08.01.2025

ŞELALE MODELİ İLE KUMAŞ DELİK TESPİTİ MOBİL UYGULAMASI RAPORU

MEHMET ALTINKILIÇ

İÇİNDEKİLER

Giriş

Bu proje, derin öğrenme algoritmalarıyla eğitilen bir yapay zeka modelini mobil uygulamaya entegre etmeyi hedeflemektedir. **Sistem**; kumaş üzerindeki delikleri tespit edebilen bir model geliştirmek, bu modeli mobil uygulamada kullanmak üzere entegre etmek ve sistemin bakımını sağlamayı kapsamaktadır. Bu rapor; planlama, analiz, tasarım, gerçekleştirim ve bakım aşamalarını **Şelale Modeli** çerçevesinde ayrıntılı ve **net** bir şekilde ele almaktadır.

---

1. Planlama Aşaması

1.1 Proje Amacı ve Hedefleri

Proje Amacı: Kumaş üzerindeki delikleri otomatik olarak tespit edebilen bir derin öğrenme modelini eğitmek ve bu modeli mobil bir uygulamaya entegre etmektir.

Hedefler:

1) Delik tespiti için YOLOv8 tabanlı bir model eğitmek.
2) Eğitilen modeli mobil uygulama (Flutter) aracılığıyla kullanıcı dostu bir arayüzle kullanıma sunmak.
3) Mobil uygulama ile model arasında iletişimi sağlayacak bir backend (FastAPI) oluşturmak.
4) Uygulamanın doğruluk, performans ve kullanıcı deneyimini optimize etmek.

1.2 Proje Kapsamı

a) Veri toplama, düzenleme ve verisetinin hazırlanması.
b) YOLOv8 modelinin eğitimi ve optimize edilmesi.
c) Modelin mobil uygulama için daha hızlı çalışması için gerekli format dönüşümlerinin yapılması.
d) FastAPI backend geliştirilmesi ve mobil uygulamanın bu backend ile iletişiminin sağlanması.
e) Uygulama arayüzü tasarımı, test edilmesi ve optimize edilmesi.

1.3 Kullanılan Araçlar ve Kaynaklar

Model Geliştirme ve Derin Öğrenme:

- **Python** -> Derin öğrenme modellerinin geliştirilmesi için temel programlama dili.
- **YOLOv8** -> Kumaş üzerindeki delikleri tespit etmek için kullanılan nesne tespiti algoritması.
- **TensorFlow** -> Model eğitimi ve çıkarımı için kullanılan açık kaynaklı bir derin öğrenme kütüphanesi.

- **PyTorch** -> Derin öğrenme modellerini esnek bir şekilde geliştirmek için kullanılan diğer bir güçlü kütüphane.
- **OpenCV** -> Görüntü işleme işlemleri için kullanılan kütüphane.
- **NumPy** -> Veri analizi ve işleme için kullanılan bilimsel hesaplama kütüphanesi.
- **Pillow** -> Görüntü işlemleri ve manipülasyonları için kullanılan bir Python kütüphanesi.

Veri Kaynakları ve Etiketleme:

- **Kaggle** -> Veri setleri bulmak ve paylaşmak için kullanılan bir platform.
- **Roboflow** -> Veri setleri bulmak , verileri etiketlemek ve işlemek için kullanılan bir platform.

Mobil Geliştirme ve Backend:

- **Flutter** -> Mobil uygulamaların çapraz platform geliştirilmesi için kullanılan bir framework.
- **Dart** -> Flutter projelerinde kullanılan programlama dili.
- **FastAPI** -> Backend API geliştirme için hızlı ve modern bir framework.
- **Uvicorn** -> FastAPI uygulamalarını çalıştırmak için kullanılan bir ASGI sunucusu.

Dağıtım ve Diğer Araçlar:

- **Render.com** -> Backend uygulamalarını barındırmak için kullanılan bir bulut platformu.
- **Postman** -> API testlerini gerçekleştirmek ve doğrulamak için kullanılan bir araç.

1.4 Zaman Planlaması

Proje Aşaması	Tahmini Süre
Veri toplama ve hazırlanması	2 Hafta
Model eğitimi ve iyileştirmeleri	3 Hafta
Backend (FastAPI) geliştirme	2 Hafta
Mobil uygulama (Flutter) geliştirme	2 Hafta
Test ve entegrasyon	2 Hafta
Bakım ve iyileştirmeler	Sürekli

Proje aşamaları ve tahmini süreleri yukarıdaki tabloda gannt şeması şeklinde belirtilmiştir.

2. Analiz Aşaması

2.1 Gereksinim Analizi

Kullanıcı Gereksinimleri:

- Kullanıcılar, kamera veya galeriden kumaş fotoğrafı yükleyerek delikleri tespit edebilmelidir.
- Uygulama; tespit edilen delik sayısını, doğruluk oranını ve deliklerin konumlarını kullanıcıya sunmalıdır.
- Uygulama, kullanıcı dostu ve anlaşılır bir arayüze sahip olmalıdır.

