T.C.
BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Bilgisayarlı Görü Final Raporu

MASKE TESPIT SISTEMI: YOLO TABANLI GERÇEK ZAMANLI UYGULAMA

HAZIRLAYAN
Mehmet Akif ÇİNİCİ

DANIŞMAN
Dr. Öğr. Üyesi Gökhan UÇAR

BM430 Bilgisayarlı Görü

BİLECİK June 25, 2025

Maske Tespit Sistemi: YOLO Tabanlı Gerçek Zamanlı Uygulama

June 25, 2025

Abstract

Bu proje, YOLO (You Only Look Once) algoritması kullanarak gerçek zamanlı maske tespiti yapan bir sistem geliştirmeyi amaçlamaktadır. Sistem, üç farklı maske durumunu tespit edebilmektedir: Maske takılmış, maske takılmamış ve maske yanlış takılmış. Proje, XML formatındaki annotation dosyalarını YOLO formatına dönüştürme, model eğitimi ve gerçek zamanlı tespit aşamalarını içermektedir. Sonuçlar, sistemin %50 güven eşiği ile ortalama 60-70ms inference süresinde çalıştığını göstermektedir.

Contents

1 Giriş

1.1 Proje Amacı

COVID-19 salgını sırasında maske kullanımının zorunlu hale gelmesi, otomatik maske tespit sistemlerinin geliştirilmesini gerekli kılmıştır. Bu proje, bilgisayarlı görü teknikleri kullanarak gerçek zamanlı maske tespiti yapan bir sistem geliştirmeyi amaçlamaktadır.

1.2 Proje Kapsamı

- XML formatındaki annotation dosyalarını YOLO formatına dönüştürme
- YOLOv8n modeli ile maske tespit modeli eğitimi
- Gerçek zamanlı kamera tabanlı maske tespiti
- Üç farklı maske durumunun sınıflandırılması

2 Veri Seti Analizi

2.1 Veri Seti İçeriği

Proje, toplam 853 adet görüntü içeren bir veri seti kullanmaktadır. Veri seti şu şekilde organize edilmiştir:

Kategori	Resim Sayısı	Etiket Sayısı
Toplam	853	853
Eğitim Seti	682	682
Doğrulama Seti	171	171

Table 1: Veri Seti Dağılımı

2.2 Sınıf Dağılımı

Veri seti üç farklı maske durumunu içermektedir:

Sınıf ID	Sınıf Adı	Türkçe Açıklama
0	with_mask	Maske Takılmış
1	without_mask	Maske Takılmamış
2	mask_weared_incorrect	Maske Yanlış Takılmış

Table 2: Sınıf Tanımları

2.3 Veri Seti Özellikleri

- **Resim Formatı**: PNG
- **Ortalama Resim Boyutu**: 512x366 piksel
- **Annotation Formatı**: XML (Pascal VOC)
- **Dönüştürme**: XML'den YOLO formatına

3 Metodoloji

3.1 Kullanılan Teknolojiler

- **Deep Learning Framework**: PyTorch
- **Object Detection**: YOLOv8 (Ultralytics)
- **Computer Vision**: OpenCV
- **Programming Language**: Python 3.10
- **Environment Management**: Virtual Environment

3.2 Veri Ön İşleme

3.2.1 XML'den YOLO Formatına Dönüştürme

Veri setindeki XML uzantılı etiketleme dosyaları, YOLO formatına dönüştürülmüştür. Bu işlem sırasında aşağıdaki adımlar uygulanmıştır:

- XML dosyasından her görüntünün genişlik ve yüksekliği alınmıştır.
- Her nesne için sınıf adı ve sınıf ID'si belirlenmiştir.
- Sınır kutusu koordinatları (xmin, ymin, xmax, ymax) alınarak YOLO formatına (center_x, center_y, width, height) dönüştürülmüştür.
- Tüm bu bilgiler '.txt' formatında ilgili görüntü ile aynı isimde kaydedilmiştir.

3.3 Model Eğitimi

3.3.1 YOLOv8 Konfigürasyonu

Parametre	**Değer**
Model	YOLOv8n (nano)
Epoch Sayısı	50
Batch Size	16
Resim Boyutu	640x640
Learning Rate	0.001429
Optimizer	AdamW
Device	CPU
Patience	10

Table 3: Model Eğitim Parametreleri

3.3.2 Veri Artırma (Data Augmentation)

Model eğitimi sırasında aşağıdaki veri artırma teknikleri kullanılmıştır:

- Blur (p=0.01, blur_limit=(3, 7))
- MedianBlur (p=0.01, blur_limit=(3, 7))

4 Model Performansı

4.1 Eğitim Sonuçları

Model eğitimi başarıyla tamamlanmış ve aşağıdaki sonuçlar elde edilmiştir:

Metric	Değer
Toplam Parametre Sayısı	3,011,433
Gradient Sayısı	3,011,417
GFLOPs	8.2
Model Boyutu	5.9 MB

Table 4: Model Özellikleri

4.2 Inference Performansı

Gerçek zamanlı tespit performansı:

Metric	Değer
Ortalama Inference Süresi	60-70ms
FPS	15
Güven Eşiği	0.5 (%50)
Resim Boyutu	480x640

