaklaşımın amacı, OCR'a yalnızca karakterlerin bulunduğu minimum bölgeyi değil, karakterlerin çevresindeki gerekli bağlamı da sağlayabilmektir.

. . .

OCR Aşaması: Plaka Görüntüsünü Okumak

YOLO11n'in görevi tamamlandıktan sonra artık ikinci probleme geçiyoruz:

Plakanın üzerinde ne yazıyor?

Burada OCR devreye giriyor.

Ancak plaka görüntüsünü doğrudan OCR motoruna göndermek her zaman iyi sonuç vermiyor.

Çünkü gerçek dünya görüntülerinde;

- düşük çözünürlük,
- hareket bulanıklığı,
- perspektif,
- ışık yansımaları,
- düşük kontrast,
- kir,
- karakterlerin birbirine benzemesi

gibi problemler bulunabiliyor.

Örneğin OCR açısından:

O ↔ 0
I ↔ 1
S ↔ 5
B ↔ 8
G ↔ 6

gibi karakterler birbirine karışabiliyor.

Bu nedenle OCR aşamasını tek bir görüntü üzerinden değil, farklı **preprocessing varyantları** üzerinden ele aldım.

. . .



Görüntü Ön İşleme Pipeline'ı

Plaka crop'u üzerinde farklı görüntü işleme yöntemleri denedim.

Genel yapı:

Plaka Crop
↓
Resize
↓
Grayscale
↓
Kontrast / Threshold
↓
Sharpening
↓
OCR

Buradaki amaç görüntüyü estetik olarak iyileştirmek değil.

Asıl amaç:

Karakterleri OCR motorunun daha kolay ayırt edebileceği hale getirmek.

Özellikle plaka küçük olduğunda resize işlemi, karakterlerin OCR tarafından işlenmesini kolaylaştırabiliyor.

Grayscale renk bilgisini ortadan kaldırarak karakter ve arka plan arasındaki yoğunluk farkına odaklanmamızı sağlıyor.

Thresholding karakterleri arka plandan ayırmaya yardımcı olabilirken, sharpening karakter kenarlarını belirginleştirebiliyor.

Ancak burada önemli bir sonuçla karşılaştım:

Her preprocessing yöntemi her görüntüde aynı sonucu vermiyor.

Bir görüntüde faydalı olan thresholding başka bir görüntüde karakterleri bozabiliyor.

Bu nedenle tek bir preprocessing sonucuna güvenmek yerine **birden fazla varyant üretmek** daha anlamlı hale geldi.

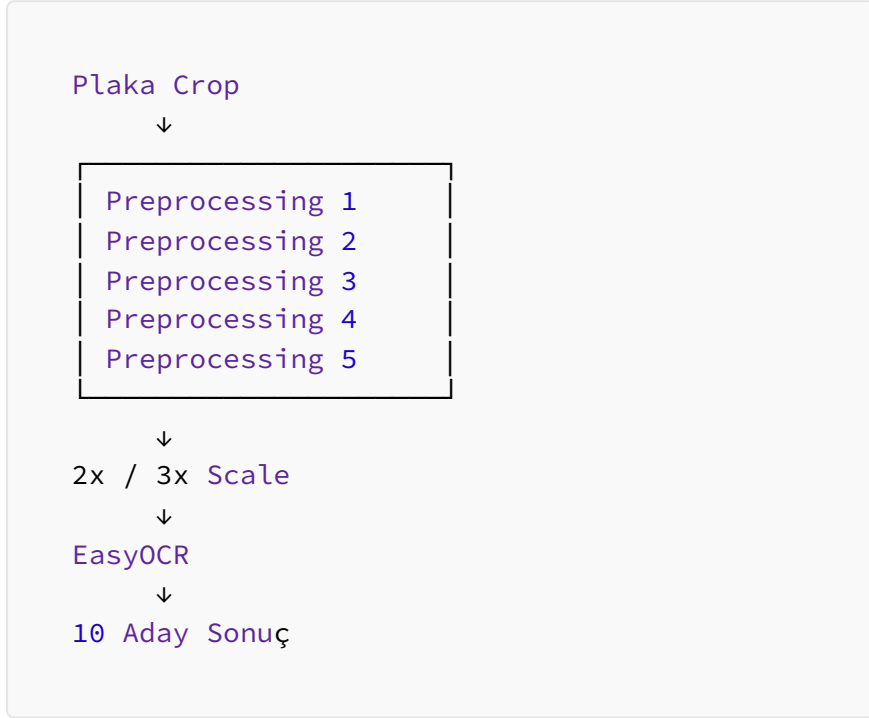
. . .