İşlevsel Gereksinimler:

- YOLOv8 modelinin kumaş üzerindeki delikleri tespit etmek için eğitilmesi.
- Mobil uygulamanın kullanıcıdan aldığı fotoğrafları backend'e gönderip sonrasında modelden gelen sonuçları alarak kullanıcıya göstermesi.
- Backend'in (FastAPI) mobil uygulamadan aldığı fotoğrafta delik tespiti yapması ve sonuçları dönmesi.

Performans Gereksinimleri:

- Modelin yüksek doğruluk ve yeterince hızlı tespit performansına sahip olması.
- Mobil uygulamada düşük gecikme ile tespit sonuçlarının kullanıcıya sunulması.
- Backend ve mobil uygulama arasındaki iletişimin hızlı ve güvenilir olması.

2.2 Verisetinin Analizi ve Hazırlanması

- **Veri Toplama:** Kaggle ve Roboflow gibi platformlardan kumaş delikleri ile ilgili verisetleri toplandı.
- **Veri Temizleme ve Düzenleme:**
  - ❖ Elde edilen veriler tek tek etiketlenerek model eğitimi için hazırlandı.
  - ❖ Veri setleri, YOLOv8 formatına uygun hale getirildi.
- **Veri Artırma (Augmentation):**
  - ❖ Modelin genelleme kabiliyetini artırmak için veri augmentasyonu teknikleri (yansıma, döndürme, kontrast değişiklikleri vb.) uygulandı.

2.3 Model Analizi ve Seçimi

- ➢ **Model Seçimi:**
  - ❖ YOLOv8, nesne tespiti için en son teknolojilerden biri olduğu için seçildi.
  - ❖ Modelin gerçek zamanlı nesne tespiti ve yüksek doğruluk özellikleri dikkate alındı.

- **Model Eğitimi ve Performansı:**
  - YOLOv8 modeli, hazırlanan verisetleri ile eğitildi.
- **Modelin Optimize Edilmesi:**
  - Mobil cihazlarda modelin performansını artırmak için model ağırlıklarında optimizasyon yapıldı.
  - **ONNX Formatına Dönüşüm:** Model, TFLite formatına daha kolay geçiş yapabilmek için önce .PT'den .ONNX formatına dönüştürülür. ONNX, modelin farklı platformlarda uyumlu çalışmasını sağlar.
  - **TFLite Formatına Dönüşüm:** ONNX formatındaki model, TensorFlow Lite (TFLite) formatına dönüştürülerek mobil cihazlarda düşük gecikme ve yüksek performansla çalışması sağlanır.

2.4 Sistem Analizi

Sistem, üç temel bileşene sahip:

1. **Mobil Uygulama (Flutter):**
   - Kullanıcı arabirimi ve kumaş fotoğrafı alma/delik tespiti sonuçlarını görüntüleme.
   - Kullanıcı, mobil uygulama üzerinden delik tespit işlemini başlatır.
2. **Backend (FastAPI):**
   - FastAPI, mobil uygulama ile model arasında bir köprü görevi üstlenir.
   - Mobil uygulamadan alınan fotoğrafı YOLOv8 modeline binary olarak gönderir. Model, tespit sonuçlarını işleyerek geri gönderir.
3. **Yapay Zeka Modeli (YOLOv8):**
   - Kumaş üzerindeki deliklerin tespitinde kullanılan derin öğrenme modeli.
   - TFLite formatında backend içerisinde çalışır.

Veri Akış Şeması:

1. Kullanıcı, mobil uygulamada fotoğrafı seçer veya çeker.
2. Mobil uygulama, fotoğrafı FastAPI endpoint'ine gönderir.
3. Backend, YOLOv8 modelini kullanarak fotoğraftaki delikleri tespit eder.
4. Backend, tespit sonuçlarını mobil uygulamaya JSON formatında gönderir.
5. Mobil uygulama, sonuçları kullanıcıya gösterir.

3. Tasarım Aşaması

3.1 Sistem Mimarisi Tasarımı

Sistem bileşenleri şu şekildedir:

I. **Mobil Uygulama (Flutter):**
   - Kullanıcıdan alınan fotoğrafı backend'e iletir ve model tespit sonuçlarını görüntüler.

II. **Backend (FastAPI):**
   - Modeli çalıştırarak fotoğraftaki delikleri tespit eder, sonuçları mobil uygulamaya iletir.

