AI Destekli ve Platformdan Bağımsız Deri Tanı API’si

Ahmet Al Rusutm NO:210260612 2.Adham Wasim Sherif NO:190260612

3.Ömer Faruk Çelik NO:220260138

[1 2 3] Bilgisayar Mühendisliği Bölümü, Fırat Üniversitesi, Elâzığ

Ders: BMU 326 Yazılım Mühendisliği

Danışman: Dr. ZEYNEP KARACA

Sorumlu: Prof. Dr. Erdal Özbay

Tarih: 11 Haziran 2025

Özet

Bu rapor, cilt hastalıklarının yapay zekâ tabanlı görüntü işleme teknikleri ile tespiti için geliştirilen platformdan bağımsız bir mobil uygulama projesini detaylandırmaktadır. Proje, modern yazılım mühendisliği prensipleri çerçevesinde, mikroservis mimarisi kullanılarak hayata geçirilmiştir. Sistem, bir Flutter mobil istemcisi, Python tabanlı bir RESTful API ve Google Firebase kimlik doğrulama servisinden oluşmaktadır. Derin öğrenme modeli olarak, transfer öğrenme tekniği ile MobileNetV3Large mimarisi kullanılmış ve HAM10000 veri seti üzerinde eğitilmiştir. Proje yönetimi, Jira üzerinde Kanban metodolojisi ile gerçekleştirilmiş ve sürüm kontrolü için Git ve GitHub Flow iş akışları benimsenmiştir. Bu çalışma, teknolojiyi kullanarak sağlık hizmetlerine erişimi kolaylaştıran, kullanıcı dostu ve sürdürülebilir bir çözüm sunmaktadır.

Anahtar Kelimeler: *Cilt Hastalığı Tespiti, Derin Öğrenme, Flutter, Mikroservis Mimarisi, RESTful API, TensorFlow, Kanban, Firebase*

---

1. Giriş

Dermatolojik rahatsızlıklar, dünya genelinde en sık karşılaşılan sağlık sorunları arasında yer almaktadır. Erken teşhis, özellikle melanom gibi tehlikeli cilt kanseri türlerinde tedavi başarısını doğrudan etkileyen en kritik faktördür. Ancak, dermatoloji uzmanlarına erişimdeki coğrafi ve ekonomik engeller, birçok bireyin zamanında tanı almasını zorlaştırmaktadır. Bu projenin temel motivasyonu, yapay zekâ ve mobil teknolojilerin sunduğu imkanlardan faydalanarak, bu erişim sorununa pratik bir çözüm sunmaktır.

Geliştirilen "Cilt Hastalığı Tanı ve Öneri Mobil Uygulaması", kullanıcıların akıllı telefonları aracılığıyla ciltlerindeki lezyonların fotoğraflarını çekerek, saniyeler içinde potansiyel bir ön tanı

ve temel öneriler almasını sağlayan bir sistemdir. Uygulama, kullanıcı dostu arayüzü ve güvenli kimlik doğrulama sistemiyle öne çıkmaktadır. Bu sistem, tıbbi bir teşhisin yerini almayı hedeflememekle birlikte, kullanıcıyı bir sağlık profesyoneline danışması yönünde teşvik eden önemli bir ilk adım aracı olarak tasarlanmıştır.

1.1. Ek Puan İçin Neler Eklendi

- GitHub Flow yerine Git Flow branch
- Firebase
- API(RestfulAPI)
- veritaban

2. Yazılım Geliştirme Süreç ve Metodolojileri

Projenin geliştirme yaşam döngüsü, modern ve çevik (agile) yazılım mühendisliği pratikleri üzerine kurulmuştur.

2.1. Proje Yönetimi: Kanban ve Jira

Proje yönetimi için, esnekliği ve görsel takibi kolaylaştırması nedeniyle Kanban metodolojisi tercih edilmiştir. Tüm iş akışı, Jira platformu üzerinde oluşturulan bir Kanban panosu ile yönetilmiştir. Görevler; projenin genel hedeflerini temsil eden **Epic**, kullanıcı odaklı özellikleri tanımlayan **Story** ve teknik uygulama adımlarını içeren **Task** olarak hiyerarşik bir düzende organize edilmiştir.

2.2. Sürüm Kontrolü ve Entegrasyon: Git, GitHub ve Jira

Sürüm kontrol sistemi olarak Git tercih edilmiş, proje kodları merkezi bir GitHub deposunda barındırılmıştır. Branch yönetimi için, basit ve etkili bir model olan **GitHub Flow** iş akışı benimsenmiştir. Bu modele göre, main branch'i her zaman projenin kararlı ve dağıtıma hazır sürümünü temsil ederken, tüm yeni geliştirmeler amaca yönelik olarak isimlendirilmiş feature branch'lerinde gerçekleştirilmiştir.

Jira ve GitHub entegrasyonu sayesinde, her bir Jira görevi kendine özel bir Git feature branch'i ile ilişkilendirilmiş, geliştirme tamamlandıktan sonra **Pull Request (PR)** ve **Code Review** süreçlerinden geçirilerek main branch'ine birleştirilmiştir. Bu disiplinli yaklaşım, hem kod kalitesini artırmış hem de proje yönetimini şeffaf ve izlenebilir kılmıştır.

3. Sistem Mimarisi ve Tasarımı

Sistem, bakım kolaylığı, esneklik ve ölçeklenebilirlik sağlamak amacıyla **Mikroservis Mimarisi** temel alınarak tasarlanmıştır. Bu mimari, sistemi bağımsız, küçük ve odaklı servisler olarak böler. Her servis, kendi veritabanına sahip olabilme ve bağımsız dağıtım döngülerine sahip olma potansiyeli taşır.

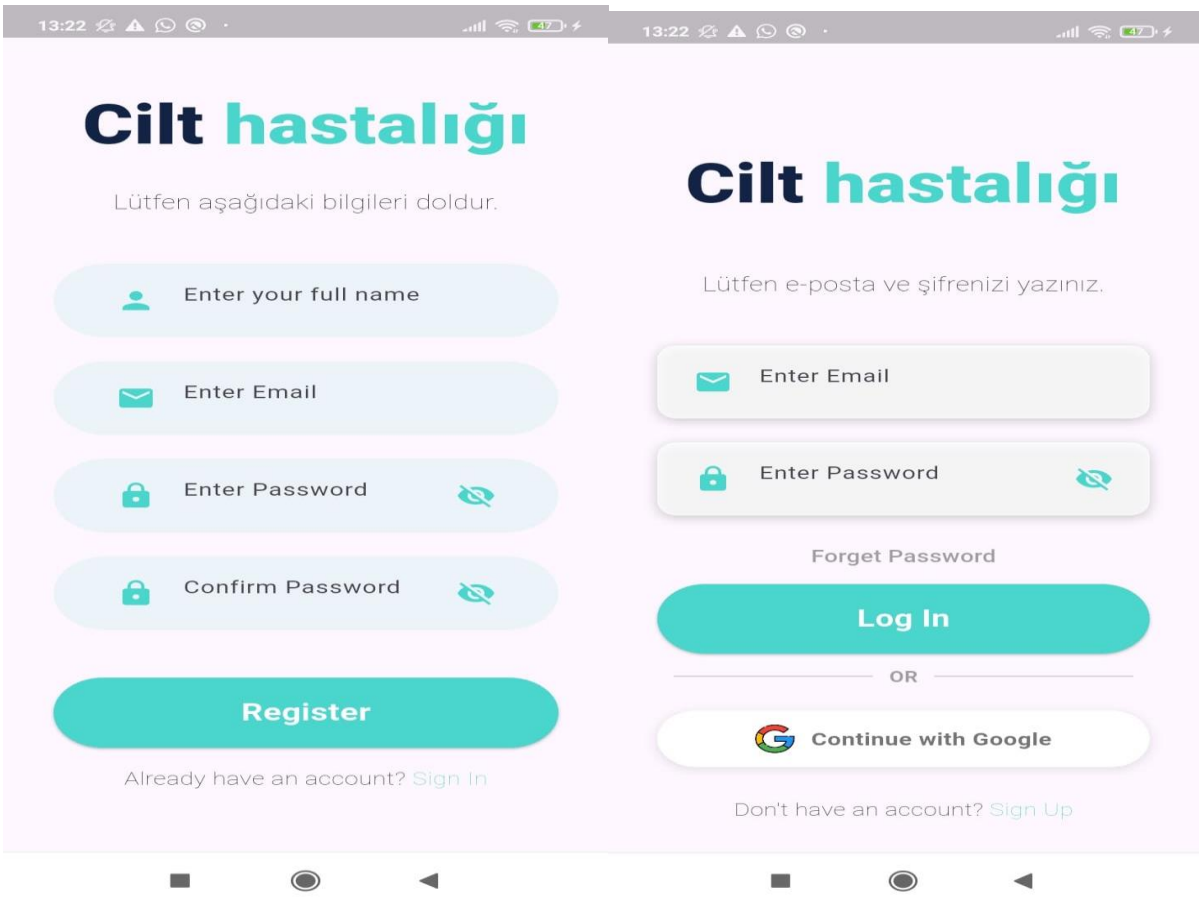
3.1. Mimarinin Bileşenleri (Mikroservisler)

1. **Flutter İstemcisi (Client):** Kullanıcı arayüzünü (UI) ve kullanıcı etkileşimlerini (UX) yöneten mobil uygulama.
2. **Tahmin Servisi (Prediction Service):** Python, Flask ve TensorFlow Lite ile geliştirilen, gelen görüntüleri analiz eden ve yapay zekâ modeli ile deri hastalığı sınıflandırması yapan RESTful API.
3. **Kimlik Doğrulama Servisi (Authentication Service):** Google'ın Firebase Authentication platformu kullanılarak gerçekleştirilen, kullanıcı yönetimi ve kimlik doğrulama işlemlerinden sorumlu harici servis.
4. **Öneri Servisi (Suggestion Service):** Teşhis sonrası tedavi önerilerini sunan mantıksal birim. Mevcut implementasyonda, bu servis hem backend tarafından dinamik olarak hem de istemci tarafında yerel bir veri dosyasından (disease_data.dart) yedekli olarak çalışmaktadır.
5. **Geçmiş Servisi (History Service - Gelecek Geliştirme):** Kullanıcıların teşhis geçmişini depolaması planlanan servis.
6. **Google Firebase ile Kullanici Kaydi Tutma:** Kullanıcıların bilgilerini ve daha önceden geçirdikleri hastaliklarin bilgisini tutar.

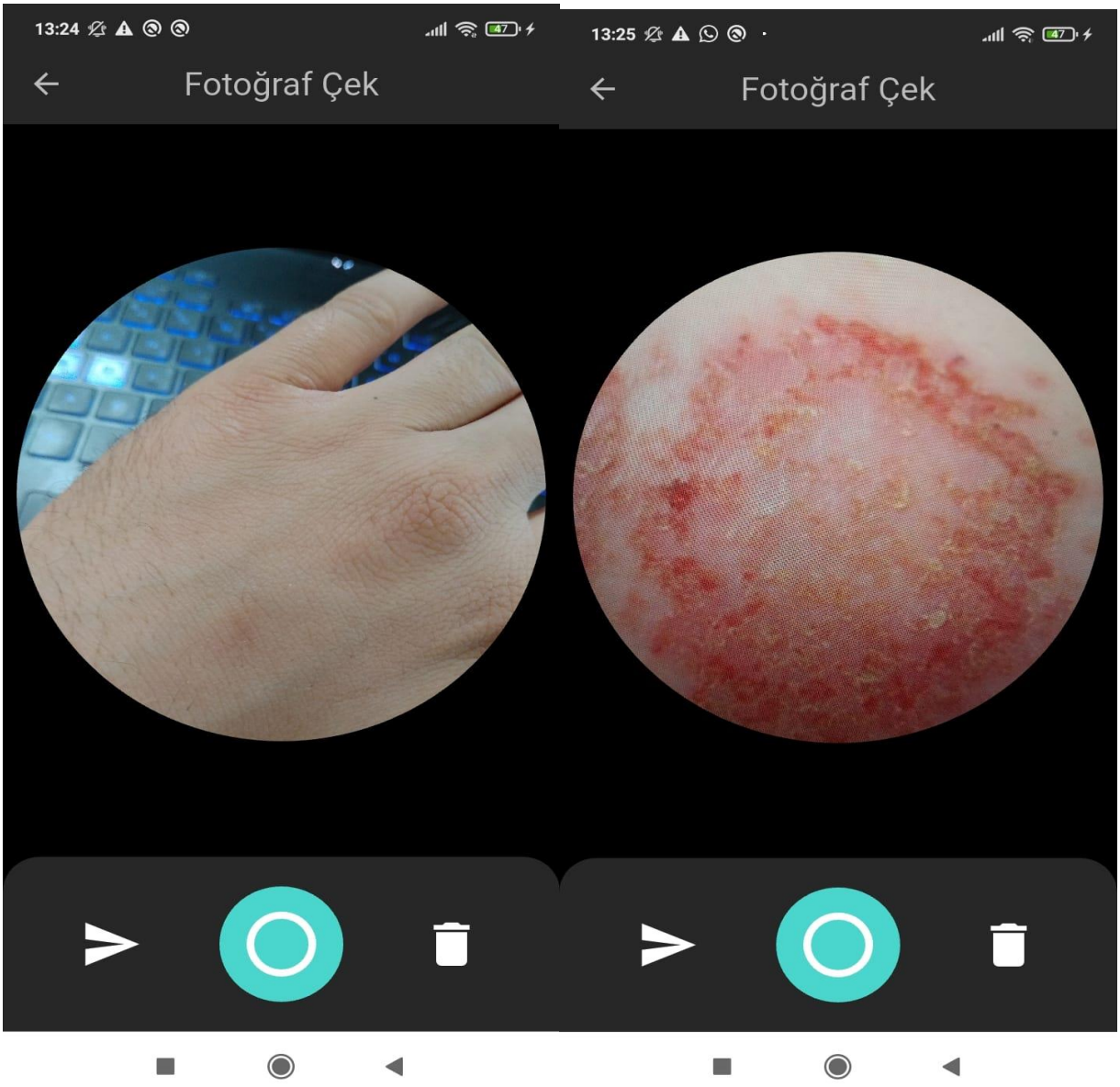
3.2. Temel Akış ve Uygulama Arayüzü (User Flow)

Uygulamanın kullanıcı akışı, basit ve sezgisel bir deneyim sunmak üzere tasarlanmıştır.

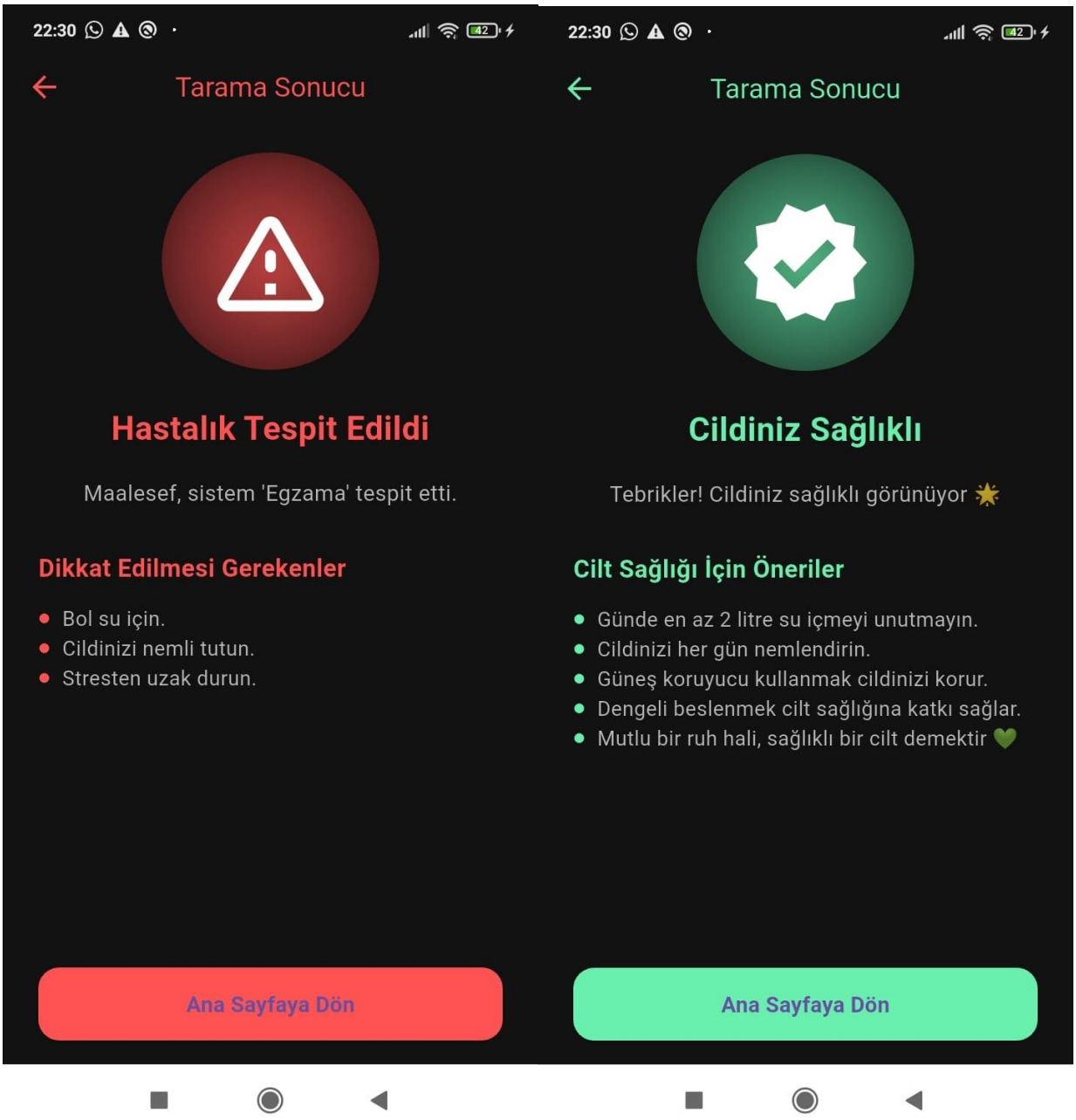
1. **Kimlik Doğrulama:** Kullanıcı, Şekil 1’de görülen modern tasarımlı ekranlar üzerinden e-posta/şifre ile uygulamaya giriş yapar veya kayıt olur.
2. **Fotoğraf İşleme:** Ana sayfadan, Şekil 2’de gösterildiği gibi, kamera veya galeriden fotoğraf yükleme seçeneklerine erişir.
3. **Analiz ve Sonuç:** Yüklenen fotoğraf API'ye gönderilir ve analiz edilir. Sonuçlar, Şekil 3’te görüldüğü gibi, hastalığın tespit edilip edilmemesine göre farklılaşan, anlaşılır ve yönlendirici bir arayüzde kullanıcıya sunulur.
4. **Kullanıcı Profili:** Kullanıcılar, Şekil 4'te gösterilen profil ekranından kişisel bilgilerini düzenleyebilir.