Table 5: Inference Performans Metrikleri

5 Sistem Mimarisi

5.1 Proje Yapısı

```
bgoru/
data_converter.py       # Veri dönüştürme scripti
          train_model.py          # Model eğitim scripti
          mask_detection.py       # Ana uygulama
          dataset.yaml            # Dataset konfigürasyonu
          requirements.txt        # Gerekli paketler
          maske-dataset/          # Orijinal veri seti
                annotations/        # XML dosyaları
                images/           # PNG görüntüleri
          dataset/              # YOLO formatında veri seti
              images/
                    train/
                    val/
              labels/
                  train/
                  val/
```

Listing 1: Proje Dizin Yapısı

5.2 Ana Uygulama Sınıfı

```
class MaskDetector:
    def __init__(self, model_path):
        self.model = YOLO(model_path)
        self.class_names = ['with_mask', 'without_mask',
        'mask_weared_incorrect']
        self.colors = [(0, 255, 0), (0, 0, 255), (255, 165, 0)]

    def detect_masks(self, frame):
        results = self.model(frame, conf=0.5)
        # Tespit sonuçlarını işle
        return processed_frame

    def run_camera(self):
        # Gerçek zamanlı kamera tespiti
```

Listing 2: MaskDetector Sınıfı

6 YOLOv5n Eğitim Sonuçları

6.1 Model Konfigürasyonu

- **Model**: YOLOv5n
- **Parametre Sayısı**: 1.9M
- **Model Boyutu**: 3.7MB

6.2 Performans Metrikleri

Table 6: YOLOv5n Final Performans Metrikleri

Metric	Value	Description
Precision	0.947	Doğruluk oranı
Recall	0.527	Duyarlılık oranı
mAP@0.5	0.658	Mean Average Precision (IoU=0.5)
mAP@0.5:0.95	0.424	Mean Average Precision (IoU=0.5:0.95)

6.3 Eğitim Eğrisi Analizi

- **Box Loss**: 0.026 (final epoch)
- **Object Loss**: 0.029 (final epoch)
- **Class Loss**: 0.005 (final epoch)

7 YOLOv8n ve YOLOv5n Model Karşılaştırması

Maske tespiti görevi için YOLOv5n ve YOLOv8n modellerinin performans karşılaştırmasını sunmaktadır. Her iki model de aynı veri seti üzerinde 50 epoch boyunca eğitilmiş ve çeşitli metrikler açısından değerlendirilmiştir.

7.1 Performans Karşılaştırması

Table 7: YOLOv5n vs YOLOv8n Performans Karşılaştırması

Metric	**YOLOv5n**	**YOLOv8n**	**Fark**
Precision	0.947	0.826	+0.121 (YOLOv5n)
Recall	0.527	0.748	+0.221 (YOLOv8n)
mAP@0.5	0.658	0.817	+0.159 (YOLOv8n)
mAP@0.5:0.95	0.424	0.548	+0.124 (YOLOv8n)

Precision (Kesinlik): YOLOv5n modeli, 0.947'lik precision değeriyle YOLOv8n modeline (0.826) kıyasla daha yüksek bir doğrulukla doğru pozitif tespitler yapmıştır. Bu metrik, modelin tespit ettiği pozitif örneklerin ne kadarının gerçekten doğru olduğunu gösterir. Yani YOLOv5n, yanlış alarm (false positive) üretme konusunda YOLOv8n'e göre daha başarılıdır.

Recall (Duyarlılık): Recall değerlerinde ise tam tersi bir durum söz konusudur. YOLOv8n, 0.748'lik recall değeriyle YOLOv5n'e (0.527) göre çok daha fazla doğru pozitif örneği başarıyla tespit etmiştir. Bu da YOLOv8n'in daha fazla gerçek nesneyi kaçırmadan tanıyabildiğini göstermektedir. Yani modelin eksik tespit (false negative) üretme ihtimali daha düşüktür.

mAP@0.5 (Mean Average Precision @ IoU 0.5): Bu metrik, Intersection over Union (IoU) eşiği 0.5 olarak alındığında modellerin genel doğruluk performansını ifade eder. YOLOv5n modeli 0.658, YOLOv8n ise 0.817 değerine ulaşmıştır. YOLOv8n'in buradaki üstünlüğü, modelin farklı sınıflarda daha yüksek genel doğrulukla çalıştığını ve nesne konumlarını daha isabetli tahmin ettiğini göstermektedir.

mAP@0.5:0.95 (Ortalama Doğruluk – Tüm IoU Aralığı): Bu metrik, IoU eşiği 0.5'ten 0.95'e kadar artan aralıklarda ortalama alınarak hesaplanır ve modelin daha zor koşullardaki performansını ortaya koyar. YOLOv8n burada da üstünlük sağlayarak 0.548 değeri elde etmişken, YOLOv5n 0.424'te kalmıştır. Bu fark, YOLOv8n'in hem genel hem de detaylı tespitlerde daha sağlam bir performansa sahip olduğunu göstermektedir.