III. **Model (YOLOv8):**
   - Delik tespiti için eğitilmiş model, tespit işlemini gerçekleştirir.

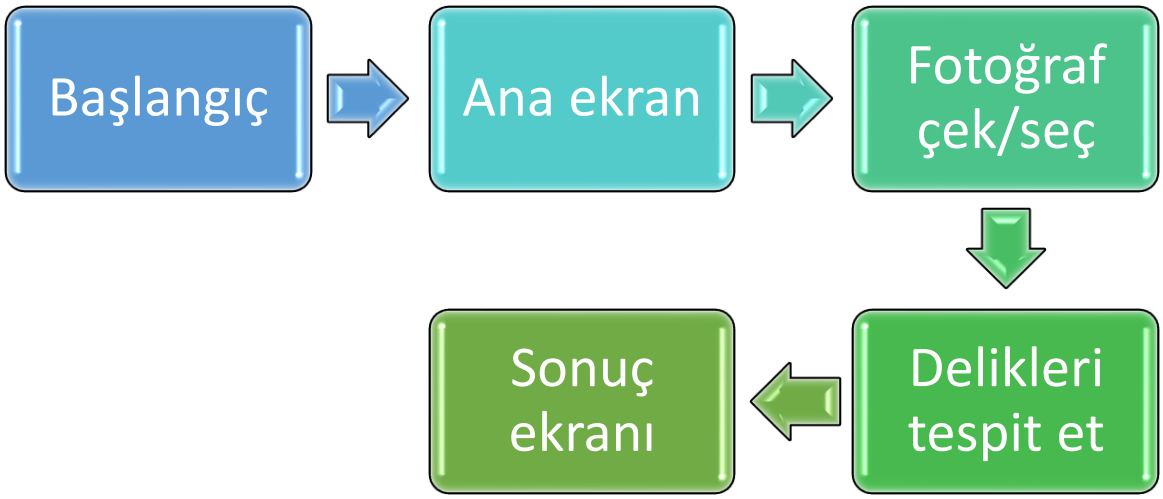
Sistem Mimarisi:

3.2 Kullanıcı Arayüzü Tasarımı

Mobil uygulama:

1. **Ana Ekran (HomeScreen):**
   - Kullanıcının kamera veya galeri aracılığıyla fotoğraf ekleyebildiği ekran.
   - Eklenen fotoğrafın önizlemesi ve "Delikleri Tespit Et" butonu.
2. **Delik Tespit Ekranı (DetectionScreen):**
   - Tespit edilen deliklerin gösterildiği ekran.
   - Model sonuçları, delik sayısı ve deliklerin doğruluk oranlarını içerir.