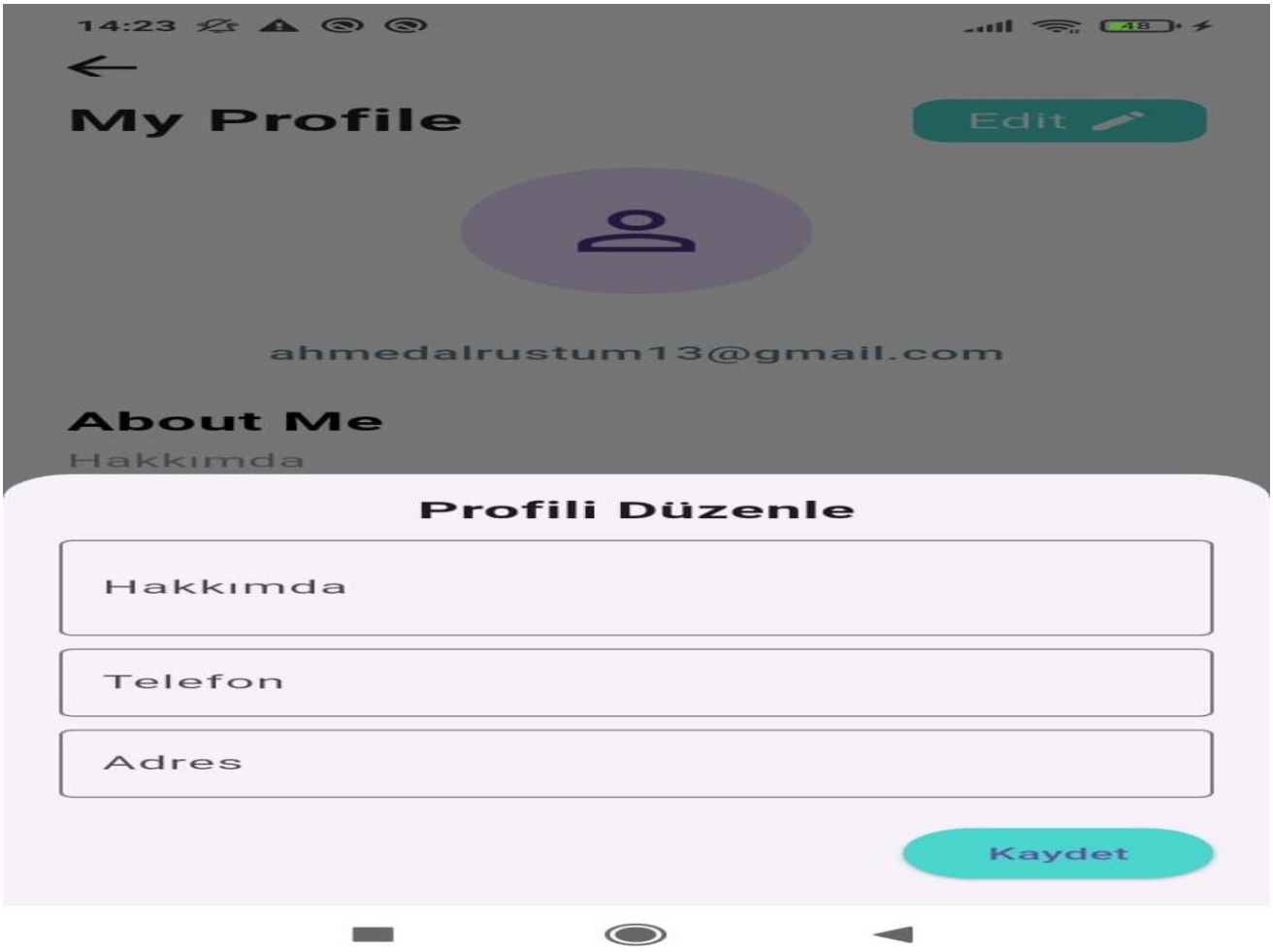
Şekil 1: Mobil Uygulama Giriş ve Kayıt Ekranları



Şekil 2: Kamera ve Galeri Kullanım Arayüzleri



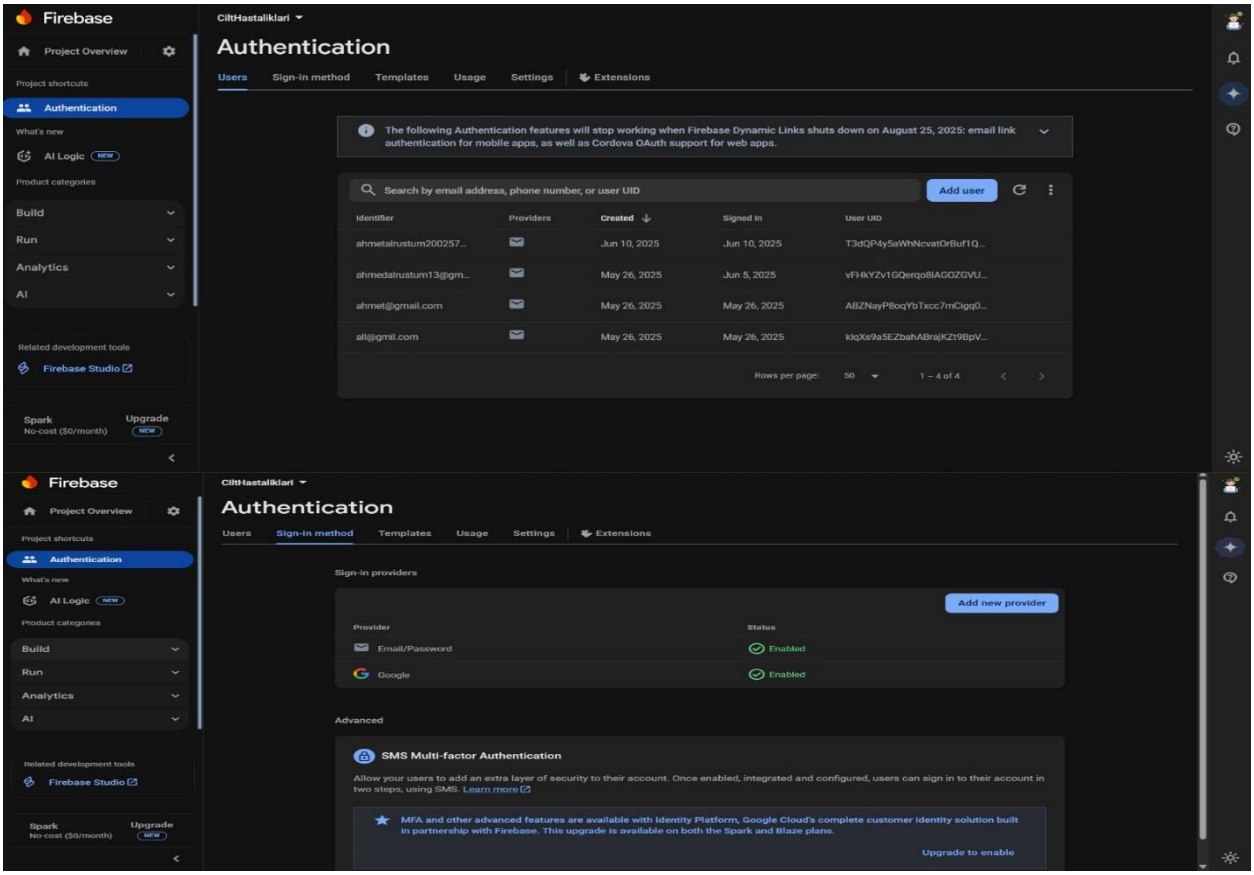
Şekil 3: Analiz Sonuç Ekranları



Şekil 4: Kullanıcı Profil Ekranı

Analiz sonuçlarına ek olarak, uygulama kişiselleştirilmiş bir kullanıcı deneyimi sunmayı hedefler. Şekil 4'te gösterilen "My Profile" ekranı, bu deneyimin merkezinde yer alır. Kullanıcılar bu ekranda, Firebase Authentication ile ilişkilendirilmiş olan e-posta adreslerini ve profil bilgilerini görüntüleyebilirler. Arayüz, kullanıcının temel bilgilerini net bir şekilde sunarken, sağ üst köşede bulunan "Edit" butonu ile kişiselleştirme fonksiyonlarına kolay erişim sağlar. Bu tasarım, kullanıcının uygulama üzerindeki kontrolünü ve aidiyet hissini artırmayı amaçlamaktadır.

Profil düzenleme işlevi, kullanıcı etkileşimini bir adım öteye taşır. "Edit" butonuna tıklandığında, ekranın altından modern bir "modal bottom sheet" açılarak kullanıcıya "Hakkımda", "Telefon" ve "Adres" gibi ek bilgileri girebileceği bir form sunulur. Bu bilgiler şu an için lokal olarak saklanmakta olup, projenin gelecek geliştirmeleri kapsamında bir veritabanına (örneğin Firebase Firestore) kaydedilerek, daha zengin ve kalıcı bir kullanıcı profili oluşturma potansiyeli taşımaktadır. Bu özellik, uygulamanın gelecekteki kişiselleştirilmiş öneriler veya raporlama gibi fonksiyonları için temel bir altyapı sunar.



Şekil 4a: Kullanıcı Giris Biliglerini Tutan Firebase Veritabani Entegresi

Projenin güvenli ve ölçeklenebilir bir kullanıcı yönetimine sahip olması için **Google Firebase** platformu tercih edilmiştir. Şekil 4a'da görüldüğü gibi, projemiz Firebase konsolu üzerinde oluşturulmuş ve **Authentication** servisi aktive edilmiştir. Bu entegrasyon sayesinde, e-posta/şifre tabanlı kullanıcı kaydı, girişi ve oturum yönetimi gibi kritik ve karmaşık işlemler, Google'ın sunduğu güvenli ve yönetilen altyapı üzerinden kolayca gerçekleştirilmiştir. Firebase kullanımı, geliştirme sürecini önemli ölçüde hızlandırmış ve projenin güvenlik standardını yükseltmiştir.

4. Teknolojik Altyapı ve Kütüphaneler

- **Frontend (Mobil Uygulama):**
  - **Flutter (Dart):** Tüm uygulama arayüzü ve iş mantığı.
  - **Firebase:** Kimlik doğrulama (Authentication) ve bulut altyapısı.
  - **http:** Sunucu ile REST API üzerinden veri alışverişi.
  - **image_picker:** Cihaz kamerası ve galerisinden görüntü seçimi.
- **Backend (API & AI):**
  - **Python:** Sunucu tarafı programlama dili.
  - **Flask:** Hafif ve esnek REST API altyapısı.

    - **TensorFlow & Keras:** Derin öğrenme modelinin eğitimi ve geliştirilmesi.
    - **TensorFlow Lite Runtime:** Optimize edilmiş yapay zekâ modelinin sunucuda çalıştırılması.
- **Veri Kümesi:**
    - **HAM10000:** Model eğitiminde kullanılan, 7 farklı kategoride 10.000'den fazla dermatolojik görüntü içeren halka açık veri kümesi.

5. Yapay Zekâ Modeli: Eğitim, Optimizasyon ve Değerlendirme

5.1. Model Mimarisi ve Eğitimi (train1234567.py)

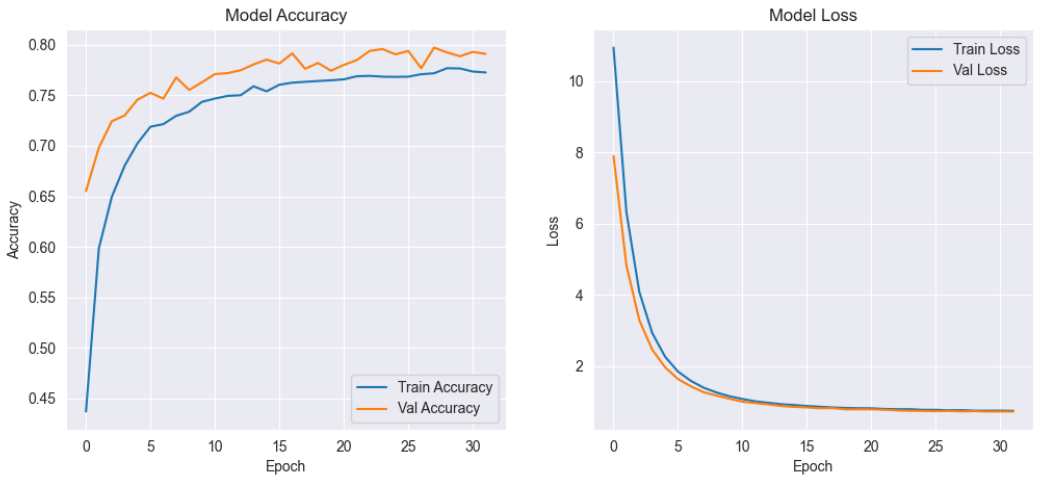
Modelin temelini, mobil cihazlar için optimize edilmiş, yüksek verimliliğe sahip **MobileNetV3Large** mimarisi oluşturmaktadır. **Transfer Learning** tekniği kullanılarak, ImageNet üzerinde ön-eğitilmiş ağırlıklar başlangıç noktası olarak alınmıştır. Bu yaklaşım, daha az veri ile daha hızlı ve daha yüksek doğrulukta bir model elde etmemizi sağlamıştır. Eğitim, **Feature Extraction** ve **Fine-Tuning** olmak üzere iki ana aşamada gerçekleştirilmiş ve sınıf dengesizliklerinin etkisini minimize etmek amacıyla **categorical_focal_loss** kayıp fonksiyonu kullanılmıştır.

5.2. Model Optimizasyonu ve Sunulması (convert_tflite.py, server.py)

Eğitilen Keras modeli, sunucuda yüksek performansla çalışabilmesi için **TensorFlow Lite (.tflite)** formatına dönüştürülmüştür. Bu süreçte, model boyutunu küçültmek ve çıkarım hızını artırmak için **Full Integer Quantization (INT8)** tekniği uygulanmıştır. Optimize edilen bu model, Flask API içerisine entegre edilerek, /images endpoint'i üzerinden gelen isteklere hizmet vermektedir.