📌 Proje A: EasyOCR + Format Doğrulama

İlk yaklaşımda OCR tarafında EasyOCR kullandım. Ancak yalnızca tek bir görüntü varyantını EasyOCR'a göndermek yerine 5 farklı preprocessing versiyonu $\times$ 2 farklı ölçek kullandım.

Böylece toplam 10 OCR adayı oluşturdum.

Genel yapı:



Burada her OCR sonucunu doğrudan kabul etmek yerine, elde edilen metinleri plaka formatıyla karşılaştırdım.

Ayrıca karakter karışıklıklarını ele almak için kendi geliştirdiğim

`smart_correct` fonksiyonunu kullandım.

Bu fonksiyon OCR'ın ürettiği olası hatalı karakterleri değerlendirerek alternatifler üretmeye ve daha olası plaka formatına ulaşmaya çalışıyor.

Böylece sistem yalnızca OCR'ın verdiği ilk metne güvenmek yerine:

OCR sonucu + format bilgisi + karakter düzeltme

üçlüsünü birlikte kullanıyor.

. . .

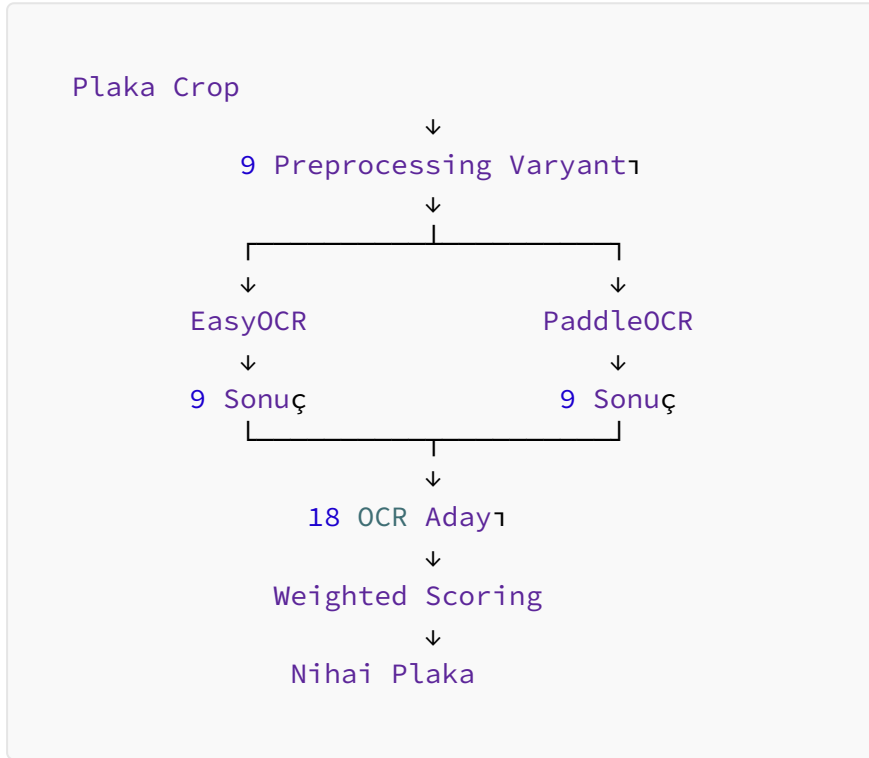


Proje B: EasyOCR + PaddleOCR

İkinci yaklaşımda OCR tarafını daha da genişlettim. Bu kez 9 farklı preprocessing varyantı oluşturdum. Her varyantı hem EasyOCR hem de PaddleOCR'a gönderdim.