3.3 Mobil Uygulama Ekranı Akış Şeması

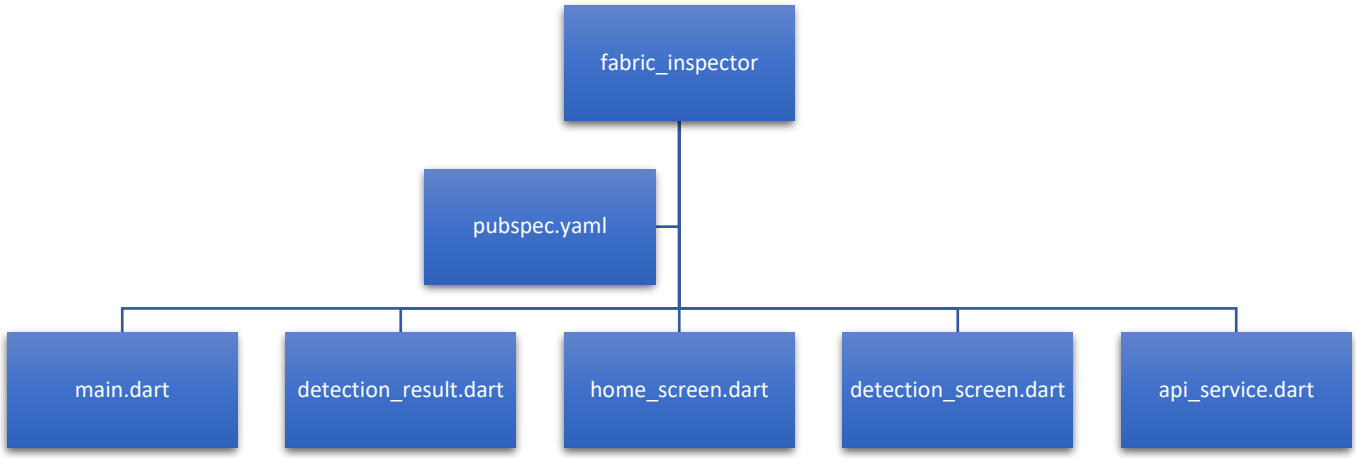


1. **Ana Ekran (HomeScreen):**
    - ❖ Uygulamanın başlangıç ekranıdır.
    - ❖ "Kamerayla Çek" ve "Galeriden Seç" butonları bulunur.
2. **Fotoğraf Seç/Çek (Kamera/Galeri):**
    - ❖ Kullanıcı, kamera veya galeri yardımıyla kumaş fotoğrafı seçer.
    - ❖ Seçilen fotoğraf ana ekranda gösterilir.
3. **"Delikleri Tespit Et" Butonu:**
    - ❖ Kullanıcı delik tespit işlemini başlatmak için bu butona tıklar.
    - ❖ Uygulama, fotoğrafı FastAPI aracılığıyla YOLOv8 modeline gönderir.
4. **FastAPI Aracılığıyla Model Tespiti:**
    - ❖ Backend, fotoğrafı model üzerinde çalıştırarak delikleri tespit eder.
    - ❖ Tespit sonuçları JSON formatında mobil uygulamaya geri gönderilir.
5. **Delik Tespit Ekranı (DetectionScreen):**
    - ❖ Kullanıcı, tespit sonuçlarının gösterildiği ekrana yönlendirilir.
    - ❖ **Gösterilen Bilgiler:**
        - ▪ Tespit edilen delikler ve kutucuklarla işaretlenmiş görsel.
        - ▪ Deliklerin sayısı ve doğruluk oranları.

3.4 Kaynak Dosya Hiyerarşisi ve Detaylı Açıklamalar

Bu bölümde, projede kullanılan dosya yapısı ve bu dosyaların görevleriyle ilgili detaylı bilgiler sunulacaktır.

3.4.1 Flutter Uygulama Dosyaları ve Kütüphaneleri



Flutter Proje Hiyerarşisi

---

1. pubspec.yaml

- Görevi: Flutter projesinin bağımlılıklarını, sürüm bilgilerini ve meta bilgileri tanımlar.
- Ayarlar:
  - dependencies: Uygulamanın ihtiyaç duyduğu paketleri belirtir.
  - assets: Özel fontlar ve ikonlar gibi kaynakların tanımlandığı kısım.
  - flutter_icons: Uygulamanın başlatma ikonunu özelleştirmek için kullanılan ayarları içerir.

---

2. main.dart

- Görevi: Uygulamanın başlangıç dosyasıdır, FabricInspectorApp sınıfını başlatır ve ekran rotalarını tanımlar.
- Kullanılan Sınıflar:
  - FabricInspectorApp: Stateless widget, uygulamanın giriş noktasıdır, tema ayarlarını ve ekran rotalarını içerir.

---

3. models/detection_result.dart

- Görevi: Delik tespit sonuçlarını modellemek ve JSON verilerini nesnelere dönüştürmek için kullanılır.
- Kullanılan Sınıflar:
  - DetectionResult: Görsel ve tespit edilen delikleri tutar, fromJson metodu JSON'u dönüştürür.

- Detection: Her delik tespitinin kutusu (box) ve doğruluk oranını içerir, fromJson metodu JSON'u dönüştürür.

4. screens/home_screen.dart

- Görevi: Ana ekranı oluşturur, kullanıcı görsel çeker veya seçer, ardından delik tespiti başlatır.
- Kullanılan Sınıflar:
  - HomeScreen: Stateful widget, görsel seçimi ve delik tespiti işlemlerini yönetir.
  - getImageFromCamera: Kameradan görsel alır ve sıkıştırır.
  - getImageFromGallery: Galeriden görsel alır ve sıkıştırır.
  - _detectHoles: Görseli backend'e gönderir ve sonuçları gösterir.
  - _HomeScreenState: Seçilen görseli ve yükleme durumunu yöneten state sınıfı.

---

5. screens/detection_screen.dart

- Görevi: Backend'den alınan delik tespit sonuçlarını kullanıcıya gösterir.
- Kullanılan Sınıflar:
  - DetectionScreen: Stateless widget, delikleri işaretleyerek kullanıcıya sunar.
  - Image.memory: Base64 formatındaki görselleri görüntüler.
  - Card: Tespit edilen delikleri kartlarla listeler, tespit yoksa "Hiç delik tespit edilmedi" mesajını gösterir.

---

6. services/api_service.dart

- Görevi: Backend API ile iletişimi sağlar, görseli gönderir ve sonuçları JSON formatında alır.
- Kullanılan Sınıflar:
  - ApiService:
    - detectHolesInImage: Görseli POST isteğiyle backend'e gönderir, JSON sonuçlarını döndürür.
    - baseUrl: Backend URL'sini tanımlar.