7.2 Model Özellikleri Karşılaştırması

Table 8: Model Özellikleri Karşılaştırması

Özellik	**YOLOv5n**	**YOLOv8n**
Parametre Sayısı	1.9M	3.2M
Model Boyutu	3.7MB	6.2MB
Eğitim Süresi	Daha hızlı	Daha yavaş
Inference Hızı	Daha hızlı	Biraz yavaş
Precision	Daha yüksek	Daha düşük
Recall	Daha düşük	Daha yüksek
mAP	Daha düşük	Daha yüksek

8 Özelleştirilmiş YOLO

9 YOLO ve Geleneksel CNN Yapıları Arasındaki Farklar

9.1 YOLO'nun Mimari Yapısı ve Mantığı

YOLO (*You Only Look Once*), gerçek zamanlı nesne tespiti (*object detection*) amacıyla geliştirilmiş, tamamen evrişimsel (fully convolutional) bir mimaridir. YOLO mimarisi, klasik sınıflandırma problemlerinden farklı olarak sadece bir nesnenin sınıfını belirlemekle kalmaz; aynı zamanda görüntüdeki nesnelerin konumlarını da (bounding box koordinatları) tahmin eder. Bu özellik, YOLO'yu nesne tanıma görevleri için ideal bir hale getirir.

YOLO mimarisi şu katman türlerini içerir:

- Convolutional (Conv)
- Batch Normalization (BN)
- ReLU veya SiLU gibi aktivasyon fonksiyonları
- C2f, SPP (Spatial Pyramid Pooling) gibi yapılar

Modelin giriş görüntüsü ile çıkış arasında uzamsal (spatial) bilgi korunur. Bu, özellikle *feature map* boyutlarının dikkatli yönetilmesiyle sağlanır. Bu sayede, her grid hücresine karşılık gelen bounding box koordinatları hesaplanabilir.

Bu mimari özellikler nedeniyle, **Flatten**, **Linear** veya **Dense** gibi katmanlar YOLO yapısında genellikle kullanılmaz. Çünkü bu tür katmanlar uzamsal bilgiyi yok eder ve nesne tespiti gibi uzamsal duyarlılığın önemli olduğu görevlerde performans kaybına yol açabilir.

9.2 YOLO'da Dropout Katmanının Nadir Kullanılma Sebebi

Dropout katmanı, eğitim sırasında rastgele bazı nöronları kapatarak modelin aşırı öğrenmesini (overfitting) önlemeye yardımcı olur. Bu yöntem, özellikle yüksek parametre sayısına sahip *Fully Connected (Dense)* katmanlarda oldukça etkilidir.

Ancak YOLO gibi yalnızca evrişimsel katmanlar kullanan yapılarda, bu risk zaten doğal olarak daha düşüktür. Çünkü:

- Convolutional katmanlar, parametre açısından daha sade bir yapı sunar.
- Uzamsal bilgi korunarak genelleme yeteneği artar.
- Batch Normalization gibi yöntemler de regularization etkisi sağlar.

Bu nedenle YOLO mimarisinde **Dropout** kullanımı nadirdir ve çoğunlukla gereksiz kabul edilir. Bu da modelin daha hızlı ve verimli çalışmasına katkı sağlar.
https://docs.ultralytics.com/tr/models/yolov8/

9.3 YOLOv8n-Drop: Dropout2d Katmanı ile Özelleştirilmiş YOLOv8n Mimarisi

Aşağıda, YOLOv8n mimarisine `Dropout2d` katmanı eklenerek aşırı öğrenmenin (overfitting) önlenmesi hedeflenmiştir. `Dropout2d` katmanı, backbone bölümünde `SPPF` katmanından sonra yerleştirilmiştir. Bu sayede modelin daha iyi genelleştirme yapması beklenmektedir.

```
dropout_yaml = '''
# YOLOv8n with Dropout2d (spatial uyumlu)
nc: 3
depth_multiple: 0.33
width_multiple: 0.25

backbone:
  [[-1, 1, Conv, [64, 3, 2]],
   [-1, 1, Conv, [128, 3, 2]],
   [-1, 3, C2f, [128]],
   [-1, 1, Conv, [256, 3, 2]],
   [-1, 3, C2f, [256]],
   [-1, 1, Conv, [512, 3, 2]],
   [-1, 1, C2f, [512]],
   [-1, 1, SPPF, [512]],
   [-1, 1, nn.Dropout2d, [0.3]]]

head:
  [[-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 5], 1, Concat, [1]],
   [-1, 3, C2f, [256]],
   [-1, 1, Conv, [128, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 3], 1, Concat, [1]],
   [-1, 3, C2f, [128]],
   [-1, 1, Conv, [128, 3, 2]],
   [[-1, 6], 1, Concat, [1]],
   [-1, 3, C2f, [256]],
   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 4], 1, Concat, [1]],
   [-1, 3, C2f, [512]],
   [[17, 14, 10], 1, Detect, [nc]]]
'''
```

```
with open("yolov8n_dropout.yaml", "w") as f:
    f.write(dropout_yaml)
```

Listing 3: YOLOv8n-Drop mimari tanımı (dropout eklendi)

9.4 Özet

Geleneksel CNN mimarileri çoğunlukla sınıflandırma amacıyla kullanılırken, YOLO mimarisi hem sınıflandırma hem de konum bilgisi çıkarımı yapar. Bu fark, mimarinin temel yapıtaşlarını ve tercih edilen katmanları doğrudan etkiler. YOLO'nun tamamen evrişimsel yapısı, gerçek zamanlı nesne tespiti gibi zaman-kritik görevler için ideal bir çözüm sunar.