5.3. Model Performansı ve Değerlendirme

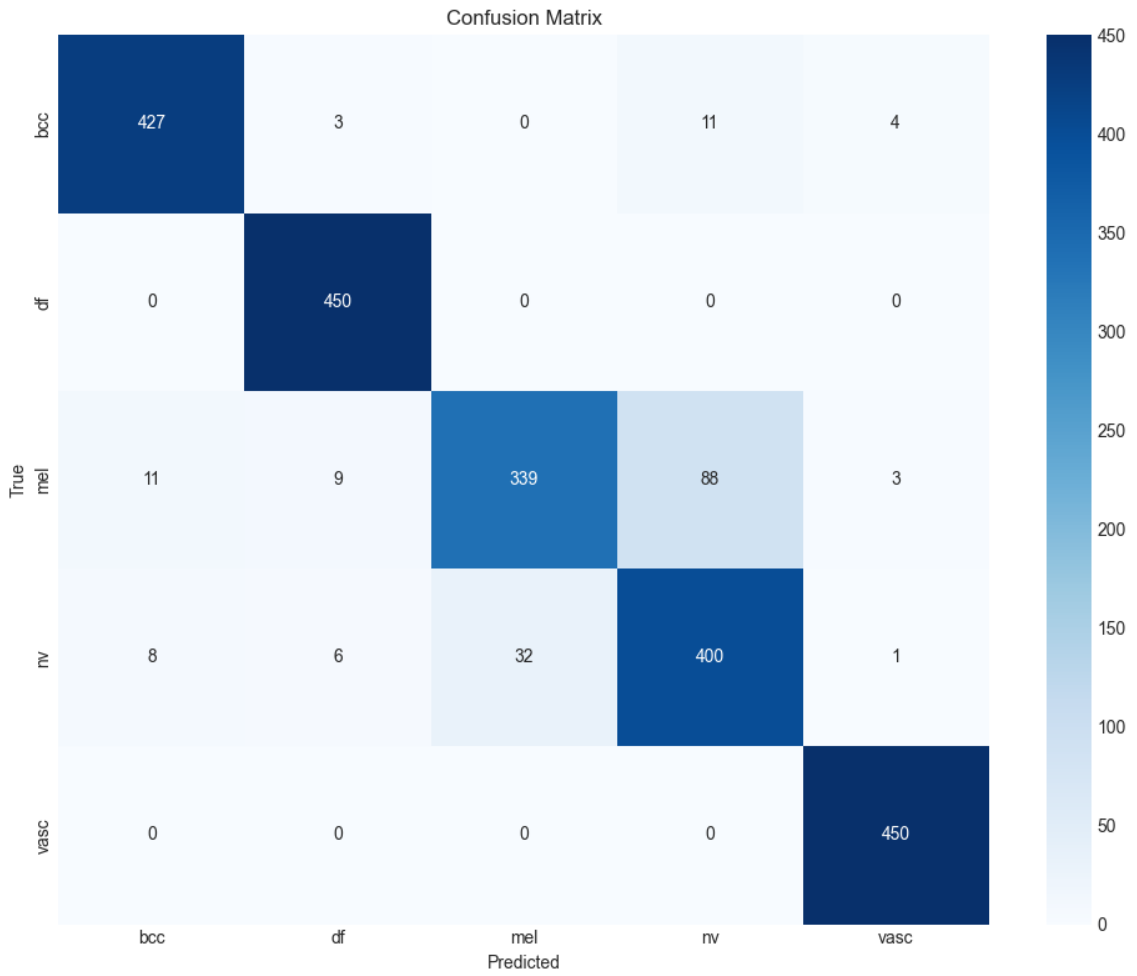
Modelin performansı, test veri seti üzerinde çeşitli metrikler kullanılarak titizlikle değerlendirilmiştir. Eğitim süreci boyunca modelin öğrenme eğilimleri **doğruluk (accuracy)** ve **kayıp (loss)** grafikleri ile izlenmiştir.



Şekil 5: Modelin Eğitim ve Doğrulama Doğruluk/Kayıp Grafikleri

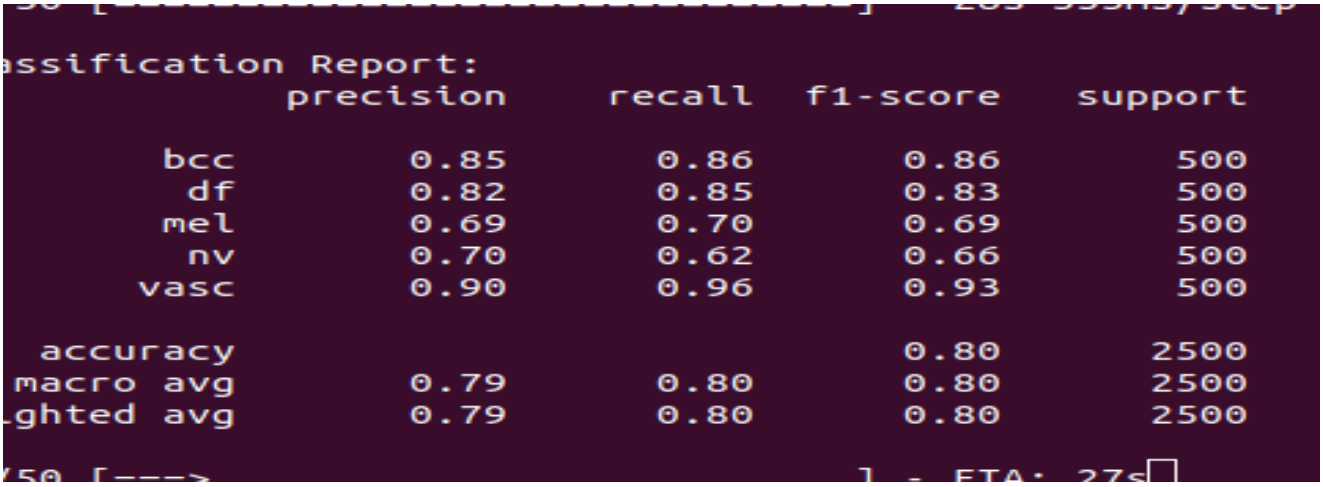
Şekil 5'te görüldüğü gibi, eğitim (Train) ve doğrulama (Validation) eğrileri birbirine paralel bir şekilde ilerlemektedir. Bu durum, modelin aşırı öğrenme (overfitting) problemine maruz kalmadığını ve genelleme yeteneğinin yüksek olduğunu göstermektedir. Model, yaklaşık 30 epoch sonunda %80'e yakın bir doğruluk oranına ulaşmıştır.

Modelin sınıflandırma yeteneğini daha detaylı analiz etmek için bir **Karmaşıklık Matrisi (Confusion Matrix)** ve **Sınıflandırma Raporu (Classification Report)** oluşturulmuştur (Bkz. Şekil 6 ve Şekil 7).



Şekil 6: Test Veri Seti Üzerinde Oluşturulan Karmaşıklık Matrisi

Şekil 6'daki karmaşıklık matrisi, modelin her bir sınıf için ne kadar doğru tahmin yaptığını göstermektedir. Özellikle df (dermatofibroma) ve vasc (vasküler lezyonlar) sınıflarında neredeyse mükemmel bir performans sergilediği görülmektedir. Modelin en çok zorlandığı sınıfın ise, diğer sınıflarla (özellikle nv - nevüs) karıştırabildiği mel (melanom) olduğu gözlemlenmiştir.

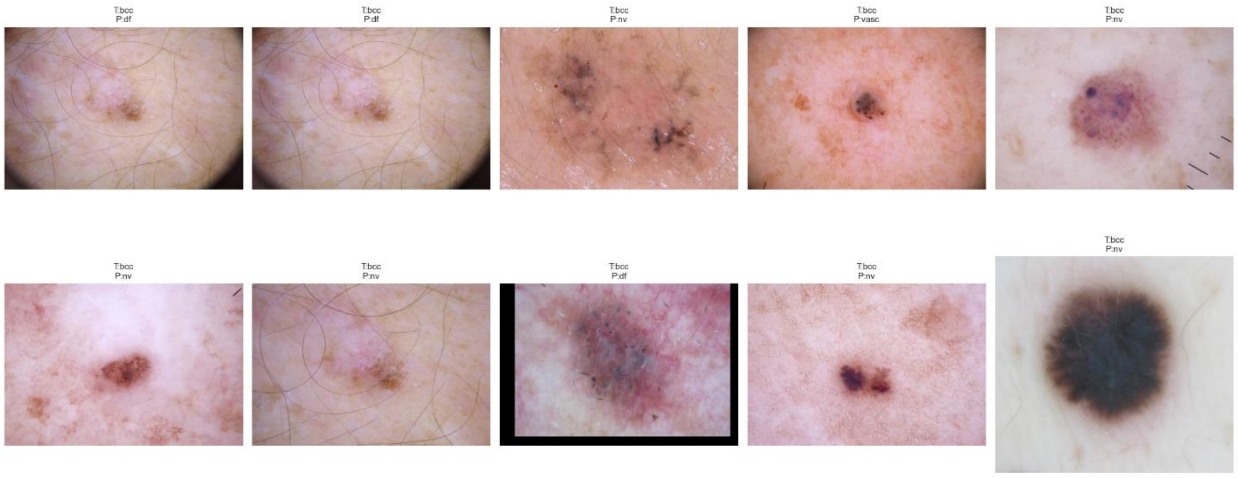


Şekil 7: Modelin Sınıflandırma Raporu

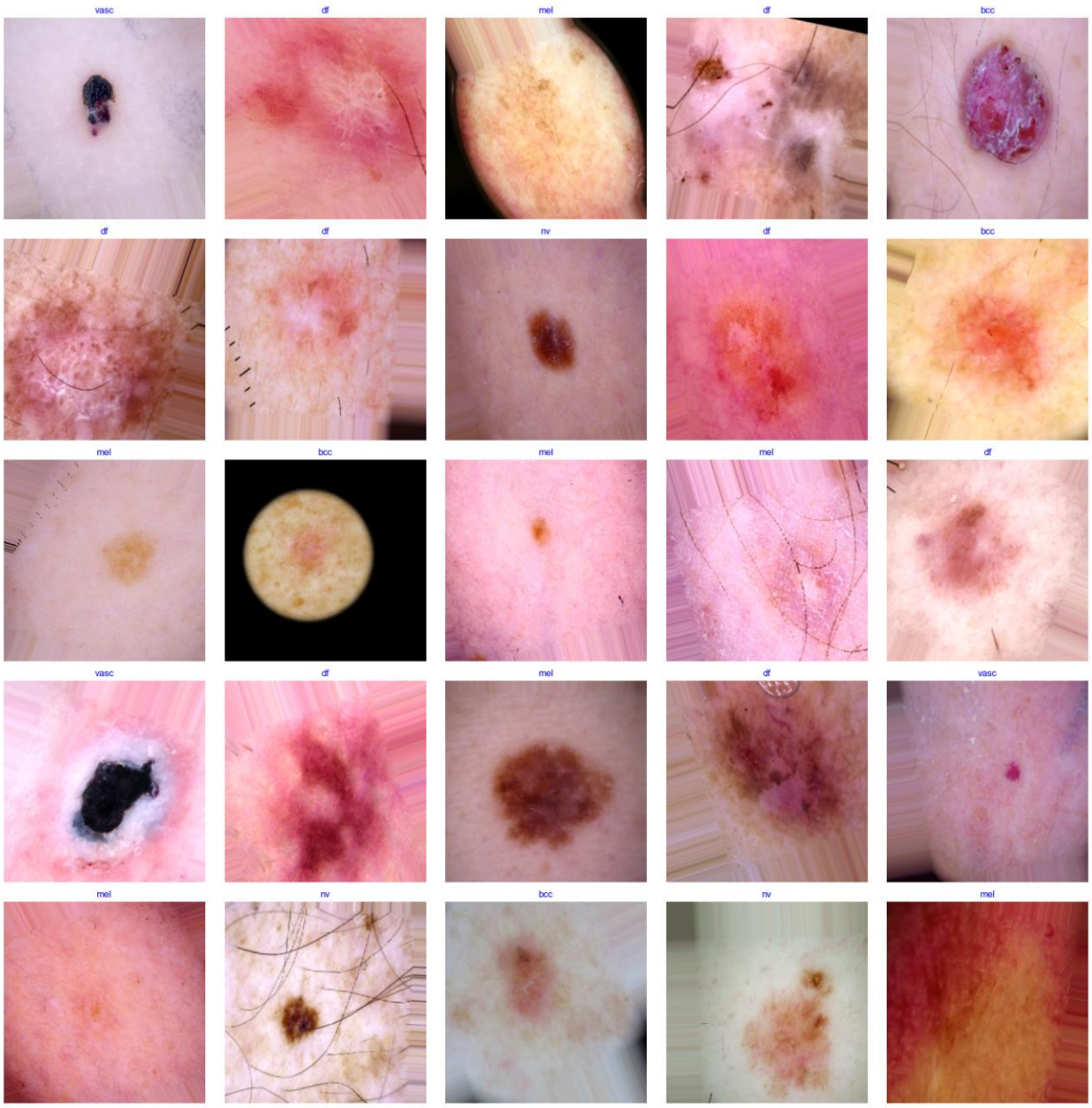
Şekil 7'deki sınıflandırma raporu, her bir sınıf için **precision (kesinlik)**, **recall (duyarlılık)** ve **f1-score** metriklerini sunmaktadır. Modelin genel doğruluğu %80 olarak ölçülmüştür. vasc sınıfı için elde edilen 0.93'lük F1 skoru, modelin bu sınıftaki başarısını teyit etmektedir. mel ve nv gibi görsel olarak birbirine benzeyen sınıflarda metriklerin daha düşük olması beklenen bir durum olup, bu sınıflar için modelin daha fazla veri ile veya farklı mimari yaklaşımlarla iyileştirilebileceğini göstermektedir.

5.4. Tahmin Örnekleri

Modelin tahmin yeteneğini görsel olarak incelemek amacıyla, test setinden bazı doğru ve yanlış sınıflandırma örnekleri Şekil 8 ve Şekil 9'da sunulmuştur.



Şekil 8: Modelin Yanlış Sınıflandırdığı Örnek Görüntüler (T: Gerçek Sınıf, P: Tahmin Edilen Sınıf)



Şekil 9: Modelin Doğru Sınıflandırdığı Örnek Görüntüler

6. Güvenlik, Test ve Bakım

- **Güvenlik:**
  - **SSL Sertifika Doğrulama:** Geliştirme ortamında HttpOverrides ile devre dışı bırakılan SSL doğrulaması, canlı (production) ortama geçilirken mutlaka aktive edilmelidir. Bu, istemci ile sunucu arasındaki iletişimin şifrelenmesini sağlar.
  - **Firebase Anahtarları:** Firebase konfigürasyon anahtarları gibi hassas bilgiler, sürüm kontrol sistemine doğrudan eklenmemeli ve güvenli bir şekilde yönetilmelidir.

- **Test ve Bakım:**
    - Uygulamanın temel akışları (giriş/kayıt, fotoğraf yükleme, sonuç görüntüleme) farklı cihazlarda ve senaryolarda manuel olarak test edilmiştir.
    - Kod tabanı, okunabilirliği ve modülerliği ön planda tutularak yazılmıştır. Her ekran kendi dosyasında modüler olarak tutulmuş, bu da bakım ve gelecekteki geliştirmeleri kolaylaştırmıştır.

7. Sonuç ve Gelecek Çalışmalar

Bu proje, cilt hastalıklarının hızlı ve kolay tespiti için modern, güvenli ve kullanıcı dostu bir çözüm sunmaktadır. Geliştirilen sistemin kod yapısı, modülerliği ve test edilebilirliği, projenin sürdürülebilirliğini güvence altına almaktadır.

Gelecek Geliştirmeler:

- **Google ile Giriş:** Kullanıcı deneyimini iyileştirmek için Firebase ile Google Sign-In entegrasyonunun tamamlanması.
- **Kullanıcı Geçmişi ve Raporlar:** Kullanıcıların geçmiş analizlerini görüntüleyebileceği bir profil ekranı ve analiz raporları.
- **Modelin Genişletilmesi:** Daha fazla cilt hastalığını tanıyabilmesi için modelin ek veri setleri ile yeniden eğitilmesi.
- **Çoklu Dil Desteği** ve **Gerçek Zamanlı Bildirimler** gibi ek özelliklerin eklenmesi.