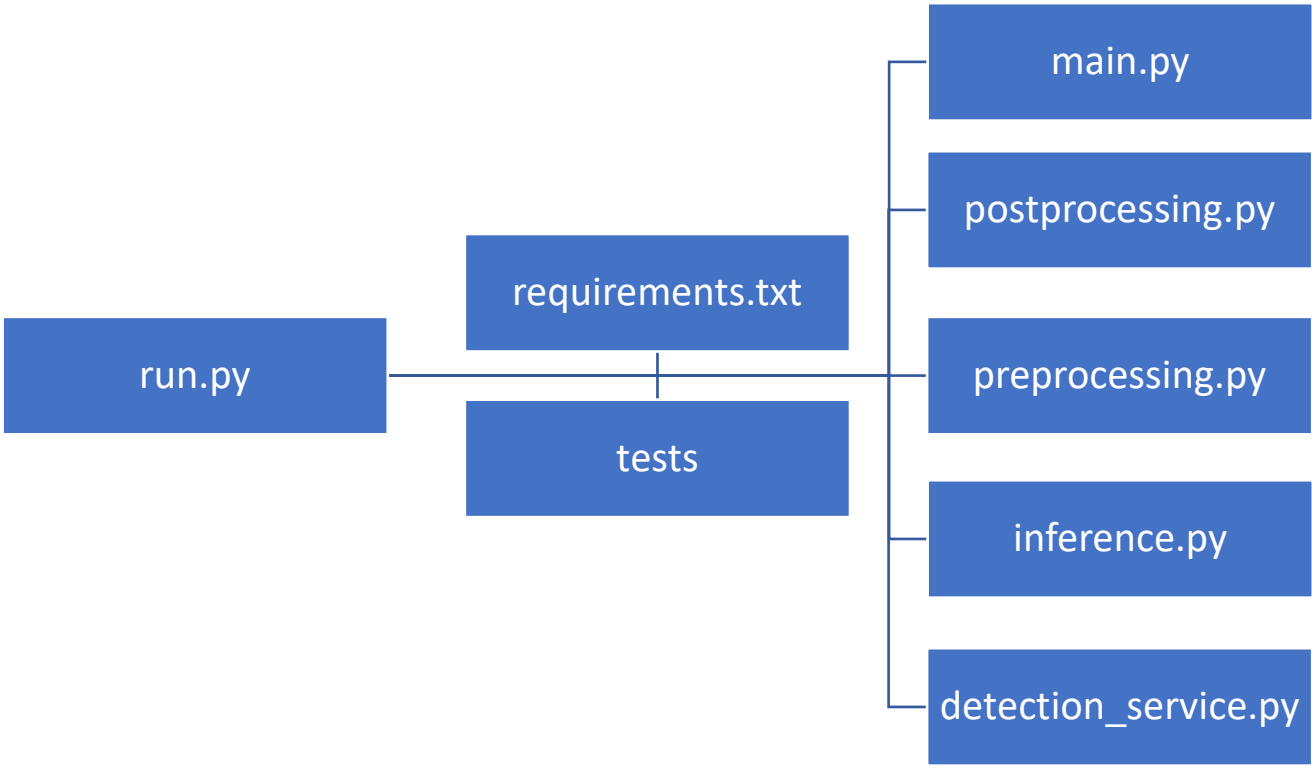
---

7. widgets/loading_indicator.dart

- Görevi: Yükleme sırasında dönen bir animasyon gösterir.
- Kullanılan Sınıflar:
  - LoadingIndicator: Stateless widget, ana yükleme bileşenidir.
  - _RotatingCircleIndicator: Stateful widget, dönen bir çember animasyonu oluşturur.
  - _CirclePainter: Çemberin görsel olarak çizimini yapan sınıf.

3.4.2 FastAPI Backend Dosyaları ve Kütüphaneleri

Backend Proje Hiyerarşisi



run.py → Uygulamayı başlatır.

requirements.txt → API'yı buluta deploy etmek için gerekli kütüphaneleri listeler.

- **fastapi:** API geliştirme.
- **uvicorn:** Sunucu çalıştırma.
- **opencv-python:** Görüntü işleme.
- **tensorflow/onnxruntime:** Model çalıştırma.
- **numpy:** Bilimsel hesaplama.
- **pillow:** Görüntü işlemleri.
- **python-multipart:** Dosya yükleme.
- **pydantic:** Veri doğrulama.

app/**main.py** → FastAPI uygulamasının ana dosyası.
app/model/**model.tflite** → YOLOv8 modelinin TFLite formatında çalıştırılabilir dosyası.
app/model/**class_names.txt** → Model tarafından tespit edilen sınıfların isimleri yazar.
app/services/**detection_service.py** → Delik tespiti yapan hizmet.
app/utils/**inference.py** → Model çıkarımı ve tahmin işlemleri.
app/utils/**postprocessing.py** → Çıktıları işler ve üst üste binen kutuları kaldırır.
app/utils/**preprocessing.py** → Görselleri yeniden boyutlandırır ve ön işler.