Dolayısıyla $9 \times 2 = 18$ OCR adayı elde edildi.

Pipeline şu şekilde:



Burada projenin daha özgün taraflarından biri devreye giriyor:



Ağırlıklı Skor Oylaması

18 adaydan hangisinin doğru olduğuna yalnızca OCR confidence değerine bakarak karar vermedim.

Çünkü yüksek confidence her zaman doğru metin anlamına gelmiyor.

Bunun yerine aday sonuçları birden fazla kriter üzerinden değerlendirdim.

Kullandığım skor:

$$\text{Skor} = (\text{Confort} \times 0.30) + (\text{Confmax} \times 0.20) + (\text{Destekskor} \times 0.25) + (\text{Destekagırlıklı} \times 0.10)$$

Burada beş farklı bilgi birlikte değerlendiriliyor:

- **Ortalama confidence:** Adayın genel güvenilirliği
- **Maksimum confidence:** En güçlü OCR çıktısının güveni
- **Destek skoru:** Aynı metnin kaç farklı OCR sonucunda tekrarlandığı
- **Ağırlıklı destek:** Daha güvenilir sonuçların daha fazla etkili olması
- **Format skoru:** Metnin beklenen plaka formatına ne kadar uyduğu

Bu yaklaşımın temel fikri şu:

Bir OCR sonucu yalnızca yüksek confidence aldığı için doğru kabul edilmemeli; farklı denemelerden ne kadar destek aldığı ve beklenen plaka formatına ne kadar uyduğu da hesaba katılmalı.

Örneğin 18 adayın çoğu benzer bir plaka metni üretirken yalnızca bir aday farklı bir karakter söylüyorsa, tek başına o adayın confidence değeri yüksek olsa bile diğer kriterler sonucu değiştirebilir.

Bu nedenle weighted voting, tek bir OCR çıktısına bağımlılığı azaltmayı amaçlıyor.

. . .



Model Eğitim Sonuçları

Şimdi projenin en önemli kısmına, deneysel sonuçlara gelemim.

İki YOLO11n modeli farklı veri setleri ve farklı eğitim stratejileriyle eğitildi.

Proje A

1.653 görüntülük veri seti kullanıldı.

Model toplamda 5 eğitim/fine-tuning denemesi üzerinden geliştirildi ve toplam eğitim süresi yaklaşık 23 saat oldu.

Proje B

432 görüntülük veri seti kullanıldı.

Model tek eğitim seansı ile toplam 50 epoch eğitildi.

Sonuçlar oldukça dikkat çekiciydi.

Performans Metriği	Proje A — 1.653 Görsel	Proje B — 432 Görsel
Precision	0,990	0,918
Recall	0,988	0,856
mAP@50	0,994	0,909
mAP@50-95	0,869	0,497
Validation Box Loss	0,555	1,486
Validation Class Loss	0,303	0,939

Bu sonuçlarda özellikle **mAP@50-95** değerindeki fark dikkat çekiyor.

Proje A:

0,869

Proje B:

0,497

Bu, yalnızca plakanın bulunup bulunmadığını değil, bounding box'ın gerçek plaka konumuyla ne kadar iyi örtüştüğünü değerlendiren daha katı bir ölçekte Proje A'nın belirgin şekilde daha başarılı olduğunu gösteriyor.

. . .



Precision, Recall ve mAP Bize Ne Söylüyor?

Model sonuçlarını yalnızca sayısal olarak vermek yerine bu metriklerin ne ifade ettiğine de bakmak gerekiyor.

Precision—0,990

Precision, modelin plaka olarak işaretlediği bölgelerin ne kadarının gerçekten plaka olduğunu gösteriyor.

Proje A'daki 0,990 Precision değeri oldukça yüksek.

Başka bir ifadeyle model, plaka olmayan bölgeleri plaka olarak işaretleme konusunda oldukça başarılı.

. . .

Recall—0,988

Recall ise gerçek plakaların ne kadarının model tarafından bulunabildiğini gösteriyor.

Proje A'daki 0,988 Recall değeri, gerçek plakaların büyük bölümünün tespit edilebildiğini gösteriyor.