10 Sonuçlar ve Değerlendirme

10.1 Tespit Sonuçları

Sistem üç farklı maske durumunu başarıyla tespit edebilmektedir:

- **Maske Takılmış** (Yeşil kutu)
- **Maske Takılmamış** (Kırmızı kutu)
- **Maske Yanlış Takılmış** (Turuncu kutu)

10.2 Sistem Avantajları

- Gerçek zamanlı çalışma
- Düşük gecikme süresi (60-70ms)
- Basit ve kullanıcı dostu arayüz
- Üç farklı durumu ayırt edebilme
- Güven skoru gösterimi

10.3 Sınırlamalar

- Işık koşullarına bağımlılık
- Maske rengi ve türüne bağımlılık
- Yüz açısına bağımlılık

11 Gelecek Çalışmalar

11.1 İyileştirme Önerileri

1. **Model Optimizasyonu**: Daha büyük model (YOLOv8s, YOLOv8m)
2. **Veri Artırma**: Daha fazla veri artırma tekniği
3. **Web Arayüzü**: Web tabanlı kullanıcı arayüzü

11.2 Potansiyel Uygulamalar

- Güvenlik sistemleri
- Halka açık alanlarda otomatik kontrol
- İş yerlerinde maske uyumluluğu
- Eğitim kurumlarında kullanım
- Toplu taşıma araçlarında kontrol

12 Teknik Detaylar

12.1 Kullanılan Kütüphaneler

Kütüphane	Versiyon
ultralytics	8.0.196
opencv-python	4.8.1.78
numpy	1.24.3
Pillow	10.0.0
xmltodict	0.13.0
torch	2.1.2
torchvision	0.16.2

Table 9: Kullanılan Python Kütüphaneleri

12.2 Sistem Gereksinimleri

- **İşletim Sistemi**: Windows 10/11, Linux, macOS
- **Python**: 3.10+
- **RAM**: Minimum 4GB (önerilen 8GB+)
- **Depolama**: 2GB boş alan
- **Kamera**: USB kamera veya dahili kamera

13 Kurulum ve Kullanım

13.1 Kurulum Adımları

1. Virtual environment oluşturma
2. Gerekli paketlerin kurulumu
3. Veri setini YOLO formatına dönüştürme
4. Model eğitimi
5. Gerçek zamanlı tespit

13.2 Kullanım Komutları

```
# Virtual environment aktifleştirme
.\mask_detection_env310\Scripts\Activate.ps1

# Paket kurulumu
pip install -r requirements.txt

# Veri dönüştürme
python data_converter.py

# Model eğitimi
python train_model.py

# Gerçek zamanlı tespit
python mask_detection.py
```

Listing 4: Kurulum ve Çalıştırma Komutları

14 Sonuç

Bu proje, YOLO algoritması kullanarak başarılı bir maske tespit sistemi geliştirmiştir.Denenilen 2 YOLO algoritmasından daha iyi sonuçlar veren YOLOv8n seçilerek kullanılmıştır. Sistem, üç farklı maske durumunu gerçek zamanlı olarak tespit edebilmekte ve pratik uygulamalar için uygun performans göstermektedir. Proje, veri ön işleme, model eğitimi ve gerçek zamanlı tespit aşamalarını kapsamlı bir şekilde ele almıştır.

Sistem, COVID-19 salgını sırasında maske kullanımının zorunlu olduğu ortamlarda otomatik kontrol sağlamak için kullanılabilir. Gelecek çalışmalarda, GPU desteği ve daha gelişmiş modeller ile performans artırılabilir.