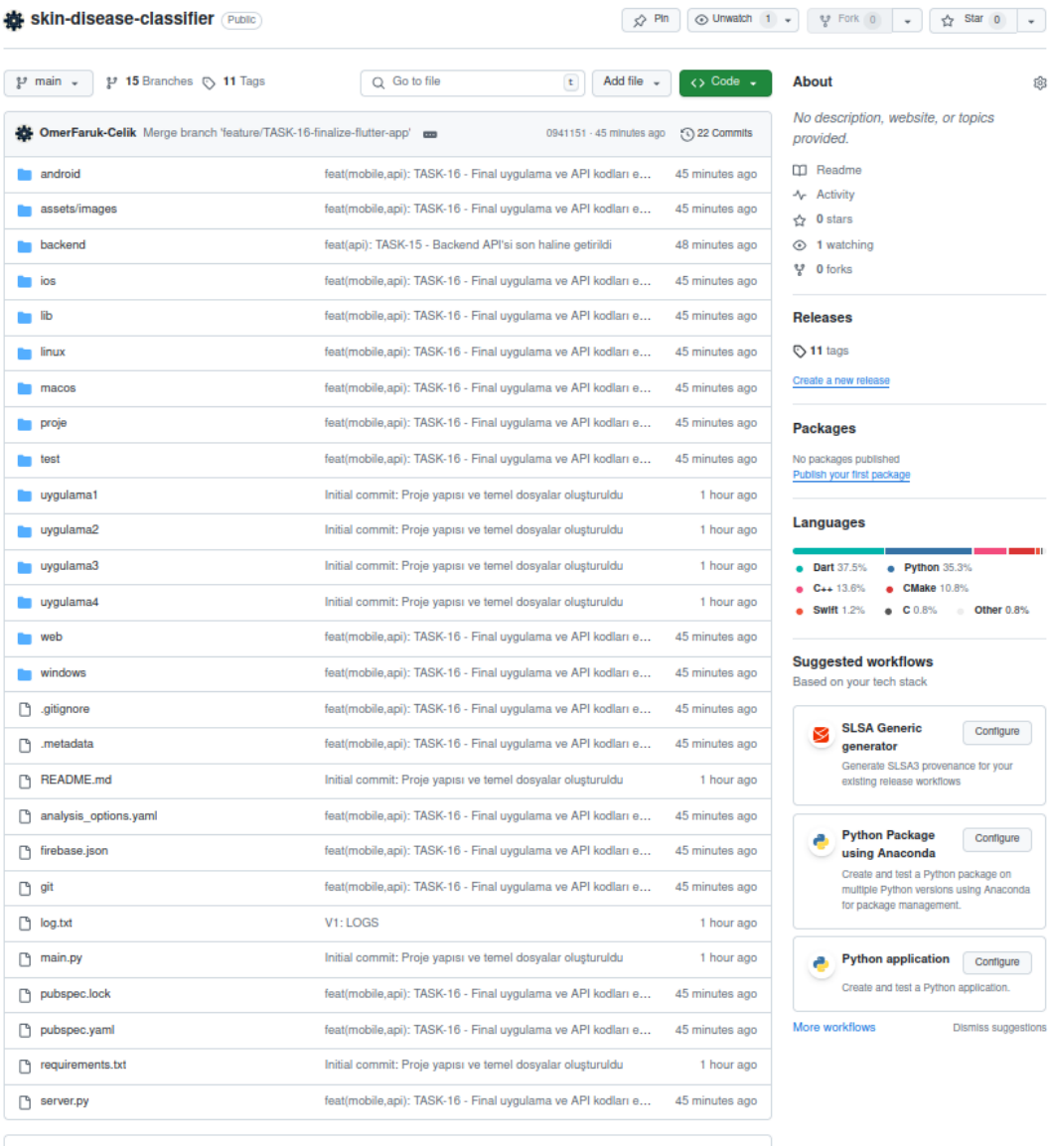
Bu geliştirmelerle, uygulamanın sunduğu değerin ve kullanıcı kitlesinin daha da artırılması hedeflenmektedir.

Ek-A: Git Commit Geçmişi (Sürüm Kontrol Logları Bir Kısmı)

Bu bölümde, projenin geliştirme yaşam döngüsü boyunca oluşturulan Git commit geçmişinin tamamı sunulmaktadır. Loglar, projenin dallanma yapısını, birleştirme noktalarını, sürüm etiketlerini ve her bir değişikliğin amacını göstererek, uygulanan Agile ve GitHub Flow metodolojisinin bir kanıtı niteliğindedir.

```
      *   0941151 - (HEAD -> main, tag: v1.0.0, origin/main) Merge branch
'feature/TASK-16-finalize-flutter-app' Closes: TASK-16 (8 days ago) <Ahmet Al
Rusutm>
|\
| * 3039274 - (feature/TASK-16-finalize-flutter-app) feat(mobile,api): TASK-
16 - Final uygulama ve API kodları entegre edildi (8 days ago) <Ahmet Al
Rusutm>
|/
*   0043fb0 - Merge branch 'feature/TASK-15-finalize-backend-api' feat:
Backend API stabilizasyonu tamamlandı. Closes: TASK-15 (11 days ago) <Omer
Faruk Celik>
|\
```

```
| * ff72be8 - (feature/TASK-15-finalize-backend-api) feat(api): TASK-15 -
Backend API'si son haline getirildi (11 days ago) <Omer Faruk Celik>
|/
* 7bfc5ee - V1: LOGS (12 days ago) <Omer Faruk Celik>
*   3630384 - (tag: v1.0.0-RC1) Merge branch 'feature/TASK-13-final-model-
tuning' (14 days ago) <Omer Faruk Celik>
|\
| * d485a20 - perf(ai): TASK-13 - Model, final hiperparametre ayarları ile
eğitildi ve optimize edildi (14 days ago) <Omer Faruk Celik>
|/
*   a9adfb9 - (tag: v0.9.0) Merge branch 'feature/TASK-8-result-screen' (18
days ago) <Ahmet Al Rusutm>
|\
| * c4ae098 - feat(ui): TASK-8 - Dinamik sonuç ekranı tasarlandı ve API'ye
bağlandı (18 days ago) <Ahmet Al Rusutm>
|/
*   7532a38 - (tag: v0.5.0) Merge branch 'feature/TASK-12-upgrade-to-
mobilenetv3' (22 days ago) <Omer Faruk Celik>
|\
| * 67835bb - feat(ai): TASK-12 - Model mimarisi daha verimli olan
MobileNetV3'e geçirildi (22 days ago) <Omer Faruk Celik>
|/
*   2604fb0 - (tag: v0.4.0) Merge branch 'feature/TASK-6-camera-module' (4
weeks ago) <Ahmet Al Rusutm>
|\
| * e0373ca - feat(mobile): TASK-6 - Kamera ve galeri ile görüntü yükleme
özelliği eklendi (4 weeks ago) <Ahmet Al Rusutm>
|/
*   825572e - (tag: v0.3.0) Merge branch 'feature/TASK-11-model-balancing-
improvement' (5 weeks ago) <Omer Faruk Celik>
|\
| * 3070617 - refactor(ai): TASK-11 - Model eğitim script'i veri dengeleme
teknikleri ile güncellendi (5 weeks ago) <Omer Faruk Celik>
|/
*   f92e3f2 - Merge branch 'feature/TASK-5-initial-ai-model' (6 weeks ago)
<Omer Faruk Celik>
|\
| * ec50c59 - feat(ai): TASK-5 - MobileNetV2 tabanlı ilk AI modeli entegre
edildi (6 weeks ago) <Omer Faruk Celik>
| * 279f526 - feat(ai): TASK-5 - MobileNetV2 tabanlı ilk AI modeli entegre
edildi (7 weeks ago) <Omer Faruk Celik>
|/
*   05e6a3b - (tag: v0.2.0) Merge branch 'feature/TASK-4-backend-setup' (7
weeks ago) <Omer Faruk Celik>
|\
| * bf96ec5 - feat(backend): TASK-4 - Flask API iskeleti ve Firebase Auth
entegrasyonu (7 weeks ago) <Omer Faruk Celik>
|/
* 3e8889f - (tag: v0.1.0) feat(ui): TASK-2 - Login, Register ve Home
ekranlarının temel UI'ı oluşturuldu (8 weeks ago) <Ahmet Al Rusutm>
* af83657 - (master) Initial commit: Proje yapısı ve temel dosyalar
oluşturuldu (2 months ago) <Omer Faruk Celik>
```



Şekil 10: Github Reposu

Ek-B: Flutter Mobil Uygulama ve Google Firebase ile Sunucu Baglanti Kodlari

```
// main.dart

import 'dart:io';
```

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_auth/firebase_auth.dart';

/* —— Firebase Options —— */
import 'package:test3/firebase_options.dart';

/* —— Ekranlar (Screens) —— */
import 'package:test3/Screens/Login.dart' show LogIn;
import 'package:test3/Screens/Register.dart' show Register;
import 'package:test3/Screens/Galeri.dart' show GaleriScanUI;
import 'package:test3/Screens/Camera.dart' show CameraScanUI;
import 'package:test3/Screens/MyProfile.dart' show UserProfile;
import 'package:test3/Screens/Result.dart' show ResultScreen;
import 'package:test3/Screens/Home.dart' show HomePage;

/* —— Hastalık verileri —— */
import 'package:test3/Screens/disease_data.dart'
    show diseaseNames, diseaseSuggestions;

/* —— SSL Sertifika Doğrulamasını Pas Geçmek İçin HttpOverrides —— */
class MyHttpOverrides extends HttpOverrides {
  @override
```

```
  HttpClient createHttpClient(SecurityContext? context) {
    var client = super.createHttpClient(context);
    // Tüm sertifikalar geçerli sayılır (Geliştirme için)
    client.badCertificateCallback =
        (X509Certificate cert, String host, int port) => true;
    return client;
  }
}

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // SSL doğrulamayı pas geçmek için override set et
  HttpOverrides.global = MyHttpOverrides();

  await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);

  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
```

```
Widget build(BuildContext context) {
 return MaterialApp(
  debugShowCheckedModeBanner: false,
  title: 'Cilt Uygulaması',
  theme: ThemeData(primarySwatch: Colors.blue),

  routes: {
   '/LogIn': (_) => const LogIn(),
   '/Register': (_) => const Register(),
   '/GaleriScanUI': (_) => const GaleriScanUI(),
   '/CameraScanUI': (_) => CameraScanUI(),
   '/UserProfile': (_) => const UserProfile(),
   '/HomePage': (_) => const HomePage(),
  },

  onGenerateRoute: (settings) {
   if (settings.name == '/ResultScreen') {
    final args = settings.arguments as Map<String, dynamic>?;
    final String? diseaseName = args?['diseaseName'] as String?;
    final bool hasDisease = diseaseName != "None";
    final List<String> suggestionsFromBackend =
        (args?['suggestions'] as List?)?.cast<String>() ?? <String>[];
    List<String> suggestions;
    if (suggestionsFromBackend.isNotEmpty) {
```

```
        suggestions = suggestionsFromBackend;
      } else {
        final index = diseaseNames.indexOf(diseaseName ?? '');
        suggestions = index >= 0 ? diseaseSuggestions[index] : <String>[];
      }
      return MaterialPageRoute(
        builder:
            (_) => ResultScreen(
              hasDisease: hasDisease,
              diseaseName: diseaseName,
              suggestions: suggestions,
            ),
      );
    }
    return null;
  },

  home: const AuthGate(),
 );
 }
}

/*———————————————— Auth Gate ————————————————*/

class AuthGate extends StatelessWidget {
```

```
  const AuthGate({super.key});

  @override
  Widget build(BuildContext context) {
    return StreamBuilder<User?>(
      stream: FirebaseAuth.instance.authStateChanges(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return const Scaffold(
            body: Center(child: CircularProgressIndicator()),
          );
        }

        if (snapshot.hasData) {
          return const HomePage();
        }

        return const LogIn();
      },
    );
  }
}
```

```
// firebase_options.dart
//.....................................................................................
```

```
// File generated by FlutterFire CLI.
// ignore_for_file: type=lint
import 'package:firebase_core/firebase_core.dart' show FirebaseOptions;
import 'package:flutter/foundation.dart'
    show defaultTargetPlatform, kIsWeb, TargetPlatform;

/// Default [FirebaseOptions] for use with your Firebase apps.
///
/// Example:
/// ```dart
/// import 'firebase_options.dart';
/// // ...
/// await Firebase.initializeApp(
///   options: DefaultFirebaseOptions.currentPlatform,
/// );
/// ```
class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
```

```
    return web;
  }
  switch (defaultTargetPlatform) {
    case TargetPlatform.android:
      return android;
    case TargetPlatform.iOS:
      return ios;
    case TargetPlatform.macOS:
      return macos;
    case TargetPlatform.windows:
      return windows;
    case TargetPlatform.linux:
      throw UnsupportedError(
        'DefaultFirebaseOptions have not been configured for linux - '
        'you can reconfigure this by running the FlutterFire CLI again.',
      );
    default:
      throw UnsupportedError(
        'DefaultFirebaseOptions are not supported for this platform.',
      );
  }
}

static const FirebaseOptions web = FirebaseOptions(
```

```
  apiKey: 'AIzaSyD0_XOAnq56a4dSYZfLWZo3mODgOsmr-JY',
  appId: '1:634361672473:web:4b511ba4a8e7c8df104b88',
  messagingSenderId: '634361672473',
  projectId: 'cilthastaliklari-2ef11',
  authDomain: 'cilthastaliklari-2ef11.firebaseapp.com',
  storageBucket: 'cilthastaliklari-2ef11.firebasestorage.app',
);

static const FirebaseOptions android = FirebaseOptions(
  apiKey: 'AIzaSyArhVmiqCF9BoNVWIK0uUYPXAdemKFSsys',
  appId: '1:634361672473:android:3a1b750ed5a045e6104b88',
  messagingSenderId: '634361672473',
  projectId: 'cilthastaliklari-2ef11',
  storageBucket: 'cilthastaliklari-2ef11.firebasestorage.app',
);

static const FirebaseOptions ios = FirebaseOptions(
  apiKey: 'AIzaSyCJPbqKH5tgRyqH1R-F-ZrVCt6u4CFp22Q',
  appId: '1:634361672473:ios:f44c64946fce6d7d104b88',
  messagingSenderId: '634361672473',
  projectId: 'cilthastaliklari-2ef11',
  storageBucket: 'cilthastaliklari-2ef11.firebasestorage.app',
  iosBundleId: 'com.example.test3',
);
```

```
  static const FirebaseOptions macos = FirebaseOptions(
    apiKey: 'AIzaSyCJPbqKH5tgRyqH1R-F-ZrVCt6u4CFp22Q',
    appId: '1:634361672473:ios:f44c64946fce6d7d104b88',
    messagingSenderId: '634361672473',
    projectId: 'cilthastaliklari-2ef11',
    storageBucket: 'cilthastaliklari-2ef11.firebasestorage.app',
    iosBundleId: 'com.example.test3',
  );

  static const FirebaseOptions windows = FirebaseOptions(
    apiKey: 'AIzaSyD0_XOAnq56a4dSYZfLWZo3mODgOsmr-JY',
    appId: '1:634361672473:web:61bdc7a496a044b5104b88',
    messagingSenderId: '634361672473',
    projectId: 'cilthastaliklari-2ef11',
    authDomain: 'cilthastaliklari-2ef11.firebaseapp.com',
    storageBucket: 'cilthastaliklari-2ef11.firebasestorage.app',
  );
}

// Camera.dart

//.....
```

```dart
import 'dart:convert';
import 'dart:io';

import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'package:http/http.dart' as http;



class CameraScanUI extends StatefulWidget {
  const CameraScanUI({super.key});

  @override
  State<CameraScanUI> createState() => _CameraScanUIState();
}

class _CameraScanUIState extends State<CameraScanUI> {
  final ImagePicker _picker = ImagePicker();
  File? _capturedImage;
  bool _isLoading = false;

  //----------------------------
  // FOTOĞRAF ÇEK (izin kontrolü olmadan, sadece image_picker)
```

```
//-----------------------------
Future<void> _takePhoto() async {
  try {
    final XFile? image = await _picker.pickImage(source: ImageSource.camera);
    if (image != null) {
      setState(() => _capturedImage = File(image.path));
    }
  } catch (e) {
    _showSnackBar('Kamera açılırken bir hata oluştu. Lütfen tekrar deneyin.');
  }
}

//-----------------------------
// FOTOĞRAFI YÜKLE + TAHMİN AL
//-----------------------------
Future<Map<String, dynamic>> _uploadAndPredict(File image) async {
  //---------------------------------------------------------------
  // 1) Fotoğrafı POST /images ile sunucuya gönder
  //---------------------------------------------------------------
  final uploadUri = Uri.parse('https://omerfarukcelik.duckdns.org:5001/images');

  final request = http.MultipartRequest('POST', uploadUri)
    ..files.add(await http.MultipartFile.fromPath('file', image.path));
```

```
  final uploadResponse = await request.send();

  // 200, 201, 202 gibi başarılı kodlar kabul edilir
  if (uploadResponse.statusCode < 200 || uploadResponse.statusCode >= 300) {
    final body = await uploadResponse.stream.bytesToString();
    throw Exception('Yükleme başarısız: $body');
  }

  //---------------------------------------------------------------
  // 2) GET /tahmin ile tahmin iste
  //---------------------------------------------------------------
  final predictUri = Uri.parse('https://omerfarukcelik.duckdns.org:5001/tahmin');
  final predictResponse = await http.get(predictUri);

  if (predictResponse.statusCode != 200) {
    throw Exception(
        'Tahmin alınamadı: ${predictResponse.statusCode} – ${predictResponse.body}');
  }

  //---------------------------------------------------------------
  // 3) JSON cevabı Map<String, dynamic> olarak döndür
  //---------------------------------------------------------------
  return jsonDecode(predictResponse.body) as Map<String, dynamic>;
}
```