Bu önemli çünkü bir plaka görüntüde olduğu halde YOLO tarafından bulunamazsa OCR aşamasına hiç ulaşmıyor.

. . .

mAP@50—0,994

Proje A'nın 0,994 mAP@50 değeri, modelin plaka tespitinde oldukça yüksek performans gösterdiğini ortaya koyuyor.

Ancak burada yalnızca mAP@50'ye bakmak yeterli değil.

. . .

mAP@50–95–0,869

mAP@50–95 daha katı bir değerlendirme sunuyor. Bu metrik farklı IoU eşikleri üzerinden bounding box'ın ne kadar doğru konumlandırıldığını değerlendiriyor.

Proje A'nın 0,869. değerine karşı Proje B'nin 0,497 değerinde kalması, daha büyük veri seti ve daha uzun/kademeli eğitim stratejisinin özellikle **bounding box hassasiyeti** üzerinde önemli bir fark oluşturduğunu düşündürüyor.

. . .



Eğitim Süresinin Etkisi

İki proje arasındaki fark yalnızca veri miktarı değildi.

Proje A'da model tek seferde bırakılmadı.

Toplam 5 eğitim denemesi / fine-tuning aşaması uygulandı.

Toplam süre yaklaşık:

23 saat

oldu.

Proje B ise tek eğitim sürecinde:

50 epoch

ile tamamlandı.

Dolayısıyla Proje A'nın daha yüksek performans göstermesinde yalnızca 1.653 > 432 şeklinde basit bir veri miktarı ilişkisi kurmak doğru olmaz.

Burada aynı zamanda:

- veri çeşitliliği,
- veri bölme stratejisi,
- eğitim süresi,

- fine-tuning aşamaları

da birlikte etkili olmuş olabilir.

Bu nedenle deney sonucunu “daha fazla veri kesinlikle daha iyi modeldir” şeklinde yorumlamak yerine:

Daha büyük ve çeşitli veri setiyle birlikte uygulanan daha kapsamlı eğitim stratejisinin, bu deneyde plaka tespit performansını belirgin şekilde iyileştirdiği

şeklinde değerlendirmek daha doğru.

. . .



OCR Deneyleri

YOLO tarafındaki sonuçlar oldukça net olsa da OCR tarafında durum biraz daha farklıydı.

Proje B’de 20 görüntülük toplu OCR testi gerçekleştirdim.

Bu testlerde sistem yaklaşık:

%90 bulma oranına

ulaştı.

Ayrıca vakaların yaklaşık %30’unda PaddleOCR, EasyOCR’a kıyasla **en yüksek skoru alan OCR motoru** oldu.

Bu sonuç önemli çünkü PaddleOCR’ın sisteme eklenmesi her görüntüde EasyOCR’ın yerini almıyor.

Bunun yerine bazı örneklerde **tamamlayıcı bir kaynak** haline geliyor.

Yani deneyin sonucu:

“PaddleOCR her zaman daha iyi.”

şeklinde değil.

Daha doğru yorum:

Bazı görüntülerde EasyOCR'ın, bazı görüntülerde ise PaddleOCR'ın daha güçlü sonuçlar üretmesi, iki OCR motorunun birlikte değerlendirilmesini anlamlı hale getiriyor.

Tam olarak bu nedenle ikinci projede weighted voting mekanizmasını kullandım.

. . .



Neden İki OCR Motorunu Birleştirdim?

İlk bakışta iki OCR motoru kullanmak gereksiz bir karmaşıklık gibi görünebilir. Ancak deneyler sırasında bunun belirli avantajları olduğunu gördüm.

Tek bir OCR motorunda:

Görüntü
↓
OCR
↓
Tek sonuç

elde ediliyor.

Benim ikinci yaklaşımında ise:

Görüntü
↓
Farklı preprocessing
↓
EasyOCR + PaddleOCR
↓
18 farklı aday
↓
Skorlama

↓
Nihai sonuç

elde ediliyor.

Bu yaklaşım hesaplama maliyetini artırıyor. Ancak karşılığında tek bir OCR sonucuna bağlı kalmak yerine birden fazla gözlem üzerinden karar verme imkânı sağlıyor. Bu da özellikle görüntü kalitesinin değişken olduğu durumlarda faydalı olabiliyor.