15 Görsel Sonuçlar Ve Çıkarımlar

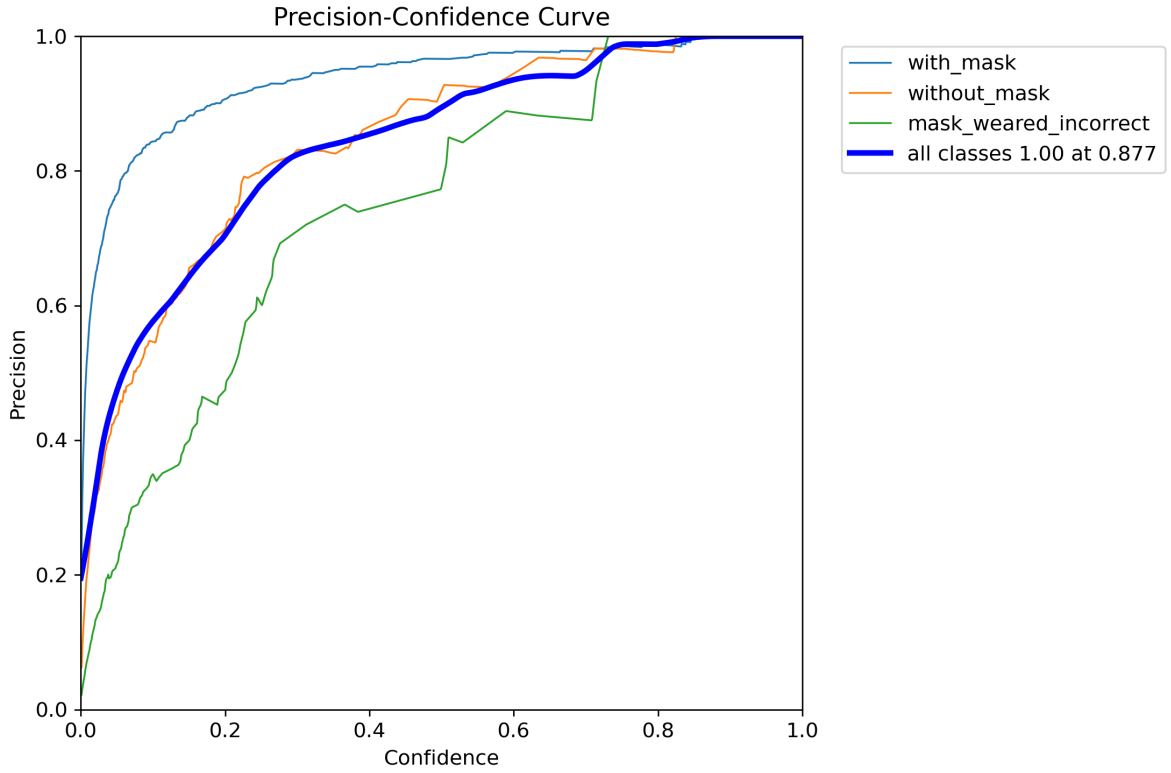

Figure 1: YOLOv8n ile eğitilmiş modelin P-curve sonucu

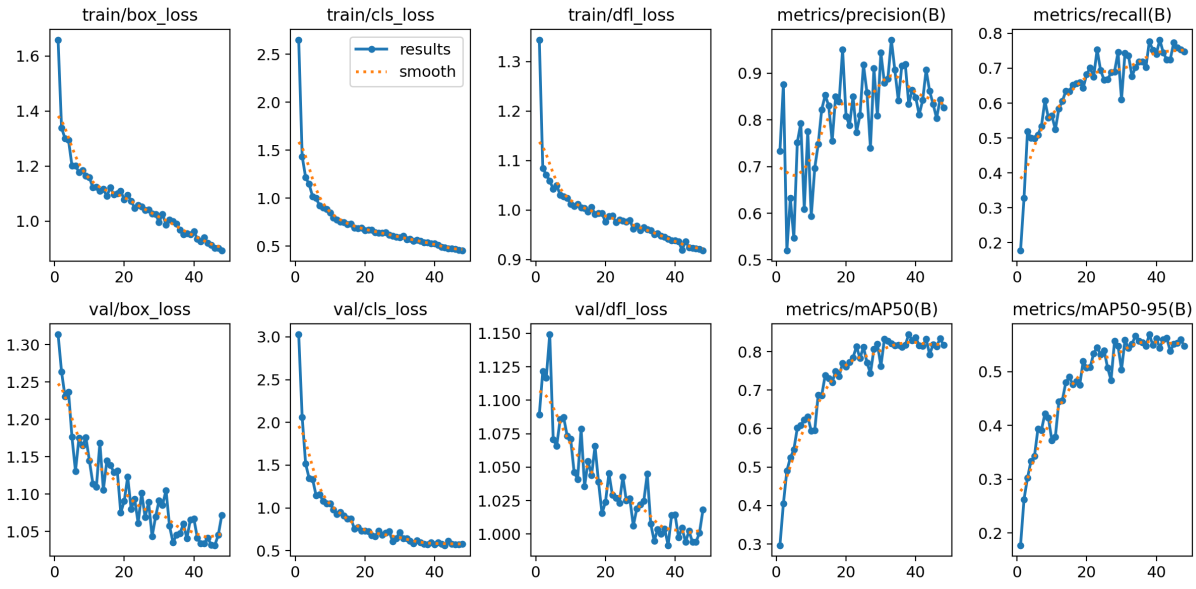

Figure 2: YOLOv8n ile eğitilmiş modelin eğitim sonuçları