```
//----------------------------
// GÖNDER BUTONUNU YÖNET
//----------------------------
void _handlePredictButton() async {
  if (_capturedImage == null) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Lütfen önce bir fotoğraf seçin.'),
        backgroundColor: Colors.redAccent,
        behavior: SnackBarBehavior.floating,
      ),
    );
    return;
  }

  setState(() => _isLoading = true);

  try {
    final result = await _uploadAndPredict(_capturedImage!);

    // Tahmin ekranına geçiş
    if (!mounted) return;
    Navigator.pushNamed(
```

```
      context,
      '/ResultScreen',
      arguments: result, // Örn. {"hasDisease":true, "diseaseName":"Egzama", ...}
    );
  } catch (e) {
    if (mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Hata: $e'),
          backgroundColor: Colors.redAccent,
          behavior: SnackBarBehavior.floating,
        ),
      );
    }
  } finally {
    if (mounted) setState(() => _isLoading = false);
  }
}

// SnackBar gösterimi için yardımcı fonksiyon
void _showSnackBar(String message, {Color color = Colors.redAccent}) {
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
      content: Text(message),
```

```
      backgroundColor: color,
      behavior: SnackBarBehavior.floating,
    ),
  );
}

//-----------------------------
// WIDGET AĞACI
//-----------------------------
@override
Widget build(BuildContext context) {
  final size = MediaQuery.of(context).size;

  return Stack(
    children: [
      Scaffold(
        backgroundColor: Colors.black,
        appBar: AppBar(
          backgroundColor: const Color(0xFF272727),
          elevation: 0,
          centerTitle: true,
          leading: const BackButton(color: Color(0xFFBCBCBC)),
          title: const Text(
            'Fotoğraf Çek',
```

```
    style: TextStyle(color: Color(0xFFBCBCBC), fontSize: 24),
  ),
),
body: Stack(
  children: [
    // Fotoğraf önizlemesi
    Positioned(
      top: size.height * 0.12,
      left: (size.width - size.width * 0.95) / 2,
      child: Container(
        width: size.width * 0.95,
        height: size.width * 0.95,
        decoration: const BoxDecoration(
          color: Color(0xFF383636),
          shape: BoxShape.circle,
        ),
        child: ClipOval(
          child: _capturedImage == null
              ? const Center(
                  child:
                      Icon(Icons.camera_alt, color: Colors.white24, size: 80),
                )
              : Image.file(_capturedImage!,
                  fit: BoxFit.cover,
```

```
            width: double.infinity,
            height: double.infinity,
          ),
    ),
  ),
),

// Alt Menü
Align(
  alignment: Alignment.bottomCenter,
  child: Container(
    height: size.height * 0.17,
    decoration: const BoxDecoration(
      color: Color(0xFF272727),
      borderRadius: BorderRadius.vertical(top: Radius.circular(24)),
    ),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: [
        // Gönder (tahmin) butonu
        IconButton(
          icon: const Icon(Icons.send, color: Colors.white, size: 40),
          onPressed: _isLoading ? null : _handlePredictButton,
        ),
```

```
// Fotoğraf çekme butonu
Opacity(
  opacity: _isLoading ? 0.5 : 1.0,
  child: InkWell(
    onTap: _isLoading ? null : _takePhoto,
    child: Container(
      margin: const EdgeInsets.symmetric(vertical: 5),
      height: 80,
      width: 80,
      decoration: const BoxDecoration(
        color: Color(0xFF4AD5CD),
        shape: BoxShape.circle,
      ),
      child: Center(
        child: Container(
          height: 45,
          width: 45,
          decoration: BoxDecoration(
            shape: BoxShape.circle,
            border: Border.all(color: Colors.white, width: 5),
          ),
        ),
      ),
```

```
    ),
  ),
),

// Silme butonu
IconButton(
  icon: const Icon(Icons.delete, color: Colors.white, size: 40),
  onPressed: _isLoading
      ? null
      : () async {
          if (_capturedImage == null) {
            _showSnackBar('Silinecek bir fotoğraf bulunamadı.', color: Colors.orangeAccent);
          } else {
            // Uzun basınca onay diyaloğu
            final bool? confirm = await showDialog<bool>(
              context: context,
              builder: (context) => AlertDialog(
                title: const Text('Fotoğrafı Sil'),
                content: const Text('Fotoğrafı silmek istediğinize emin misiniz?'),
                actions: [
                  TextButton(
                    onPressed: () => Navigator.pop(context, false),
                    child: const Text('Vazgeç'),
                  ),
```

```
                    TextButton(
                     onPressed: () => Navigator.pop(context, true),
                     child: const Text('Sil'),
                    ),
                   ],
                  ),
                 );
                 if (confirm == true) {
                  setState(() => _capturedImage = null);
                  _showSnackBar('Fotoğraf kaldırıldı.', color: Colors.grey);
                 }
                }
               },
              ),
             ],
            ),
           ),
          ),
         ],
        ),
       ),
      ),

      //-----------------------------------
      // Yükleme göstergesi (tam ekran overlay)
```

```
        //-------------------------------------
        if (_isLoading)
          Container(
            color: Colors.black54,
            child: const Center(child: CircularProgressIndicator()),
          ),
      ],
    );
  }
}


// Galeri.dart
//.....................................................................................................
import 'dart:convert';
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'package:http/http.dart' as http;

class GaleriScanUI extends StatefulWidget {
  const GaleriScanUI({super.key});

  @override
```

```
  State<GaleriScanUI> createState() => _GaleriScanUIState();
}

class _GaleriScanUIState extends State<GaleriScanUI> {
  final ImagePicker _picker = ImagePicker();
  File? _capturedImage;
  bool _isLoading = false;

  Future<void> _pickImageFromGallery() async {
    final XFile? image = await _picker.pickImage(source: ImageSource.gallery);
    if (image != null) {
      setState(() => _capturedImage = File(image.path));
    }
  }

  Future<Map<String, dynamic>> _uploadAndPredict(File image) async {
    final uploadUri = Uri.parse('https://omerfarukcelik.duckdns.org:5001/images');

    final request = http.MultipartRequest('POST', uploadUri)
      ..files.add(await http.MultipartFile.fromPath('file', image.path));

    final uploadResponse = await request.send();

    // 200, 201, 202 gibi başarılı kodlar kabul edilir
```

```
  if (uploadResponse.statusCode < 200 || uploadResponse.statusCode >= 300) {
    final body = await uploadResponse.stream.bytesToString();
    throw Exception('Yükleme başarısız: $body');
  }

  final predictUri = Uri.parse('https://omerfarukcelik.duckdns.org:5001/tahmin');
  final predictResponse = await http.get(predictUri);

  if (predictResponse.statusCode != 200) {
    throw Exception('Tahmin alınamadı: ${predictResponse.statusCode} – ${predictResponse.body}');
  }

  return jsonDecode(predictResponse.body) as Map<String, dynamic>;
}

void _handlePredictButton() async {
  if (_capturedImage == null) {
    _showSnackBar('Lütfen önce bir fotoğraf seçin.');
    return;
  }

  setState(() => _isLoading = true);

  try {
```

```
    final result = await _uploadAndPredict(_capturedImage!);

    if (!mounted) return;
    Navigator.pushNamed(
      context,
      '/ResultScreen',
      arguments: result,
    );
  } catch (e) {
    _showSnackBar('Hata: $e');
  } finally {
    if (mounted) setState(() => _isLoading = false);
  }
}

void _showSnackBar(String message, {Color color = Colors.redAccent}) {
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(
      content: Text(message),
      backgroundColor: color,
      behavior: SnackBarBehavior.floating,
    ),
  );
}
```

```
@override
Widget build(BuildContext context) {
 final size = MediaQuery.of(context).size;

 return Stack(
  children: [
   Scaffold(
    backgroundColor: Colors.black,
    appBar: AppBar(
     backgroundColor: const Color(0xFF272727),
     elevation: 0,
     centerTitle: true,
     leading: const BackButton(color: Color(0xFFBCBCBC)),
     title: const Text(
      "Galeriden Yükle",
      style: TextStyle(color: Color(0xFFBCBCBC), fontSize: 24),
     ),
    ),
    body: Stack(
     children: [
      Positioned(
       top: size.height * 0.12,
       left: (size.width - size.width * 0.95) / 2,
```

```
  child: Container(
   width: size.width * 0.95,
   height: size.width * 0.95,
   decoration: const BoxDecoration(
    color: Color(0xFF383636),
    shape: BoxShape.circle,
   ),
   child: ClipOval(
    child: _capturedImage == null
        ? const Center(
            child: Icon(Icons.photo_library_outlined,
                color: Colors.white24, size: 80),
          )
        : Image.file(
            _capturedImage!,
            fit: BoxFit.cover,
            width: double.infinity,
            height: double.infinity,
          ),
   ),
  ),
 ),
 Align(
  alignment: Alignment.bottomCenter,
```

```
child: Container(
 height: size.height * 0.17,
 decoration: const BoxDecoration(
  color: Color(0xFF272727),
  borderRadius: BorderRadius.vertical(top: Radius.circular(24)),
 ),
 child: Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
   IconButton(
    icon: const Icon(Icons.send, color: Colors.white, size: 40),
    onPressed: _isLoading ? null : _handlePredictButton,
   ),
   Opacity(
    opacity: _isLoading ? 0.5 : 1.0,
    child: InkWell(
     onTap: _isLoading ? null : _pickImageFromGallery,
     child: Container(
      height: 80,
      width: 80,
      decoration: const BoxDecoration(
       color: Color(0xFF4AD5CD),
       shape: BoxShape.circle,
      ),
```

```
      child: Center(
       child: Container(
         height: 45,
         width: 45,
         decoration: BoxDecoration(
           shape: BoxShape.circle,
           border: Border.all(color: Colors.white, width: 5),
         ),
       ),
      ),
    ),
  ),
),
IconButton(
  icon: const Icon(Icons.delete, color: Colors.white, size: 40),
  onPressed: _isLoading
      ? null
      : () {
          if (_capturedImage == null) {
            _showSnackBar('Silinecek bir fotoğraf bulunamadı.',
                color: Colors.orangeAccent);
          } else {
            setState(() => _capturedImage = null);
            _showSnackBar('Fotoğraf kaldırıldı.', color: Colors.grey);
```

```
                        }
                      },
                    ),
                  ],
                ),
              ),
            ),
          ],
        ),
      ),
      if (_isLoading)
        Container(
          color: Colors.black54,
          child: const Center(child: CircularProgressIndicator()),
        ),
    ],
  );
 }
}


// EditProfile.dart
//.....................................................................................................
import 'package:flutter/material.dart';
```

```
class EditProfile extends StatefulWidget {
  const EditProfile({super.key});

  @override
  State<EditProfile> createState() => _EditProfileState();
}

class _EditProfileState extends State<EditProfile> {
  final TextEditingController _nameController = TextEditingController();
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _mpdController = TextEditingController();
  final TextEditingController _dpwController = TextEditingController();
  final TextEditingController _heightController = TextEditingController();
  final TextEditingController _weightController = TextEditingController();

  DateTime birthDate = DateTime.utc(2019, 01, 01);
  TimeOfDay gib = TimeOfDay(hour: 22, minute: 30);
  TimeOfDay wu = TimeOfDay(hour: 06, minute: 30);

  var selectGen = 0;

  @override
  Widget build(BuildContext context) {
```

```
final size = MediaQuery.of(context).size;
return Scaffold(
  appBar: AppBar(
    backgroundColor: Colors.white,
    elevation: 0,
    title: Text("Edit Profile"),
    centerTitle: true,
    leading: IconButton(
      onPressed: () {
                    Navigator.pop(context);

      },
      icon: Icon(Icons.arrow_back, size: 30, weight: 3),
    ),
  ),
  body: SingleChildScrollView(
    child: Container(
      width: double.infinity,
      height: size.height * 1.1,
      color: Colors.white,
      padding: EdgeInsets.symmetric(horizontal: 18),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
```

```
children: [
 Text("About Me",
    style: TextStyle(
       color: Colors.black,
       fontWeight: FontWeight.w600,
       fontSize: 22)),
 Text("Name",
    style: TextStyle(fontSize: 16, fontWeight: FontWeight.w600)),
 Container(
  decoration: BoxDecoration(
   color: Color.fromRGBO(74, 213, 205, 0.1),
  ),
  margin: EdgeInsets.symmetric(vertical: 10),
  padding: EdgeInsets.symmetric(vertical: 7, horizontal: 25),
  child: TextFormField(
   controller: _nameController,
   keyboardType: TextInputType.name,
   decoration: const InputDecoration(
     border: InputBorder.none,
     hintText: "Enter Name",
   ),
  ),
 ),
 SizedBox(height: 2),
```

```
Text("Email",
   style:
      TextStyle(fontSize: 16, fontWeight: FontWeight.w600)),
Container(
 decoration: BoxDecoration(
  color: Color.fromRGBO(74, 213, 205, 0.1),
 ),
 margin: EdgeInsets.symmetric(vertical: 10),
 padding: EdgeInsets.symmetric(vertical: 7, horizontal: 25),
 child: TextFormField(
  enabled: false,
  controller: _emailController,
  keyboardType: TextInputType.emailAddress,
  decoration: const InputDecoration(
     border: InputBorder.none, hintText: "Enter Email"),
 ),
),
SizedBox(height: 2),
Text("Gender",
   style:
      TextStyle(fontSize: 16, fontWeight: FontWeight.w600)),
GenderSelect(),
SizedBox(height: 2),
Text("Date of Birth",
```

```
    style:
        TextStyle(fontSize: 16, fontWeight: FontWeight.w600)),
InkWell(
  onTap: () {},
  child: Container(
    height: 60,
    width: double.infinity,
    decoration: BoxDecoration(
      color: Color.fromRGBO(74, 213, 205, 0.1),
    ),
    margin: EdgeInsets.symmetric(vertical: 10),
    padding: EdgeInsets.symmetric(vertical: 7, horizontal: 25),
    child: Center(
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          Text(
            "DD/MM/YYYY",
            style: TextStyle(
                color: Colors.black, fontSize: 16),
          ),
          Icon(Icons.calendar_month,
              color: Colors.grey, size: 24)
        ],
```

```
        ),
      )),
),
SizedBox(height: 5),
Text("Other",
   style: TextStyle(
       color: Colors.black,
       fontWeight: FontWeight.w600,
       fontSize: 22)),
OtherEdit(size),
SizedBox(height: 5),
InkWell(
  onTap: () {},
  child: Container(
    width: double.infinity,
    height: size.height * 0.07,
    decoration: BoxDecoration(
       color: Color.fromRGBO(74, 213, 205, 1),
       borderRadius: BorderRadius.circular(100)),
    margin: EdgeInsets.symmetric(vertical: 10),
    padding: EdgeInsets.symmetric(vertical: 5, horizontal: 25),
    child: Center(
      child: Text("Save",
         style: TextStyle(
```

```
                color: Colors.white,
                fontWeight: FontWeight.w700,
                fontSize: 22)),
          ),
        ),
      )
    ],
   ),
  ),
 ),
);
}

Widget GenderSelect() {
 return Row(
  children: [
   Expanded(
    child: InkWell(
     onTap: () {},
     child: Container(
      height: 60,
      margin: EdgeInsets.all(5),
      color: Color.fromRGBO(74, 213, 205, 0.1),
      child: Center(
```

```
        child: Text("Male",
            style: TextStyle(color: Colors.black, fontSize: 16)),
      ),
    ),
  ),
),
Expanded(
  child: InkWell(
    onTap: () {},
    child: Container(
      height: 60,
      margin: EdgeInsets.all(5),
      color: Color.fromRGBO(74, 213, 205, 0.1),
      child: Center(
        child: Text("Female",
            style: TextStyle(color: Colors.black, fontSize: 16)),
      ),
    ),
  ),
),
Expanded(
  child: InkWell(
    onTap: () {},
    child: Container(
```

```
          height: 60,
          margin: EdgeInsets.all(5),
          color: Color.fromRGBO(74, 213, 205, 0.1),
          child: Center(
            child: Text("Other",
                style: TextStyle(color: Colors.black, fontSize: 16)),
          ),
        ),
      ),
    ),
  ],
 );
}

Widget OtherEdit(var size) {
 return Container(
   width: double.infinity,
   height: size.height * 0.24,
   child: GridView(
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 2,
        crossAxisSpacing: 2,
        mainAxisSpacing: 2,
        mainAxisExtent: size.height * 0.12),
```