. . .

Masaüstü Uygulaması: Tkinter ile Sistemi Kullanılabilir Hale Getirmek

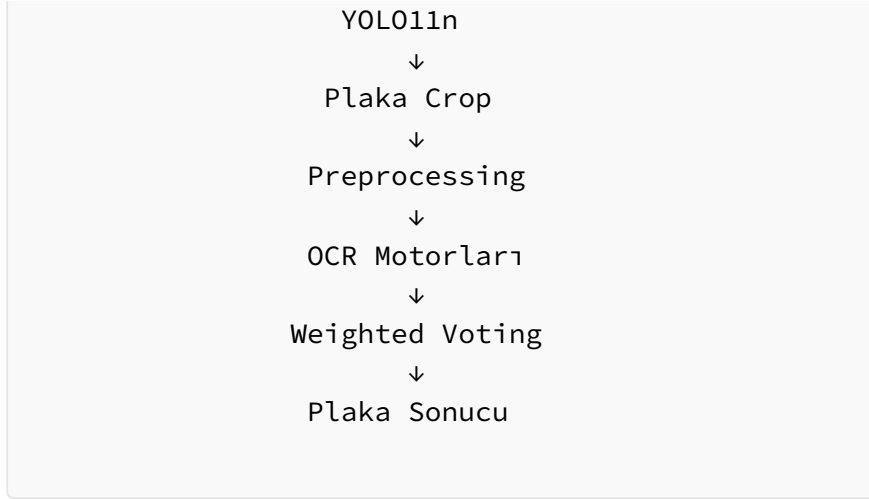
Modelin yalnızca Python kodları üzerinden çalışmasını istemedim. Bu nedenle geliştirdiğim sistemi **Tkinter** kullanarak masaüstü uygulamasına dönüştürdüm. Buradaki amacım kullanıcıdan teknik bilgi gerektirmeyen basit bir akış oluşturmaktı.

Kullanıcı:

1. Görüntüyü seçiyor.
2. Analizi başlatıyor.
3. YOLO11n plakayı tespit ediyor.
4. Plaka crop ediliyor.
5. OCR çalıştırılıyor.
6. Sonuç arayüzde gösteriliyor.

Temel yapı:

Kullanıcı
↓
Görüntü Seç
↓
Tkinter
↓



Proje A Arayüzü

İlk uygulamada Tkinter ile daha sade bir arayüz oluşturdum.

Kullanıcı tek bir görüntü veya klasör seçerek analiz başlatabiliyor.

Tespit edilen plaka bölgesi görüntü üzerinde gösteriliyor ve OCR sonucu kullanıcıya sunuluyor.

Proje B Arayüzü

İkinci uygulamada ise daha kapsamlı bir yapı oluşturdum.

Tkinter tabanlı arayüzde:

- plaka sonucu,
- format durumu,
- kalite seviyesi,
- analiz bilgileri

gibi çıktılar gösteriliyor.

Ayrıca `batch_gui.py` üzerinden toplu test sonuçlarını inceleyebileceğim bir yapı oluşturdum.