3.4.3 ONNX ve TFLite Format Dönüşümü

- **Neden ONNX'e Dönüşüm?**
    - YOLOv8 modeli .pt formatında eğitilir. Bu format, PyTorch'un orijinal formatıdır.
    - Modeli mobil uygulamalar için daha performanslı kullanabilmek için TFLite formatına dönüştürmek gerekir.
    - ONNX (Open Neural Network Exchange) formatı, farklı derin öğrenme çerçeveleri arasında **model geçişini kolaylaştırır**.
    - .pt formatındaki model önce .onnx formatına dönüştürülür. Bu adım, modelin TensorFlow Lite formatına daha kolay dönüştürülmesine yardımcı olur.
- **ONNX'ten TFLite Formatına Dönüşüm:**
    - Model .onnx formatına dönüştürüldükten sonra, ONNX Runtime veya TensorFlow aracılığıyla .tflite formatına dönüştürülür.
    - TFLite formatı, mobil ve gömülü cihazlarda derin öğrenme modellerinin hızlı ve verimli bir şekilde çalışmasına imkan tanır.

Bu dönüşüm süreci modelin mobil ortamlarda optimal performansla çalışmasını sağlar.

---

4. Gerçekleştirim (İmplementasyon) Aşaması

4.1 Model Eğitimi ve Geliştirme

1. **Veri Toplama ve Etiketleme:**
    - ✓ Kaggle ve Roboflow platformlarından elde edilen kumaş delikleri veri setleri kullanılmıştır. Her indirilen veriseti için veri temizleme vb işlemler yapılmıştır.
    - ✓ Bu veriler tek tek etiketlenerek YOLOv8 modelinin eğitimi için hazırlanmıştır.
    - ✓ Veriler; train (%75), valid (%10) ve test (%15) bölümlere ayrılmıştır.

2) Eğitim Süreci:

Model ve Çerçeve

- YOLOv8 modeli, PyTorch ve Ultralytics YOLOv8 çerçevesi kullanılarak **Google Colab** ortamında eğitilmiştir.

Eğitim Ayarları

- **Veri kümesi yolu**: data.yaml -> Eğitim sırasında kullanılacak veri kümesine ait sınıf bilgileri ve yolları içerir.
- **Epoch sayısı**: 100 -> Modelin veri kümesini kaç kez işleyeceğini belirler, öğrenme süresini etkiler.

- **Batch boyutu**: 16 -> Eğitim sırasında aynı anda işlenen veri sayısını belirtir, belleği etkiler.
- **Görüntü boyutu**: 640 -> Girdi görüntülerinin çözünürlüğünü belirler, modelin ayrıntı algısını etkiler.
- **Optimizasyon algoritması**: AdamW -> Ağırlıkları optimize eder, düzenleme ve kararlılığı artırır.
- **Başlangıç öğrenme oranı (lr0)**: 0.001 -> Modelin öğrenme hızını başlangıçta ayarlar.
- **Son öğrenme oranı (lrf)**: 0.0001 -> Eğitim sonunda öğrenme hızını düşürerek kararlılığı artırır.
- **Ağırlık çürümesi (weight_decay)**: 0.0005 -> Ağırlıkları küçülterek aşırı öğrenmeyi önler.
- **Early stopping sabrı**: 20 -> Doğrulama iyileşmezse eğitimi 20 epoch sonra durdurur.

Ek Özellikler

- **Veri artırma (augmentation)** -> Modelin daha çeşitli veri üzerinde eğitilmesini sağlar ve genelleme yeteneğini artırır.
- **Doğrulama işlemi** -> Her epoch sonunda model performansını izlemek için kullanılır.
  - ✓ **Eğitim sonrası elde edilen sonuçlar:**
    - ➢ **mAP (mean Average Precision): %95.1** -> Tespit doğruluğunun genel ortalaması.
    - ➢ **Precision (Kesinlik): %97.9** -> Doğru tespit oranı .
    - ➢ **Recall (Duyarlılık): %86.8** -> Kaçırılmayan nesne oranı.

➢ **Modelin Optimize Edilmesi ve Format Dönüşümleri:**

- ✓ Eğitilen model .pt formatındadır.
- ✓ Model, ONNX formatına (.onnx) dönüştürülerek platform bağımsız hale getirilir.
- ✓ ONNX formatındaki model, TensorFlow Lite (TFLite) formatına (.tflite) dönüştürülerek mobil cihazlar için daha verimli duruma getirilir.
- ✓ Bu dönüşümler, modelin mobil uygulamalarda düşük gecikme ve daha yüksek performansla çalışmasına yardımcı olur.

4.2 Backend (FastAPI) Geliştirme

1. **FastAPI Projesinin Yapılandırılması:**
   - ➢ FastAPI kütüphanesi ve gerekli bağımlılıklar requirements.txt dosyasında belirtilmiştir.
   - ➢ main.py dosyasında FastAPI uygulaması başlatılmıştır.
   - ➢ /detect endpoint'i tanımlanmıştır:
     - Bu endpoint, mobil uygulamadan gelen görseli alır.