```
children: [
  Container(
    padding: EdgeInsets.symmetric(horizontal: 10, vertical: 10),
    height: size.height * 0.12,
    decoration: BoxDecoration(
       color: Color.fromRGBO(74, 213, 205, 0.1)),
    child: Column(
     mainAxisAlignment: MainAxisAlignment.spaceEvenly,
     crossAxisAlignment: CrossAxisAlignment.start,
     children: [
      Text("Weight(kg)",
          style: TextStyle(
              color: Colors.grey,
              fontSize: 14,
              fontWeight: FontWeight.w600)),
      TextField(
       controller: _weightController,
       style: TextStyle(color: Colors.black, fontSize: 18),
       keyboardType: TextInputType.number,
       decoration: const InputDecoration(
           border: InputBorder.none, hintText: "00"),
      )
     ],
    ),
```

```
),
Container(
 padding: EdgeInsets.symmetric(horizontal: 10, vertical: 10),
 height: size.height * 0.12,
 decoration: BoxDecoration(
    color: Color.fromRGBO(74, 213, 205, 0.1)),
 child: Column(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
   Text("Height(cm)",
       style: TextStyle(
           color: Colors.grey,
           fontSize: 14,
           fontWeight: FontWeight.w600)),
   TextField(
    controller: _heightController,
    style: TextStyle(color: Colors.black, fontSize: 18),
    keyboardType: TextInputType.number,
    decoration: const InputDecoration(
        border: InputBorder.none, hintText: "00"),
   )
  ],
 ),
```

```
),
Container(
 padding: EdgeInsets.symmetric(horizontal: 10, vertical: 10),
 height: size.height * 0.12,
 decoration: BoxDecoration(
    color: Color.fromRGBO(74, 213, 205, 0.1)),
 child: Column(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
   Text("Medicine per day",
       style: TextStyle(
           color: Colors.grey,
           fontSize: 14,
           fontWeight: FontWeight.w600)),
   TextField(
    controller: _mpdController,
    style: TextStyle(color: Colors.black, fontSize: 18),
    keyboardType: TextInputType.number,
    decoration: const InputDecoration(
        border: InputBorder.none, hintText: "00"),
   )
  ],
 ),
```

```
),
Container(
 padding: EdgeInsets.symmetric(horizontal: 10, vertical: 10),
 height: size.height * 0.12,
 decoration: BoxDecoration(
    color: Color.fromRGBO(74, 213, 205, 0.1)),
 child: Column(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
   Text("Diagnosis per week",
       style: TextStyle(
           color: Colors.grey,
           fontSize: 14,
           fontWeight: FontWeight.w600)),
    TextField(
     controller: _dpwController,
     style: TextStyle(color: Colors.black, fontSize: 18),
     keyboardType: TextInputType.number,
     decoration: const InputDecoration(
         border: InputBorder.none, hintText: "00"),
    )
  ],
 ),
```

```
),
InkWell(
 onTap: () {},
 child: Container(
  padding: EdgeInsets.symmetric(horizontal: 10, vertical: 10),
  height: size.height * 0.12,
  decoration: BoxDecoration(
    color: Color.fromRGBO(74, 213, 205, 0.1)),
  child: Column(
   mainAxisAlignment: MainAxisAlignment.spaceEvenly,
   crossAxisAlignment: CrossAxisAlignment.start,
   children: [
    Text("Get in bed",
      style: TextStyle(
        color: Colors.grey,
        fontSize: 14,
        fontWeight: FontWeight.w600)),
    Text("22:30",
      style: TextStyle(color: Colors.black, fontSize: 18)),
   ],
  ),
 ),
),
InkWell(
```

```
        onTap: () {},
        child: Container(
         padding: EdgeInsets.symmetric(horizontal: 10, vertical: 10),
         height: size.height * 0.12,
         decoration: BoxDecoration(
            color: Color.fromRGBO(74, 213, 205, 0.1)),
         child: Column(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
           Text("Wake up",
              style: TextStyle(
                 color: Colors.grey,
                 fontSize: 14,
                 fontWeight: FontWeight.w600)),
            Text("06:30",
              style: TextStyle(color: Colors.black, fontSize: 18)),
          ],
         ),
        ),
       ],
      ),
    );
```

```
  }
}

//.....................................................................................
// Home.dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';

class HomePage extends StatelessWidget {
  const HomePage({super.key});

  @override
  Widget build(BuildContext context) {
    final user = FirebaseAuth.instance.currentUser;

    return Scaffold(
      extendBodyBehindAppBar: true,
      appBar: AppBar(
        backgroundColor: Colors.transparent,
        elevation: 0,
        title: const Text('Hoş geldiniz'),
        centerTitle: true,
        actions: [
          IconButton(
```

```
      tooltip: 'Çıkış Yap',
      icon: const Icon(Icons.logout_rounded),
      onPressed: () async {
        await FirebaseAuth.instance.signOut();
        if (context.mounted) {
          Navigator.pushNamedAndRemoveUntil(
            context,
            '/LogIn',
            (_) => false,
          );
        }
      },
    ),
  ],
),
body: SafeArea(
  child: Container(
    decoration: const BoxDecoration(
      gradient: LinearGradient(
        colors: [Color(0xFFEBFDFC), Color(0xFFF8FFFF)],
        begin: Alignment.topCenter,
        end: Alignment.bottomCenter,
      ),
    ),
```

```
padding: const EdgeInsets.all(24),
child: Column(
 children: [
  _AccountCard(user: user),
  const SizedBox(height: 40),
  const Text(
   'Hoş geldiniz!\nCilt hastalığı tespiti yapmaya başlayın.',
   textAlign: TextAlign.center,
   style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
  ),
  const Spacer(),
  CustomActionButton(
   label: 'Kamerayı Aç',
   icon: Icons.camera_alt_outlined,
   onPressed: () => Navigator.pushNamed(context, '/CameraScanUI'),
  ),
  const SizedBox(height: 14),
  CustomActionButton(
   label: 'Galeriden Fotoğraf Seç',
   icon: Icons.photo_library_outlined,
   isPrimary: false,
   onPressed: () => Navigator.pushNamed(context, '/GaleriScanUI'),
  ),
  const Spacer(flex: 2),
```

```
          ],
        ),
      ),
    ),
  );
 }
}

/*——— Alt bileşenler ———*/

class _AccountCard extends StatelessWidget {
 const _AccountCard({required this.user});
 final User? user;

 @override
 Widget build(BuildContext context) {
  return Card(
   shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(20)),
   elevation: 4,
   child: ListTile(
    contentPadding:
      const EdgeInsets.symmetric(horizontal: 20, vertical: 8),
    leading: CircleAvatar(
     radius: 28,
```

```
  backgroundImage:
      user?.photoURL != null ? NetworkImage(user!.photoURL!) : null,
  child: user?.photoURL == null
      ? const Icon(Icons.person_outline, size: 32)
      : null,
),
            title: Text(
        (user?.displayName != null && user!.displayName!.trim().isNotEmpty)
            ? user!.displayName!
            : 'Kullanıcı',
        style: const TextStyle(fontWeight: FontWeight.w600, fontSize: 18),
      ),
subtitle: Row(
  children: [
    const Icon(Icons.email_outlined, size: 16, color: Colors.grey),
    const SizedBox(width: 4),
    Expanded(
      child: Text(
        user?.email ?? '',
        style: const TextStyle(color: Colors.grey),
        overflow: TextOverflow.ellipsis,
      ),
    ),
  ],
```

```
        ),
        trailing: IconButton(
          tooltip: 'Profili Gör',
          icon: const Icon(Icons.edit, color: Color(0xFF4AD5CD)),
          onPressed: () => Navigator.pushNamed(context, '/UserProfile'),
        ),
      ),
    );
  }
}

class CustomActionButton extends StatelessWidget {
  final String label;
  final IconData icon;
  final VoidCallback onPressed;
  final bool isPrimary;

  const CustomActionButton({
    required this.label,
    required this.icon,
    required this.onPressed,
    this.isPrimary = true,
  });
```

```
@override
Widget build(BuildContext context) {
 final style = isPrimary
    ? ElevatedButton.styleFrom(
       backgroundColor: const Color(0xFF4AD5CD),
       minimumSize: const Size(double.infinity, 58),
       shape: RoundedRectangleBorder(
         borderRadius: BorderRadius.circular(14),
       ),
      )
    : OutlinedButton.styleFrom(
       minimumSize: const Size(double.infinity, 58),
       side: const BorderSide(color: Color(0xFF4AD5CD), width: 2),
       shape: RoundedRectangleBorder(
         borderRadius: BorderRadius.circular(14),
       ),
      );

 return isPrimary
    ? ElevatedButton.icon(
       onPressed: onPressed,
       icon: Icon(icon, size: 28),
       label: Text(label, style: const TextStyle(fontSize: 18)),
       style: style,
```

```
        )
      : OutlinedButton.icon(
          onPressed: onPressed,
          icon: Icon(icon, size: 28),
          label: Text(label, style: const TextStyle(fontSize: 18)),
          style: style,
        );
  }
}
//.....................................................................................

// Login.dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';

class LogIn extends StatefulWidget {
  const LogIn({super.key});

  @override
  State<LogIn> createState() => _LogInState();
}

class _LogInState extends State<LogIn> {
  final _formKey  = GlobalKey<FormState>();
```

```
final _emailCtrl = TextEditingController();
final _passCtrl  = TextEditingController();

bool _loading     = false;
bool _obscurePass = true;



/*———————— AUTH ————————*/
Future<void> _signIn() async {
  if (!_formKey.currentState!.validate()) return;
  setState(() => _loading = true);

  try {
    await FirebaseAuth.instance.signInWithEmailAndPassword(
      email: _emailCtrl.text.trim(),
      password: _passCtrl.text.trim(),
    );
    _showSnack('Login successful!');
    // Giriş başarılı → Home sayfasına
    Navigator.pushReplacementNamed(context, '/HomePage');
    _emailCtrl.clear();
    _passCtrl.clear();
  } on FirebaseAuthException catch (e) {
```

```
    final msg = switch (e.code) {
      'user-not-found'      => 'No user found for that email.',
      'wrong-password'      => 'Wrong password.',
      'invalid-credential'  => 'Invalid credentials.',
      _                     => 'Error: ${e.message}',
    };
    _showSnack(msg);
  } finally {
    if (mounted) setState(() => _loading = false);
  }
}

Future<void> _resetPassword() async {
  final email = _emailCtrl.text.trim();
  if (email.isEmpty) {
    _showSnack('Please enter your email first.');
    return;
  }
  try {
    await FirebaseAuth.instance.sendPasswordResetEmail(email: email);
    _showSnack('Password reset link sent.');
  } catch (e) {
    _showSnack('Failed: $e');
  }
```

```
}

void _showSnack(String msg) =>
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text(msg)));



 @override
void dispose() {
 _emailCtrl.dispose();
 _passCtrl.dispose();
 super.dispose();
}


/*————————— UI —————————*/
@override
Widget build(BuildContext context) {
 final size = MediaQuery.of(context).size;

 return Scaffold(
   body: SafeArea(
     child: SingleChildScrollView(
       padding: EdgeInsets.only(top: size.height * .12),
```

```
        child: Column(
          children: [
            _buildTitleSection(),
            _buildFormSection(),
            _buildForgotPassword(),
            _buildLoginButton(),
            const _OrDivider(),
            _buildGoogleButton(),
            _buildSignUpRedirect(),
          ],
        ),
      ),
    ),
  );
}

/*————————— Widgets ————————*/
Widget _buildTitleSection() => Column(
      children: const [
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text('Cilt ',
                style: TextStyle(
```

```
                fontSize: 50, fontWeight: FontWeight.w900,
                color: Color(0xFF132346))),
          Text('hastalığı',
              style: TextStyle(
                fontSize: 50, fontWeight: FontWeight.w900,
                color: Color(0xFF4AD5CD))),
        ],
      ),
      SizedBox(height: 40),
      Text('Lütfen e-posta ve şifrenizi yazınız.',
          style: TextStyle(fontSize: 18, fontWeight: FontWeight.w200)),
      SizedBox(height: 30),
    ],
  );

Widget _buildFormSection() => Padding(
    padding: const EdgeInsets.symmetric(horizontal: 30),
    child: Form(
      key: _formKey,
      child: Column(
        children: [
          _inputField(
            controller: _emailCtrl,
            hint: 'Enter Email',
```

```
            keyboard: TextInputType.emailAddress,
            isPassword: false,
            validator: (v) =>
                v == null || v.isEmpty || !v.contains('@')
                    ? 'Valid email required' : null,
          ),
          _inputField(
            controller: _passCtrl,
            hint: 'Enter Password',
            keyboard: TextInputType.text,
            isPassword: true,
            validator: (v) =>
                v == null || v.isEmpty ? 'Password required' : null,
          ),
        ],
      ),
    ),
  );

Widget _inputField({
 required TextEditingController controller,
 required String hint,
 required TextInputType keyboard,
 required bool isPassword,
```

```
  required String? Function(String?) validator,
}) =>
    Container(
      margin: const EdgeInsets.symmetric(vertical: 10),
      padding: const EdgeInsets.symmetric(vertical: 7, horizontal: 10),
      decoration: BoxDecoration(
        color: Colors.white.withOpacity(.8),
        borderRadius: BorderRadius.circular(12),
        boxShadow: [
          BoxShadow(
            color: Colors.black.withOpacity(.2),
            blurRadius: 6,
            offset: const Offset(0, 2),
          ),
        ],
      ),
      child: TextFormField(
        controller: controller,
        obscureText: isPassword ? _obscurePass : false,
        keyboardType: keyboard,
        validator: validator,
        decoration: InputDecoration(
          border: InputBorder.none,
          hintText: hint,
```

```
        prefixIcon: Icon(
          isPassword ? Icons.lock : Icons.email,
          color: const Color(0xFF4AD5CD),
        ),
        suffixIcon: isPassword
            ? IconButton(
                icon: Icon(
                  _obscurePass ? Icons.visibility_off : Icons.visibility,
                  color: const Color(0xFF4AD5CD),
                ),
                onPressed: () =>
                    setState(() => _obscurePass = !_obscurePass),
              )
            : null,
      ),
    ),
  );

Widget _buildForgotPassword() => TextButton(
    onPressed: _resetPassword,
    child: const Text('Forget Password',
      style: TextStyle(color: Colors.grey, fontSize: 16)),
  );
```

```
Widget _buildLoginButton() => Padding(
    padding: const EdgeInsets.symmetric(horizontal: 30),
    child: ElevatedButton(
     onPressed: _loading ? null : _signIn,
     style: ElevatedButton.styleFrom(
      backgroundColor: const Color(0xFF4AD5CD),
      shape: RoundedRectangleBorder(
       borderRadius: BorderRadius.circular(100),
      ),
      padding: const EdgeInsets.symmetric(vertical: 15, horizontal: 25),
      minimumSize: const Size(double.infinity, 50),
     ),
     child: _loading
       ? const CircularProgressIndicator(color: Colors.white)
       : const Text('Log In',
          style: TextStyle(
            color: Colors.white,
            fontWeight: FontWeight.w700,
            fontSize: 22)),
    ),
  );

Widget _buildGoogleButton() => Container(
    margin: const EdgeInsets.symmetric(vertical: 10, horizontal: 30),
```

```
decoration: BoxDecoration(
  color: Colors.white,
  borderRadius: BorderRadius.circular(100),
  boxShadow: [
    BoxShadow(
      color: Colors.black.withOpacity(.1),
      blurRadius: 6,
      offset: const Offset(0, 2),
    ),
  ],
),
child: InkWell(
  borderRadius: BorderRadius.circular(100),
  onTap: () {
    // Google ile giriş işlemi burada yapılacak
    _showSnack('Google ile giriş yakında!');
  },
  child: Padding(
    padding: const EdgeInsets.symmetric(vertical: 12, horizontal: 25),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Image.asset('assets/images/google_symbol.png',
            height: 24, width: 24),
```

```
            const SizedBox(width: 10),
            const Text('Continue with Google',
              style: TextStyle(
                color: Colors.black54,
                fontWeight: FontWeight.w600,
                fontSize: 16)),
          ],
        ),
      ),
    ),
  );

Widget _buildSignUpRedirect() => Padding(
    padding: const EdgeInsets.symmetric(vertical: 15),
    child: Row(mainAxisAlignment: MainAxisAlignment.center, children: [
      const Text("Don't have an account?",
        style: TextStyle(fontSize: 16, fontWeight: FontWeight.w200)),
      GestureDetector(
        onTap: () => Navigator.pushReplacementNamed(context, '/Register'),
        child: const Text(' Sign Up',
          style: TextStyle(
            fontSize: 16,
            fontWeight: FontWeight.w200,
            color: Color(0xFF4AD5CD))),
```

```
        ),
      ]),
    );
}

/*————————— Bölücü ——————————*/
class _OrDivider extends StatelessWidget {
  const _OrDivider();

  @override
  Widget build(BuildContext context) => Padding(
      padding: const EdgeInsets.symmetric(vertical: 10),
      child: Row(children: const [
        Expanded(child: Divider(color: Colors.grey, thickness: .5, indent: 40, endIndent: 10)),
        Text('OR',
           style: TextStyle(color: Colors.grey, fontWeight: FontWeight.bold)),
        Expanded(child: Divider(color: Colors.grey, thickness: .5, indent: 10, endIndent: 40)),
      ]));
}

//.....................................................................................