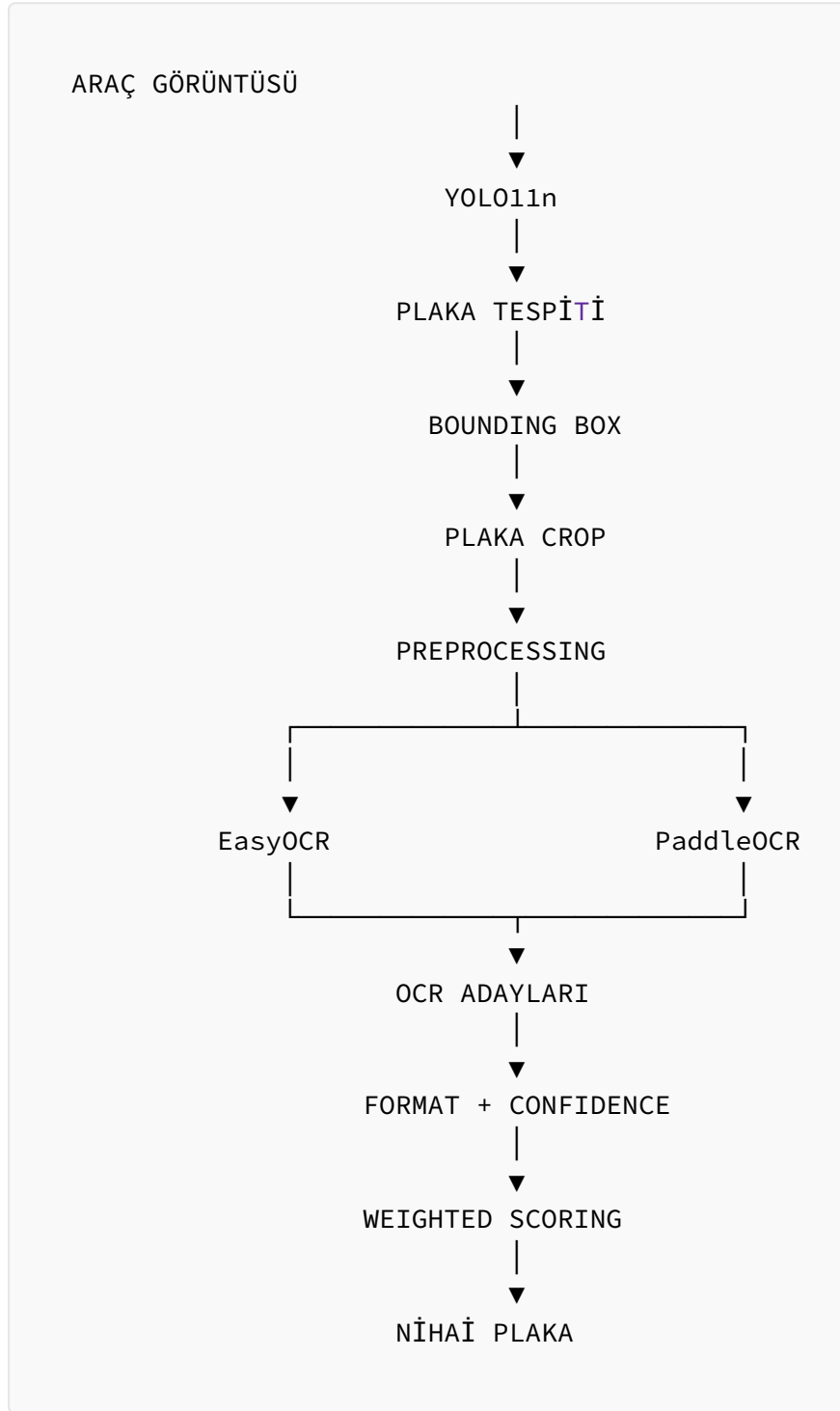
Toplu test sonuçlarının tablo, filtreleme ve grafiklerle incelenebilmesi sayesinde yalnızca tek tek görüntülere bakmak yerine deney sonuçlarını daha sistematik şekilde değerlendirebildim.

. . .



Uçtan Uca Sistem Nasıl Çalışıyor?

Projeyi bütün olarak düşündüğümüzde ortaya çıkan yapı şu:



Bu yapı sayesinde proje yalnızca bir **nesne tespit modeli** olmaktan çıkıp uçtan uca bir plaka tanıma sistemine dönüşüyor.

. . .



Deneylerden Elde Ettiğim En Önemli Çıkarımlar

Bu proje boyunca benim için en önemli sonuçlardan biri, plaka tanımanın tek bir algoritmayla çözülebilecek bir problem olmadığını görmek oldu.

1. Plaka tespiti ve OCR farklı problemlerdir

YOLO11n'in:

$mAP@50 = 0,994$

gibi yüksek bir tespit performansına ulaşması, karakterlerin de aynı oranda doğru okunacağı anlamına gelmiyor.

Çünkü iki model tamamen farklı görevler yapıyor.

YOLO:

Plakanın nerede olduğunu buluyor.