- detection_service.py içindeki detect_holes fonksiyonunu çağırarak modelin tespit işlevini gerçekleştirir.
- Tespit sonuçlarını mobil uygulamaya JSON formatında geri döndürür.

2. **TFLite Model Entegrasyonu:**
   - ✓ inference.py ve detection_service.py dosyalarında TFLite modelini çalıştırmak için gerekli kodlar yazılmıştır.
   - ✓ Model, giriş olarak verilen görüntüde delikleri tespit eder ve konumlarını döndürür.
3. **Test ve Doğrulama:**
   - ✓ Postman veya benzeri API test araçları kullanılarak /detect endpoint'inin doğru çalıştığı test edilmiştir.
   - ✓ Modelin tespit performansı kontrol edilmiş ve gerekli olduğu durumlarda iyileştirmeler yapılmıştır.

4.3 Mobil Uygulama (Flutter) Geliştirme

a) **HomeScreen Geliştirme:**

- HomeScreen dosyası kullanıcı arayüzünü içerir:
  - Kamera veya galeriden fotoğraf seçmek için butonlar mevcuttur.
  - Seçilen fotoğraf ekranda gösterilir.
  - "Delikleri Tespit Et" butonu ile fotoğraf backend'e gönderilir.,
  - Performans optimizasyonu için flutter_image_compress kullanılarak görsel sıkıştırılarak boyut düşürülmüştür.
  - loading_indicator.dart widget'ı ile model işlemleri sırasında kullanıcıya görsel geri bildirim sağlanır.

b) **DetectionScreen Geliştirme:**

- DetectionScreen dosyası, tespit edilen deliklerin işlendiği ekranı içerir:
  - Tespit sonuçları (delik konumları ve skorları) ekranda gösterilir.
  - Kullanıcı dostu bir arayüzle delik sayısı, doğruluk oranları gibi bilgiler sunulur.

c) **API Service Entegrasyonu (api_service.dart):**

- ApiService sınıfı, detectHolesInImage fonksiyonuyla seçilen fotoğrafı base64 formatında encode ederek /detect endpoint'ine gönderir.
- Backend yanıtını alır ve DetectionScreen üzerinde gösterilmek üzere tespit bilgilerini döndürür.

d) **Test ve Optimizasyon:**

- Mobil uygulama emülatör ve **gerçek cihazlarda** test edilmiştir.

4.4 Tüm Sistem Senaryosu

1. Kullanıcı, uygulamanın arayüzünden bir görsel seçer ya da kameradan yeni bir fotoğraf çeker.
2. **"Delikleri Tespit Et"** butonuna basar.
3. **Seçilen görsel sıkıştırılır** (Flutter Image Compress) ve **JPEG** formatında kaydedilir.
4. Görselin byte verisi alınır ve **Base64 formatına dönüştürülür**, ardından HTTP POST isteğiyle sunucuya gönderilir.
5. FastAPI sunucusu, /detect endpoint'i üzerinden gelen POST isteğini karşılar ve **Base64** formatındaki görseli **byte verisine dönüştürür.**
6. Görsel, NumPy dizisine dönüştürülür, OpenCV ile yeniden boyutlandırılır (640x640).
7. Histogram eşitleme ve RGB formatına dönüştürme yapılır, TensorFlow **tensör formatına çevrilir**.
8. TFLite Interpreter, model.tflite dosyasını kullanarak **tespit yapar** ve kutular (boxes), skorlar (scores) ve sınıf kimlikleri (class_ids) döndürür.
9. **Non-Max Suppression (NMS)** uygulanarak **çakışan kutular filtrelenir** ve koordinatlar ölçeklendirilir.
10. Tespit edilen delikler, OpenCV kullanılarak görsel üzerine kutular ve **açıklamalarla çizilir**.
11. İşlenen görsel, **Base64 formatına çevrilerek** JSON objesi olarak API **yanıtına** eklenir.
12. Sunucudan gelen JSON yanıtı, mobil uygulama tarafından alınır; tespit bilgileri ayrıştırılır, görsel **Base64 formatından decode edilerek binary forma dönüştürülür**.
13. Deliklerin sayısı, doğruluk oranları ve işlenmiş görsel(Image.memory), kullanıcıya mobil uygulamada gösterilir.
14. Hiç delik tespit edilmediyse, kullanıcıya "Hiç delik tespit edilmedi" mesajı gösterilir.

Base64, fotoğrafın (binary verinin) **metne çevrilerek** JSON gibi formatlarla kolayca taşınmasını sağlar.

Sistem, tasarlandığı şekilde sorunsuz olarak çalışmaktadır.