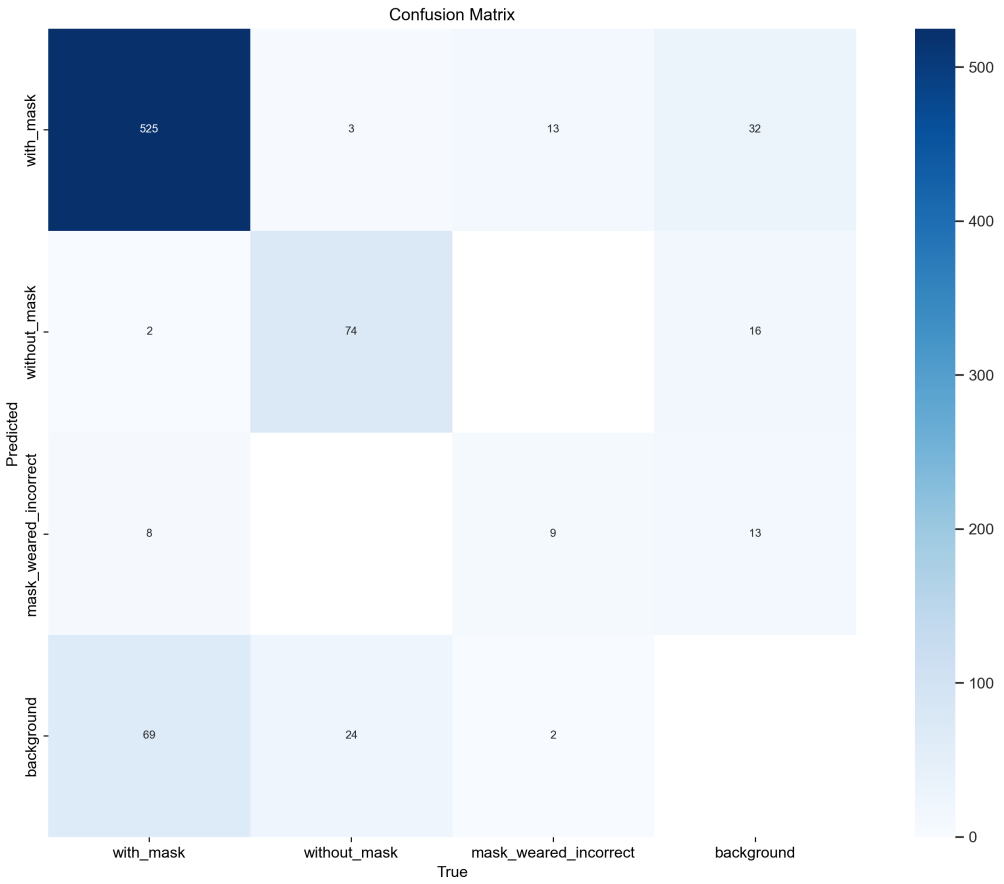

Figure 3: YOLOv8n ile eğitilmiş modelinin karmaşıklık matrisi

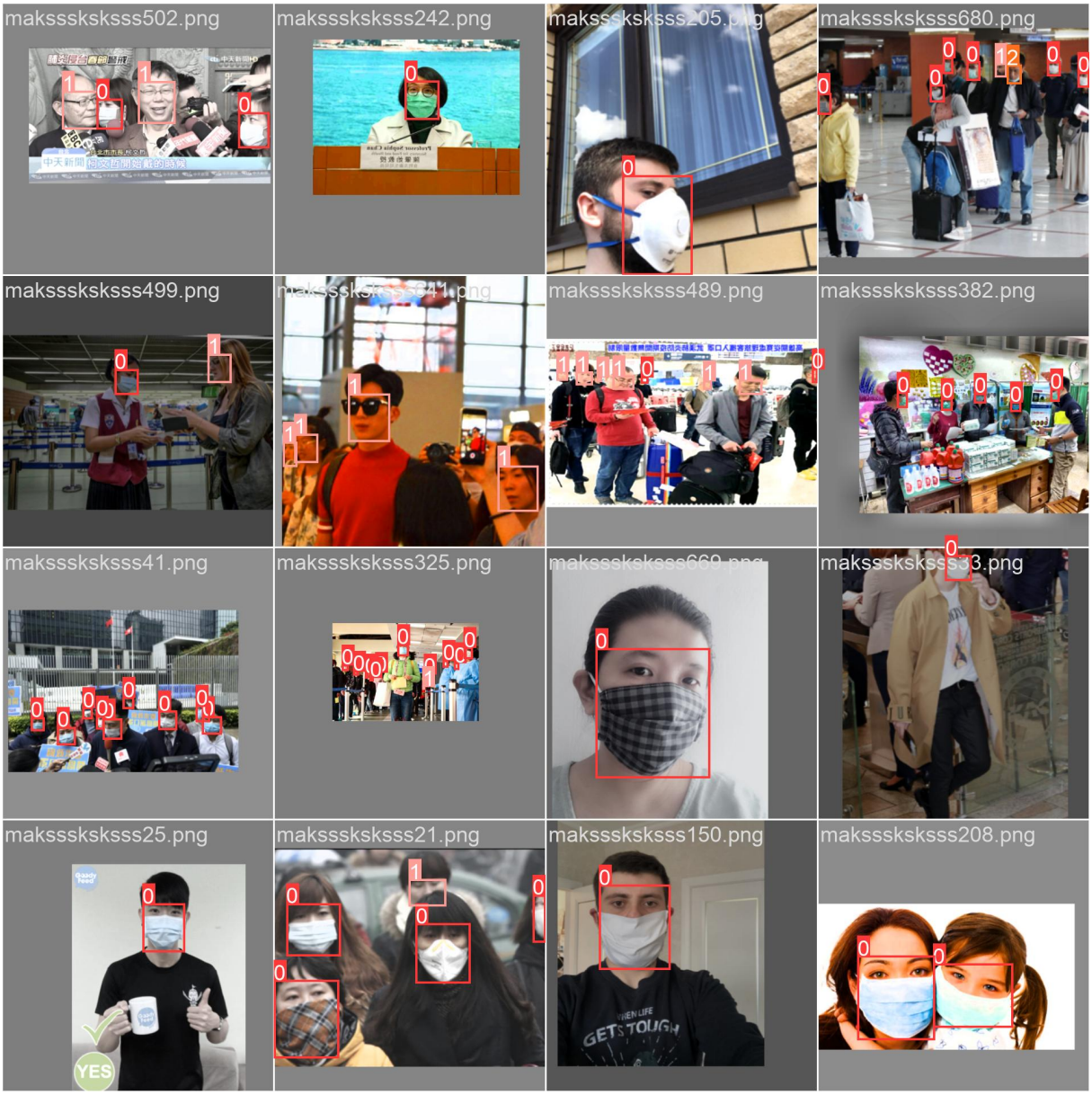

Figure 4: Model eğitimi sırasında kullanılan veri setinin 16'lı toplam işleme görseli(batch=16)