OCR:

O bölgede hangi karakterlerin olduğunu anlamaya çalışıyor.

. . .

2. Veri miktarı ve eğitim stratejisi önemli

1.653 görüntülük Proje A ile 432 görüntülük Proje B arasında belirgin bir performans farkı oluştu.

Özellikle:

$mAP@50-95$

değerindeki:

$0,869 \rightarrow 0,497$

farkı dikkat çekici.

Ancak bu farkı yalnızca veri miktarına bağlamak doğru değil.

Proje A'daki daha kapsamlı eğitim ve fine-tuning süreci de sonuç üzerinde etkili.

. . .

3. Daha fazla OCR motoru otomatik olarak daha iyi değildir

PaddleOCR'ın eklenmesi bazı örneklerde avantaj sağladı.

Ancak her görüntüde en iyi sonucu PaddleOCR vermedi.

20 görüntülük testte vakaların yaklaşık %30'unda PaddleOCR en yüksek skoru aldı.

Bu nedenle ikinci OCR motorunu sisteme eklememin asıl nedeni tek başına “daha iyi model” bulmak değil, **farklı OCR sonuçlarından yararlanarak daha güvenilir bir karar mekanizması oluşturmak** oldu.

. . .

4. Preprocessing düşündüğümde daha önemli çıktı

OCR başarısında yalnızca kullanılan motor değil, OCR'a verilen görüntünün niteliği de büyük önem taşıyor.

Resize, grayscale, thresholding, sharpening ve farklı preprocessing varyantlarını kullanmamın temel nedeni buydu.

Özellikle küçük ve düşük kaliteli plakalar üzerinde OCR'ın doğru karakterleri ayırt edebilmesi için görüntünün hazırlanması kritik hale geliyor.

. . .

5. Confidence tek başına yeterli değil

İkinci yaklaşımda benim için en önemli deneylerden biri buydu.

OCR motorunun yüksek confidence vermesi, sonucun kesinlikle doğru olduğu anlamına gelmiyor.

Bu nedenle:

Confidence

+

Tekrar / Destek

+

Format Uyumu

+

Farklı OCR Sonuçları

gibi birden fazla kriteri birlikte değerlendirmek daha anlamlı oldu.

Weighted scoring yaklaşımının temel çıkış noktası da buydu.

. . .



Sonuç

Bu projeye başlarken hedefim oldukça basitti:

Bir araç görüntüsünden plakayı bulmak ve okumak.

Ancak uygulama geliştikçe bunun tek bir modelin çözebileceği kadar basit bir problem olmadığını gördüm.

Ortaya çıkan sistem;

Veri Toplama

↓

Veri Etiketleme

↓

YOLO11n Eğitimi



Plaka Tespiti



Crop



Görüntü Ön İşleme



EasyOCR / PaddleOCR



Format Doğrulama



Weighted Scoring



Nihai Plaka



Tkinter Masaüstü Uygulaması

şeklinde uçtan uca çalışan bir pipeline'a dönüştü.

İki farklı veri seti ve iki farklı OCR yaklaşımı üzerinde yaptığım deneyler bana özellikle şu noktayı gösterdi:

Başarılı bir bilgisayarlı görü sistemi, yalnızca iyi bir model seçmekten ibaret değil.

Kaliteli ve çeşitli veri, doğru model, uygun preprocessing, OCR stratejisi ve doğru karar mekanizması birlikte düşünülmeli.

Bu projede benim için en değerli kazanım da yalnızca YOLO11n veya OCR kullanmayı öğrenmek olmadı.

Asıl kazanç, bir problemi: **veriden modele, modelden görüntü işlemeye, görüntü işlemekten kullanıcı arayüzüne kadar uçtan uca** ele alabilmektir.

Çünkü gerçek bir makine öğrenmesi projesinde amaç yalnızca:

“**Modelim çalışıyor.**”

demek değil.

Asıl amaç:

“**Çözdüğüm problem için çalışan, ölçülebilen, test edilebilen ve kullanılabilen bir sistem geliştirdim.**”

diyebilmek. Ve bu proje benim için tam olarak bunu deneyimlediğim çalışmalardan biri oldu.