// MyProfile.dart
import 'package:flutter/material.dart';
```

```
import 'package:firebase_auth/firebase_auth.dart';

class UserProfile extends StatefulWidget {
  const UserProfile({super.key});

  @override
  State<UserProfile> createState() => _UserProfileState();
}

class _UserProfileState extends State<UserProfile> {
  final User? user = FirebaseAuth.instance.currentUser;

  String aboutMe = '';
  String phone = '';
  String address = '';

  void _openEditModal() {
    final aboutController = TextEditingController(text: aboutMe);
    final phoneController = TextEditingController(text: phone);
    final addressController = TextEditingController(text: address);

    showModalBottomSheet(
      context: context,
      isScrollControlled: true,
```

```
shape: const RoundedRectangleBorder(
  borderRadius: BorderRadius.vertical(top: Radius.circular(20)),
),
builder: (context) => Padding(
  padding: MediaQuery.of(context).viewInsets,
  child: Padding(
    padding: const EdgeInsets.all(16),
    child: Column(
      mainAxisSize: MainAxisSize.min,
      children: [
        const Text("Profili Düzenle", style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold)),
        const SizedBox(height: 10),
        TextField(
          controller: aboutController,
          maxLines: 2,
          decoration: const InputDecoration(labelText: "Hakkımda", border: OutlineInputBorder()),
        ),
        const SizedBox(height: 10),
        TextField(
          controller: phoneController,
          keyboardType: TextInputType.phone,
          decoration: const InputDecoration(labelText: "Telefon", border: OutlineInputBorder()),
        ),
        const SizedBox(height: 10),
```

```
TextField(
 controller: addressController,
 decoration: const InputDecoration(labelText: "Adres", border: OutlineInputBorder()),
),
const SizedBox(height: 20),
Align(
 alignment: Alignment.centerRight,
 child: ElevatedButton(
  onPressed: () {
   if (aboutController.text.trim().isEmpty ||
     phoneController.text.trim().isEmpty ||
     addressController.text.trim().isEmpty) {
    return;
   }

   setState(() {
    aboutMe = aboutController.text.trim();
    phone = phoneController.text.trim();
    address = addressController.text.trim();
   });

   Navigator.pop(context);

   // Temizleme işlemi
```

```
                aboutController.clear();
                phoneController.clear();
                addressController.clear();
              },
              style: ElevatedButton.styleFrom(
                backgroundColor: const Color(0xFF4AD5CD),
              ),
              child: const Text("Kaydet"),
            ),
          ),
        ],
      ),
    ),
  ),
 );
}

@override
Widget build(BuildContext context) {
 final displayName = user?.displayName ?? 'Kullanıcı';
 final email = user?.email ?? '';
 final photoURL = user?.photoURL;

 return Scaffold(
```

```
appBar: AppBar(
  backgroundColor: Colors.white,
  elevation: 0,
  toolbarHeight: 30,
  leading: IconButton(
    onPressed: () => Navigator.pop(context),
    icon: const Icon(Icons.arrow_back, size: 30, weight: 3),
  ),
),
body: SingleChildScrollView(
  child: Container(
    color: Colors.white,
    width: double.infinity,
    padding: const EdgeInsets.all(18),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            const Text("My Profile", style: TextStyle(fontSize: 26, fontWeight: FontWeight.bold)),
            InkWell(
              onTap: _openEditModal,
              child: Container(
```

```
        padding: const EdgeInsets.symmetric(vertical: 5, horizontal: 15),
        decoration: BoxDecoration(
          color: const Color(0xFF4AD5CD),
          borderRadius: BorderRadius.circular(10),
        ),
        child: const Row(
          children: [
            Text("Edit ", style: TextStyle(color: Colors.white, fontSize: 18)),
            Icon(Icons.edit, color: Colors.white, size: 20),
          ],
        ),
      ),
    )
  ],
),
const SizedBox(height: 20),
Column(
  children: [
    Center(
      child: CircleAvatar(
        radius: 50,
        backgroundImage: photoURL != null ? NetworkImage(photoURL) : null,
        child: photoURL == null
            ? const Icon(Icons.person_outline, size: 50)
```

```
                    : null,

                ),

              ),

              const SizedBox(height: 10),

              Text(displayName, style: const TextStyle(fontSize: 20, fontWeight: FontWeight.w600)),

              Text(email, style: const TextStyle(fontSize: 16, fontWeight: FontWeight.w500, color:
Colors.blueGrey)),

            ],

          ),

          const SizedBox(height: 30),

          const Text("About Me", style: TextStyle(color: Colors.black, fontWeight: FontWeight.w600,
fontSize: 22)),

          UserProfileWidget(opt: "Hakkımda", val: aboutMe.isNotEmpty ? aboutMe : "Henüz bir şey
yazılmamış."),

          const SizedBox(height: 10),

          const Text("İletişim", style: TextStyle(color: Colors.black, fontWeight: FontWeight.w600, fontSize:
22)),

          UserProfileWidget(opt: "Telefon", val: phone.isNotEmpty ? phone : "Belirtilmemiş"),

          UserProfileWidget(opt: "Adres", val: address.isNotEmpty ? address : "Belirtilmemiş"),

        ],

      ),

    ),

  );

 }

}
```

```
class UserProfileWidget extends StatelessWidget {
 final String opt;
 final String val;

 const UserProfileWidget({required this.opt, required this.val});

 @override
 Widget build(BuildContext context) {
  return Container(
   margin: const EdgeInsets.symmetric(vertical: 5),
   child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
     Text(opt, style: const TextStyle(fontSize: 16, color: Colors.grey)),
     const SizedBox(height: 5),
     Text(val, style: const TextStyle(fontSize: 18, fontWeight: FontWeight.w500)),
     const Divider(thickness: 1),
    ],
   ),
  );
 }
}
```

```
.....................................................................................

// Register.dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

class Register extends StatefulWidget {
  const Register({super.key});
  @override
  State<Register> createState() => _RegisterState();
}

class _RegisterState extends State<Register> {
  final _formKey = GlobalKey<FormState>();
  final _nameCtrl = TextEditingController();
  final _emailCtrl = TextEditingController();
  final _passCtrl = TextEditingController();
  final _confirmCtrl = TextEditingController();

  bool _loading = false;

  // Şifre göster/gizle için bool değişkenler
  bool _obscurePass = true;
  bool _obscureConfirm = true;
```

```
// Güçlü şifre regex
final _passwordPattern =
    RegExp(r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[!@#\$&*~]).{8,}$');

Future<void> register() async {
  if (!_formKey.currentState!.validate()) return;

  if (_passCtrl.text.trim() != _confirmCtrl.text.trim()) {
    _showSnack('Şifreler uyuşmuyor');
    return;
  }

  setState(() => _loading = true);
  try {
    await FirebaseAuth.instance.createUserWithEmailAndPassword(
      email: _emailCtrl.text.trim(),
      password: _passCtrl.text.trim(),
    );

    _nameCtrl.clear();
    _emailCtrl.clear();
    _passCtrl.clear();
    _confirmCtrl.clear();
```

```
    _showSnack('Kayıt başarılı!');
  } on FirebaseAuthException catch (e) {
    if (e.code == 'weak-password') {
      _showSnack('Şifre çok zayıf.');
    } else if (e.code == 'email-already-in-use') {
      _showSnack('Bu e-posta zaten kayıtlı.');
    } else {
      _showSnack('Hata: ${e.code}');
    }
  } catch (e) {
    _showSnack('Bilinmeyen hata: $e');
  } finally {
    setState(() => _loading = false);
  }
}

@override
Widget build(BuildContext context) => Scaffold(
      body: LayoutBuilder(
        builder: (context, constraints) => SingleChildScrollView(
          child: ConstrainedBox(
            constraints: BoxConstraints(minHeight: constraints.maxHeight),
            child: SafeArea(
```

```
            child: Column(
              children: [
                const SizedBox(height: 40),
                _buildLogoSection(),
                _buildFormSection(constraints),
              ],
            ),
          ),
        ),
      ),
    );

// — Logo bölümü
Widget _buildLogoSection() => Column(
    children: [
      Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: const [
          Text('Cilt ',
              style: TextStyle(
                  fontSize: 50,
                  fontWeight: FontWeight.w900,
                  color: Color(0xFF132346))),
```

```
          Text('hastalığı',
            style: TextStyle(
              fontSize: 50,
              fontWeight: FontWeight.w900,
              color: Color(0xFF4AD5CD))),
        ],
      ),
      const SizedBox(height: 20),
      const Text('Lütfen aşağıdaki bilgileri doldur.',
        style: TextStyle(fontSize: 18, fontWeight: FontWeight.w200)),
    ],
  );

// — Form bölümü
Widget _buildFormSection(BoxConstraints constraints) => Container(
    width: double.infinity,
    padding: const EdgeInsets.symmetric(horizontal: 30, vertical: 30),
    child: Form(
      key: _formKey,
      child: Column(
        children: [
          _field('Enter your full name', _nameCtrl, TextInputType.name),
          _field('Enter Email', _emailCtrl, TextInputType.emailAddress,
              email: true),
```

```
          _field('Enter Password', _passCtrl, TextInputType.text,
              isPassword: true),
          _field('Confirm Password', _confirmCtrl, TextInputType.text,
              isPassword: true, confirm: true),
          const SizedBox(height: 30),
          _registerButton(),
          _loginPrompt(),
          SizedBox(height: constraints.maxHeight * 0.1),
        ],
      ),
    ),
  );

// — Tek satır input widget'ı
Widget _field(String hint, TextEditingController ctrl, TextInputType type,
    {bool isPassword = false, bool email = false, bool confirm = false}) {
  Icon? icon;

  if (email) {
    icon = const Icon(Icons.email, color: Color.fromRGBO(74, 213, 205, 1));
  } else if (isPassword) {
    icon = const Icon(Icons.lock, color: Color.fromRGBO(74, 213, 205, 1));
  } else if (confirm) {
    icon = const Icon(Icons.lock_outline, color: Color.fromRGBO(74, 213, 205, 1));
```

```
} else {
  icon = const Icon(Icons.person, color: Color.fromRGBO(74, 213, 205, 1));
}

bool obscureText = false;
if (isPassword) {
  obscureText = _obscurePass;
} else if (confirm) {
  obscureText = _obscureConfirm;
}

return Container(
  decoration: BoxDecoration(
    color: const Color.fromRGBO(74, 213, 205, .1),
    borderRadius: BorderRadius.circular(100),
  ),
  margin: const EdgeInsets.symmetric(vertical: 10),
  padding: const EdgeInsets.symmetric(vertical: 7, horizontal: 25),
  child: TextFormField(
    controller: ctrl,
    obscureText: obscureText,
    keyboardType: type,
    decoration: InputDecoration(
      border: InputBorder.none,
```

```
 hintText: hint,
 prefixIcon: icon,
 suffixIcon: (isPassword || confirm)
     ? IconButton(
         icon: Icon(
           obscureText ? Icons.visibility_off : Icons.visibility,
           color: const Color.fromRGBO(74, 213, 205, 1),
         ),
         onPressed: () {
           setState(() {
             if (isPassword) {
               _obscurePass = !_obscurePass;
             } else if (confirm) {
               _obscureConfirm = !_obscureConfirm;
             }
           });
         },
       )
     : null,
),
validator: (value) {
 if (value == null || value.trim().isEmpty) {
   return 'Bu alan boş bırakılamaz';
 }
```

```
        if (email && !value.contains('@')) {
          return 'Geçerli bir e-posta girin';
        }
        if (isPassword && !_passwordPattern.hasMatch(value)) {
          return 'Güvenliğiniz için biraz \ndaha güçlü bir şifre seçin:\n'
              'Büyük harf, küçük harf,\n rakam ve özel karakter kullanın.';
        }
        if (confirm && value != _passCtrl.text) {
          return 'Şifreler uyuşmuyor';
        }
        return null;
      },
    ),
  );
}

// — Kayıt butonu
Widget _registerButton() => SizedBox(
      width: double.infinity,
      child: ElevatedButton(
        onPressed: _loading ? null : register,
        style: ElevatedButton.styleFrom(
          backgroundColor: const Color(0xFF4AD5CD),
          padding: const EdgeInsets.symmetric(vertical: 15),
```

```
        shape:
            RoundedRectangleBorder(borderRadius: BorderRadius.circular(100)),
      ),
      child: _loading
          ? const SizedBox(
              width: 20,
              height: 20,
              child:
                  CircularProgressIndicator(strokeWidth: 3, color: Colors.white),
            )
          : const Text('Register',
              style: TextStyle(
                  color: Colors.white,
                  fontSize: 22,
                  fontWeight: FontWeight.w700)),
    ),
  );

// — Giriş sayfasına yönlendirme
Widget _loginPrompt() => Padding(
    padding: const EdgeInsets.symmetric(vertical: 15),
    child: Row(mainAxisAlignment: MainAxisAlignment.center, children: [
      const Text('Already have an account?',
          style: TextStyle(fontSize: 16, fontWeight: FontWeight.w200)),
```

```
        GestureDetector(
         onTap: () {
           Navigator.pushReplacementNamed(context, '/LogIn');
         },
         child: const Text(' Sign In',
            style: TextStyle(
                fontSize: 16,
                fontWeight: FontWeight.w200,
                color: Color(0xFF4AD5CD))),
       ),
     ]),
   );

void _showSnack(String msg) =>
   ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text(msg)));

@override
void dispose() {
 _nameCtrl.dispose();
 _emailCtrl.dispose();
 _passCtrl.dispose();
 _confirmCtrl.dispose();
 super.dispose();
}
```

```
}

//.....................................................................................................

// Result.dart
import 'package:flutter/material.dart';

class ResultScreen extends StatelessWidget {
  final bool hasDisease;
  final String? diseaseName;
  final List<String> suggestions;

  const ResultScreen({
    super.key,
    required this.hasDisease,
    this.diseaseName,
    required this.suggestions,
  });

  @override
  Widget build(BuildContext context) {
    final Color themeColor = hasDisease ? Colors.redAccent : Colors.greenAccent;
    final String titleText = hasDisease ? "Hastalık Tespit Edildi" : "Cildiniz Sağlıklı";
    final IconData iconData = hasDisease ? Icons.warning_amber_rounded : Icons.verified;
```

```
final String statusText = hasDisease
    ? "Maalesef, sistem '${diseaseName ?? 'bir cilt hastalığı'}' tespit etti."
    : "Tebrikler! Cildiniz sağlıklı görünüyor ✨";

// Eğer hastalık yoksa gösterilecek sabit öneriler:
final List<String> healthyTips = [
  "Günde en az 2 litre su içmeyi unutmayın.",
  "Cildinizi her gün nemlendirin.",
  "Güneş koruyucu kullanmak cildinizi korur.",
  "Dengeli beslenmek cilt sağlığına katkı sağlar.",
  "Mutlu bir ruh hali, sağlıklı bir cilt demektir 🤍"
];

return Scaffold(
  backgroundColor: const Color(0xFF121212),
  appBar: AppBar(
    backgroundColor: Colors.transparent,
    elevation: 0,
    leading: BackButton(color: themeColor),
    title: Text(
      "Tarama Sonucu",
      style: TextStyle(color: themeColor, fontSize: 20),
    ),
    centerTitle: true,
```

```
),
body: Padding(
 padding: const EdgeInsets.symmetric(horizontal: 24, vertical: 16),
 child: Column(
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
   // Durum İkonu
   Container(
    padding: const EdgeInsets.all(28),
    decoration: BoxDecoration(
     shape: BoxShape.circle,
     gradient: RadialGradient(
      colors: [themeColor.withOpacity(0.7), Colors.black],
      radius: 0.85,
     ),
    ),
    child: Icon(iconData, size: 90, color: Colors.white),
   ),
   const SizedBox(height: 24),

   // Başlık
   Text(
    titleText,
    style: TextStyle(
```

```
    fontSize: 24,
    fontWeight: FontWeight.bold,
    color: themeColor,
  ),
  textAlign: TextAlign.center,
),

const SizedBox(height: 16),

// Açıklama
Text(
  statusText,
  style: const TextStyle(fontSize: 16, color: Colors.white70),
  textAlign: TextAlign.center,
),

const SizedBox(height: 28),

// Öneri listesi
Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: [
    Text(
      hasDisease ? "Dikkat Edilmesi Gerekenler" : "Cilt Sağlığı İçin Öneriler",
```

```
  style: TextStyle(
    fontSize: 18,
    fontWeight: FontWeight.w600,
    color: themeColor,
  ),
),
const SizedBox(height: 12),
...List.generate(
  (hasDisease ? suggestions : healthyTips).length,
  (index) {
    final item = hasDisease ? suggestions[index] : healthyTips[index];
    return Row(
      children: [
        Icon(Icons.circle, size: 8, color: themeColor),
        const SizedBox(width: 8),
        Expanded(
          child: Text(
            item,
            style: const TextStyle(fontSize: 15, color: Colors.white70),
          ),
        ),
      ],
    );
  },
```