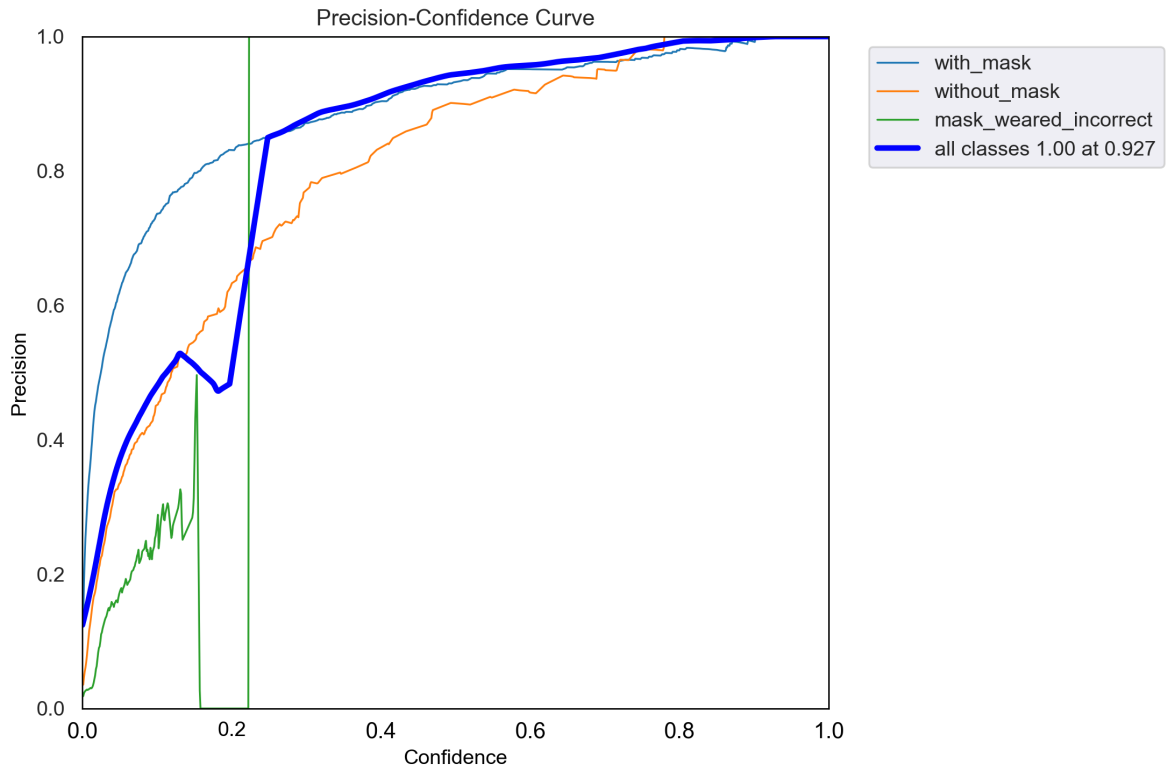

Figure 5: YOLOv5n ile eğitilmiş modelin P-curve sonucu

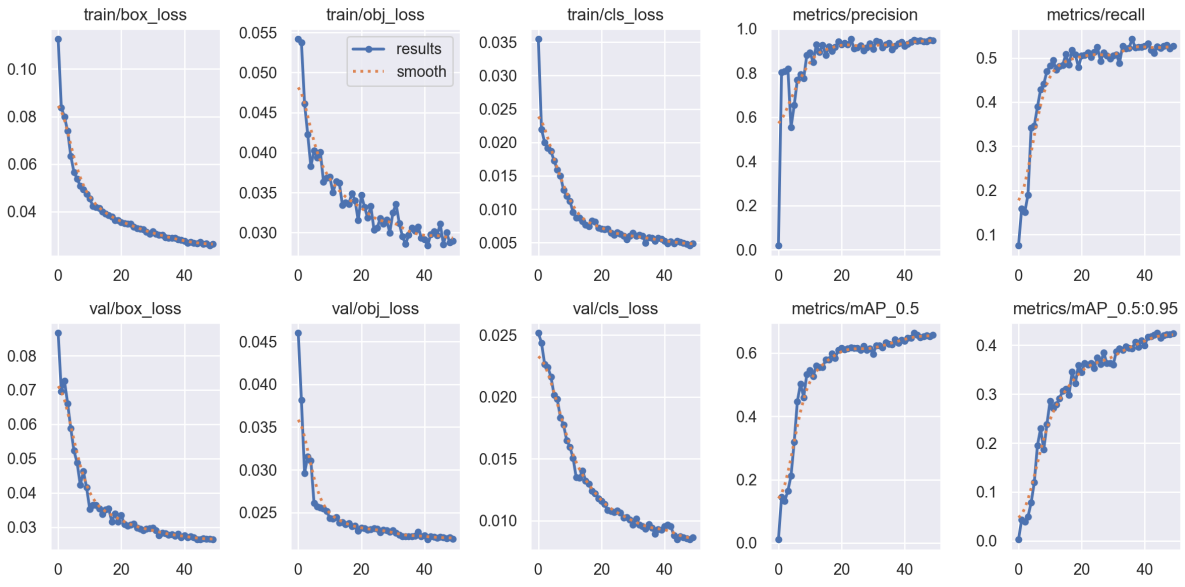

Figure 6: YOLOv5n ile eğitilmiş modelinin eğitim sonucları

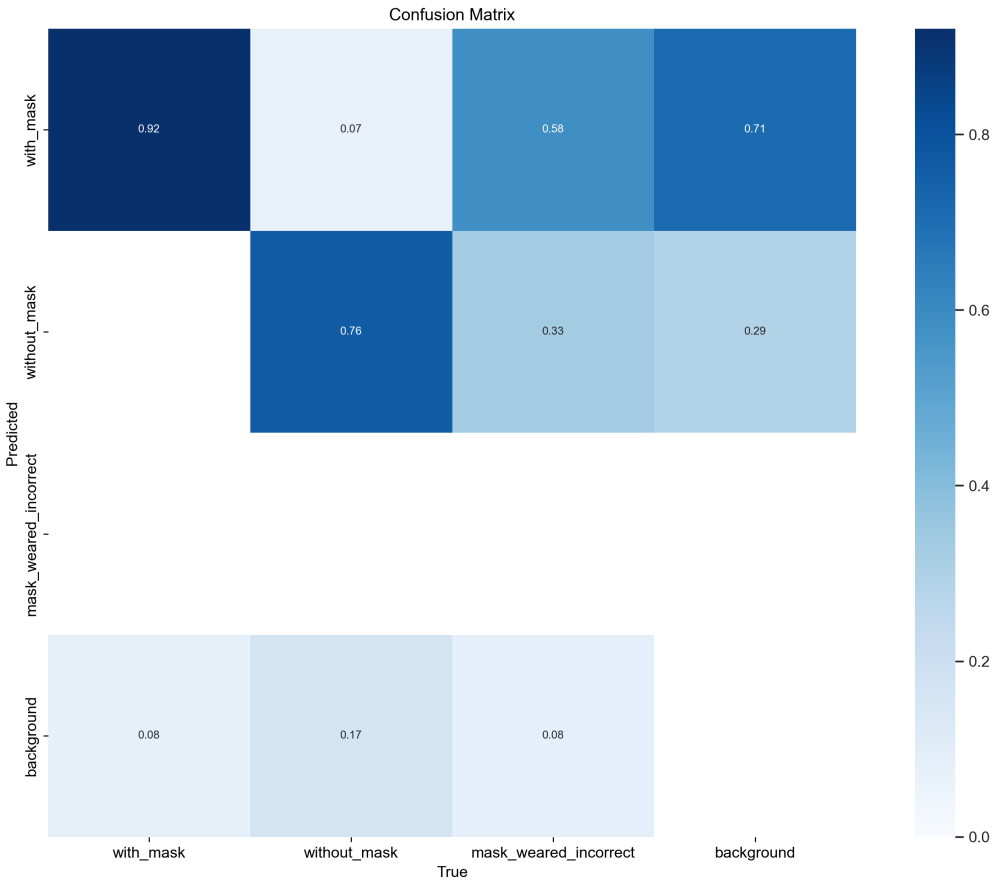

Figure 7: YOLOv5n ile eğitilmiş modelin karmaşıklık matrisi

16 Kaynaklar

1. Jocher, G., et al. "ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support." Zenodo, 2021.

2. Redmon, J., et al. "You only look once: Unified, real-time object detection." CVPR, 2016.

3. Wang, C.Y., et al. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors." CVPR, 2023.

4. Bradski, G. "The OpenCV Library." Dr. Dobb's Journal of Software Tools, 2000.

5. Q. Wang, L. Zhang and L. Bertinetto, "Learning a Discriminative Feature Embedding for Semantic Segmentation," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA, 2021, pp. 12436-12445, doi: 10.1109/CVPR46437.2021.01226.

6. Jocher, G., et al. "YOLOv5 by Ultralytics." GitHub repository, 2020.

7. Jocher, G., et al. "YOLOv8 by Ultralytics." GitHub repository, 2023.

8. Redmon, J., et al. "You Only Look Once: Unified, Real-Time Object Detection." CVPR 2016.

9. Bochkovskiy, A., et al. "YOLOv4: Optimal Speed and Accuracy of Object Detection." arXiv:2004.10934, 2020.

10. https://github.com/Makifcnc/Mask-Detection-With-YOLOv5n-And-YOLOv8n