---

5. Bakım Aşaması

1. **Model Güncellemeleri ve İyileştirmeler:**
   - ✓ Modelin doğruluk ve performansını iyileştirmek için **yeni veri** ile yeniden eğitilir.
   - ✓ Modelin ONNX ve TFLite formatına dönüştürme süreci ihtiyaç duyuldukça tekrarlanır.

2. **Uygulama Güncellemeleri:**
    - ✓ Kullanıcı geri bildirimlerine göre arayüz iyileştirmeleri yapılır.
    - ✓ Yeni özellikler (örneğin, farklı kumaş tipleri için tespit) eklenebilir.
    - ✓ Uygulamanın performansı ve kullanışlılığı arttırılır.
3. **Backend ve API Bakımı:**
    - ✓ API endpoint'leri düzenli olarak izlenir ve performans sorunları giderilir.
    - ✓ Uygulama uyumluluğunu ve güvenliğini sağlamak için bağımlılıklar güncellenir.
    - ✓ Gerekli olduğunda sunucu tarafında optimizasyonlar yapılır.
4. **Sorun Giderme ve Destek:**
    - ✓ Kullanıcılar veya otomatik hatalar tarafından bildirilen sorunlar ele alınır.
    - ✓ Sürekli izleme ile sistem kararlılığı ve kullanılabilirliği sağlanır.

Bakım aşaması, proje boyunca ve dağıtım sonrası da devam eder. Bu aşamada, sistemin güncel kalması, kullanıcı ihtiyaçlarını karşılaması ve en yeni teknolojiye uyum sağlaması için düzenli güncellemeler yapılır.

Sonuç

Bu raporda, derin öğrenme algoritmaları (YOLOv8) kullanarak kumaş üzerindeki deliklerin tespitine yönelik geliştirilmiş bir sistemin planlama, analiz, tasarım, gerçekleştirim ve bakım aşamaları detaylı bir şekilde ele alınmıştır. **Şelale Modeli** çerçevesinde ilerleyen proje kapsamında şu sonuçlar elde edilmiştir:

- ✓ YOLOv8 tabanlı bir model, kumaş üzerindeki delikleri yüksek doğrulukla tespit edebilecek şekilde eğitilmiştir.
- ✓ Eğitilen model, ONNX ve TFLite formatlarına dönüştürülerek backende entegre edilmiştir.
- ✓ Flutter kullanılarak geliştirilen mobil uygulama, FastAPI tabanlı bir backend aracılığıyla model ile iletişime geçmektedir.
- ✓ Kullanıcı, kamera veya galeri aracılığıyla kumaş fotoğrafı seçip "**Delikleri Tespit Et**" butonuna tek tıkayarak model sonuçlarını görebilmektedir.

Proje süresince elde edilen sonuçlar, modelin ve mobil uygulamanın başarılı bir şekilde entegre edildiğini göstermektedir. Sistemin bakım aşamasında, modelin ve uygulamanın güncellenmesi, kullanıcılardan gelen geri bildirimlere göre iyileştirmeler yapılması ve potansiyel sorunların giderilmesi hedeflenmiştir.

Bu proje, derin öğrenme modellerinin mobil uygulamalara entegrasyon sürecini anlamak ve uygulamak için kapsamlı bir örnek niteliği taşımaktadır.

Kaynakça

1) **Ultralytics YOLOv8 Dokümantasyonu**: https://docs.ultralytics.com
2) **Flutter Resmi Belgeleri**: https://docs.flutter.dev
3) **FastAPI Belgeleri**: https://fastapi.tiangolo.com
4) **Kaggle Verisetleri**: https://www.kaggle.com
5) **Roboflow Veri Etiketleme ve Yönetimi**: https://roboflow.com
6) **Scikit-learn, Keras ve TensorFlow ile Uygulamalı Makine Öğrenmesi**: Aurélien Géron, O'Reilly Media. (Gökhan hocam sağolsun.)
7) **ChatGPT:** https://chatgpt.com/
8) **OpenCV Resmi Belgeleri**: https://docs.opencv.org
9) **PyTorch Dokümantasyonu**: https://pytorch.org/docs
10) **Render Bulut Platformu Belgeleri (Backend Dağıtımı için)**: https://render.com/docs
11) **Google Colab Kılavuzu**: https://colab.research.google.com
12) **YOLOv8 ile Nesne Tespiti Eğitimi Blogları**: https://blog.roboflow.com

Bu rapor, proje sürecinin tamamını kapsamakta ve proje kapsamında yapılan tüm çalışmaların kronolojik bir özetini sunmaktadır. **Derin öğrenme modelinin mobil uygulamaya entegrasyon sürecini**, tüm sistemin/sistemlerin çalışma mantıklarını eksiksiz bir şekilde anlatılması hedeflenmiştir.