```
    ),
  ],
),

const Spacer(),

// Ana sayfaya dön butonu
SizedBox(
  width: double.infinity,
  child: ElevatedButton(
    style: ElevatedButton.styleFrom(
      backgroundColor: themeColor,
      padding: const EdgeInsets.symmetric(vertical: 14),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(14),
      ),
    ),
    onPressed: () {
      Navigator.pop(context);
      Navigator.pop(context);
    },
    child: const Text(
      "Ana Sayfaya Dön",
      style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
```

```
            ),
          ),
        ),
      ],
    ),
  ),
 );
 }
}

//.....
// disease_data.dart
const List<String> diseaseNames = [
 'Egzama',
 'Sedef',
 'Alerji',
 'Akne',
 'Rosacea',
 'Dermatit',
 'Bakteriyel Enfeksiyon',
 'Viral Enfeksiyon',
 'Fungal Enfeksiyon',
 'Parazit Enfeksiyonu',
 'İltihaplı Hastalık',
```

'Bağışıklık Sistemi Bozuklukları',

'Kanser',

'Diabetes Mellitus',

'Hipertansiyon',

'Astım',

'Kronik Obstrüktif Akciğer Hastalığı',

'Pulmoner Fibrozis',

'Akciğer Kanseri',

'Mide Kanseri',

'Bağırsak Kanseri',

'Karaciğer Kanseri',

'Pankreas Kanseri',

'Böbrek Kanseri',

'Mesane Kanseri',

'Prostat Kanseri',

'Cilt Kanseri',

'Lenfoma',

'Multiple Myeloma',

'Leukemi',

'Hodgkin Dışı Lenfoma',

'Akut Lenfoblastik Lösemi',

'Kronik Lenfositik Lösemi',

'Akut Miyeloid Lösemi',

'Kronik Miyeloid Lösemi',

'Eritrositler',

'Lökositler',

'Trombositler',

'Anemi',

'Polisitemi',

'Trombositopeni',

'Lökopeni',

'Lenfadenopati',

'Splenomegali',

'Hepatomegali',

'Kilo Kaybı',

'Ateş',

'Gece Terlemeleri',

'Yorgunluk',

'İştah Kaybı',

'Mide Bulantısı',

'Kusma',

'Karın Ağrısı',

'Kabızlık',

'İshal',

'Deri Döküntüleri',

'Kaşıntı',

'Ağız Yaraları',

'Boğaz Ağrısı',

'Öksürük',

'Nefes Darlığı',

'Göğüs Ağrısı',

'Bel Ağrısı',

'Eklem Ağrıları',

'Kas Ağrıları',

'Yara İyileşmesi',

'Saç Dökülmesi',

'Tırnak Değişiklikleri',

'Cilt Rengi Değişiklikleri',

'Göz Rengi Değişiklikleri',

'Ağız Kokuşması',

'Vücut Kokuşması',

'Kilo Alma',

'Şişkinlik',

'Gaz',

'Reflü',

'Yutkunma Güçlüğü',

'Ses Değişiklikleri',

'Kulak Çınlaması',

'Baş Dönmesi',

'Sersemlik',

'Bayılma',

'Nöbetler',

'Kasılmalar',

'Titreme',

'Huzursuzluk',

'Uykusuzluk',

'Depresyon',

'Anksiyete',

'Panik Atak',

'Stres',

'Bipolar Bozukluk',

'Şizofreni',

'Obsesif Kompulsif Bozukluk',

'Yeme Bozuklukları',

'Cinsel Fonksiyon Bozuklukları',

'Bağımlılıklar',

'İnkontinans',

'Fobiler',

'Travma Sonrası Stres Bozukluğu',

'Kişilik Bozuklukları',

'Zihin Bulanıklığı',

'Konsantrasyon Güçlüğü',

'Unutkanlık',

'Hafıza Kaybı',

'Zihin Okuma',

'Paranoia',

'Sosyal İzolasyon',

'İletişim Güçlüğü',

'Empati Eksikliği',

'Öfke Kontrol Sorunları',

'Davranışsal Bağımlılıklar',

'Tekrar Eden Davranışlar',

'Kendine Zarar Verme',

'Başka Birine Zarar Verme',

'İntihar Düşünceleri',

'İntihar Girişimi',

'Psikoza Dair Belirtiler',

'Gerçeklik Algısı Bozuklukları',

'Duygu Durumu Değişiklikleri',

'Motivasyon Eksikliği',

'Enerji Düşüklüğü',

'Cinsel İsteksizlik',

'Aşırı Cinsel İstek',

'Cinsel Yolla Bulaşan Enfeksiyonlar',

'Hamilelik',

'Doğum Kontrolü',

'Menopoz',

'Andropoz',

'Hormon Bozuklukları',

'Tiroid Hastalıkları',

'Diyabet',

'Hipertansiyon',

'Kalp Hastalıkları',

'Felç',

'Kronik Ağrı',

'Fibromiyalji',

'Kronik Yorgunluk Sendromu',

'Uyku Apnesi',

'Astım',

'Kronik Obstrüktif Akciğer Hastalığı',

'Akciğer Kanseri',

'Mide Kanseri',

'Bağırsak Kanseri',

'Karaciğer Kanseri',

'Pankreas Kanseri',

'Böbrek Kanseri',

'Mesane Kanseri',

'Prostat Kanseri',

'Cilt Kanseri',

'Lenfoma',

'Multiple Myeloma',

'Leukemi',

'Hodgkin Dışı Lenfoma',

'Akut Lenfoblastik Lösemi',

```
  'Kronik Lenfositik Lösemi',
  'Akut Miyeloid Lösemi',
  'Kronik Miyeloid Lösemi',
];

const List<List<String>> diseaseSuggestions = [
  ['Nemlendirici kullanın', 'Doktora görünün'],
  ['Dermatolog önerisi alın', 'Cilt bakımınıza dikkat edin'],
  ['Alerjenlerden kaçının', 'Doktora danışın'],
  ['Akne tedavisi için dermatologa başvurun'],
  ['Güneş koruyucu kullanın', 'Doktora danışın'],
  ['Cilt temizliğine özen gösterin', 'Doktora görünün'],
  ['Antibiyotik kullanın', 'Doktora danışın'],
  ['Viral enfeksiyonlar için doktora başvurun'],
  ['Antifungal krem kullanın', 'Doktora danışın'],
  ['Antiparaziter tedavi alın', 'Doktora başvurun'],
  ['İltihap önleyici krem kullanın', 'Doktora danışın'],
  ['Bağışıklık sisteminizi güçlendirin', 'Doktora görünün'],
  ['Kanser taraması yaptırın', 'Doktora başvurun'],
  ['Diyabet kontrolü yaptırın', 'Beslenmenize dikkat edin'],
  ['Hipertansiyon için doktora başvurun'],
  ['Astım tedavisi için doktora başvurun'],
  ['Kronik obstrüktif akciğer hastalığı için doktora görünün'],
  ['Pulmoner fibrozis için doktora başvurun'],
```

['Akciğer kanseri taraması yaptırın', 'Doktora başvurun'],

['Mide kanseri taraması yaptırın', 'Doktora başvurun'],

['Bağırsak kanseri taraması yaptırın', 'Doktora başvurun'],

['Karaciğer kanseri taraması yaptırın', 'Doktora başvurun'],

['Pankreas kanseri taraması yaptırın', 'Doktora başvurun'],

['Böbrek kanseri taraması yaptırın', 'Doktora başvurun'],

['Mesane kanseri taraması yaptırın', 'Doktora başvurun'],

['Prostat kanseri taraması yaptırın', 'Doktora başvurun'],

['Cilt kanseri taraması yaptırın', 'Doktora başvurun'],

['Lenfoma taraması yaptırın', 'Doktora başvurun'],

['Multiple myeloma taraması yaptırın', 'Doktora başvurun'],

['Leukemi taraması yaptırın', 'Doktora başvurun'],

['Hodgkin dışı lenfoma taraması yaptırın', 'Doktora başvurun'],

['Akut lenfoblastik lösemi taraması yaptırın', 'Doktora başvurun'],

['Kronik lenfositik lösemi taraması yaptırın', 'Doktora başvurun'],

['Akut miyeloid lösemi taraması yaptırın', 'Doktora başvurun'],

['Kronik miyeloid lösemi taraması yaptırın', 'Doktora başvurun'],

];

Ek-C: Sunucu ile RestFulAPI Kodları

import tensorflow as tf

import numpy as np

from tensorflow.keras.preprocessing.image import load_img, img_to_array

from flask import Flask, request, jsonify, send_file, send_from_directory, render_template_string

```
import os

import logging

from datetime import datetime

import threading

import re


# Flask uygulaması ve log ayarları

app = Flask(__name__)

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger(__name__)


# Klasör yolları

UPLOAD_FOLDER = 'images'

TAHMIN_FOLDER = 'tahmin'

STATIC_FOLDER = 'static'

APP_ROOT = os.path.dirname(os.path.abspath(__file__))

UPLOAD_FOLDER_ABSOLUTE = os.path.join(APP_ROOT, UPLOAD_FOLDER)

TAHMIN_FOLDER_ABSOLUTE = os.path.join(APP_ROOT, TAHMIN_FOLDER)


os.makedirs(UPLOAD_FOLDER_ABSOLUTE, exist_ok=True)
```

```
os.makedirs(TAHMIN_FOLDER_ABSOLUTE, exist_ok=True)


# TFLite modelini yükle
interpreter = None
input_details = None
output_details = None
try:
    interpreter = tf.lite.Interpreter(model_path='model_quant_old.tflite')
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    logger.info("TFLite modeli başarıyla yüklendi.")
except Exception as e:
    logger.error(f"TFLite model yüklenirken KRİTİK HATA: {e}", exc_info=True)
    exit()


classes = ['bcc', 'df', 'mel', 'nv', 'vasc']


# Default threshold for all classes
PREDICTION_THRESHOLD = 0.35
# Special threshold for "nv" class (index 3)
NV_THRESHOLD = 0.55
```

```
latest_image_filename_for_tahmin_endpoint = None
latest_tahmin_filename_for_tahmin_endpoint = None
processing_lock = threading.Lock()

def make_prediction(image_path):
    try:
        img = load_img(image_path, target_size=(224, 224))
        img_array = img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0) / 255.0

        if input_details[0]['dtype'] == np.int8 or input_details[0]['dtype'] == np.uint8:
            scale, zero_point = input_details[0]['quantization']
            if scale != 0:
                img_array = (img_array / scale +
zero_point).astype(input_details[0]['dtype'])

        interpreter.set_tensor(input_details[0]['index'], img_array)
        interpreter.invoke()
        predictions_raw = interpreter.get_tensor(output_details[0]['index'])

        if output_details[0]['dtype'] == np.int8 or output_details[0]['dtype'] ==
np.uint8:
```

```
            output_scale, output_zero_point = output_details[0]['quantization']
            if output_scale != 0:
                predictions_float = predictions_raw.astype(np.float32)
                predictions = (predictions_float - output_zero_point) * output_scale
            else:
                predictions = predictions_raw.astype(np.float32)
        else:
            predictions = predictions_raw.astype(np.float32)

        predicted_class_index = np.argmax(predictions[0])
        predicted_class_name = classes[predicted_class_index]
        max_probability = float(predictions[0][predicted_class_index])
        probabilities = {classes[i]: float(predictions[0][i]) for i in range(len(classes))}

        logger.info(f"Tahmin yapıldı: {predicted_class_name}, Olasılıklar:
{probabilities}")

        # Class-specific threshold: "nv" için 0.8, diğerleri için 0.65
        if predicted_class_name == "nv":
            threshold = NV_THRESHOLD
        else:
            threshold = PREDICTION_THRESHOLD
```

```
        if max_probability < threshold:
            logger.warning(f"Tahminin güveni düşük [{predicted_class_name}] ({max_probability:.4f} < {threshold}), sonuç null döndürülüyor.")
            return None, None

        return predicted_class_name, probabilities
    except Exception as e:
        logger.error(f"Tahmin yaparken hata: {e}", exc_info=True)
        return None, None

@app.route('/images', methods=['POST'])
def upload_image_from_flutter():
    global latest_image_filename_for_tahmin_endpoint
    global latest_tahmin_filename_for_tahmin_endpoint

    if 'file' not in request.files:
        logger.warning("POST /images: İstekte 'file' parametresi bulunamadı.")
        return jsonify({'error': "'file' parametresi eksik"}), 400

    file = request.files['file']
```

```
    if not file.filename:
        logger.warning("POST /images: Boş dosya adı.")
        return jsonify({'error': 'Geçersiz dosya adı'}), 400

    if not re.match(r'.*\.(jpg|jpeg|png)$', file.filename, re.IGNORECASE):
        logger.warning(f"POST /images: Geçersiz dosya türü veya adı:
{file.filename}")
        return jsonify({'error': 'Geçersiz dosya türü (sadece jpg, jpeg, png)'}), 400

    try:
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S_%f")
        new_filename = f"image_{timestamp}.{file.filename.rsplit('.', 1)[-1].lower()}"

        filepath = os.path.join(UPLOAD_FOLDER_ABSOLUTE, new_filename)
        file.save(filepath)
        logger.info(f"Görüntü başarıyla kaydedildi: {filepath}")

        predicted_class, probabilities = make_prediction(filepath)

        current_tahmin_filename = None
        if predicted_class and probabilities:
            tahmin_file_basename = f"tahmin_{timestamp}.txt"
```

```
            tahmin_filepath = os.path.join(TAHMIN_FOLDER_ABSOLUTE,
tahmin_file_basename)


            with open(tahmin_filepath, 'w') as f:
                f.write(f"Tahmin edilen sınıf: {predicted_class}\n")
                f.write("Olasılıklar:\n")
                for cls, prob in probabilities.items():
                    f.write(f"{cls}: {prob:.4f}\n")
            current_tahmin_filename = tahmin_file_basename
            logger.info(f"Tahmin sonucu kaydedildi: {tahmin_filepath}")
        else:
            logger.warning(f"Görüntü için tahmin yapılamadı veya güven eşiğinin
altında: {filepath}")


        with processing_lock:
            latest_image_filename_for_tahmin_endpoint = new_filename
            latest_tahmin_filename_for_tahmin_endpoint = current_tahmin_filename


        return jsonify({'message': 'Dosya başarıyla yüklendi', 'filename':
new_filename}), 201


    except Exception as e:
        logger.error(f"POST /images: Görüntü işlenirken/kaydedilirken hata: {e}",
exc_info=True)
```

```
        return jsonify({'error': 'Görüntü işlenirken sunucuda bir hata oluştu'}), 500


@app.route('/upload_image', methods=['POST'])
def upload_raw_image_from_esp32():
    global latest_image_filename_for_tahmin_endpoint
    global latest_tahmin_filename_for_tahmin_endpoint

    if 'image/jpeg' not in request.content_type:
        logger.warning(f"POST /upload_image: Geçersiz Content-Type: {request.content_type}")
        return jsonify({'error': 'Content-Type image/jpeg olmalı'}), 400

    image_data = request.get_data()
    if not image_data:
        logger.warning("POST /upload_image: Boş görüntü verisi alındı.")
        return jsonify({'error': 'Görüntü verisi alınamadı'}), 400

    try:
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S_%f")
        filename = f'image_{timestamp}.jpg'
        filepath = os.path.join(UPLOAD_FOLDER_ABSOLUTE, filename)
        with open(filepath, 'wb') as f:
```

```
            f.write(image_data)
        logger.info(f"Ham görüntü başarıyla kaydedildi: {filepath}")

        predicted_class, probabilities = make_prediction(filepath)
        current_tahmin_filename = None

        if predicted_class and probabilities:
            tahmin_file_basename = f"tahmin_{timestamp}.txt"
            tahmin_filepath = os.path.join(TAHMIN_FOLDER_ABSOLUTE,
tahmin_file_basename)
            with open(tahmin_filepath, 'w') as f:
                f.write(f"Tahmin edilen sınıf: {predicted_class}\n")
                f.write("Olasılıklar:\n")
                for cls, prob in probabilities.items():
                    f.write(f"{cls}: {prob:.4f}\n")
            current_tahmin_filename = tahmin_file_basename
            logger.info(f"Ham görüntü için tahmin sonucu kaydedildi:
{tahmin_filepath}")

            result_for_esp = {
                "hasDisease": True,
                "diseaseName": predicted_class,
                "probabilities": probabilities
```

```
            }

            with processing_lock:

                latest_image_filename_for_tahmin_endpoint = filename

                latest_tahmin_filename_for_tahmin_endpoint =
current_tahmin_filename

            return jsonify({'result': result_for_esp, 'filename': filename, 'tahmin_file':
current_tahmin_filename}), 201

        else:

            logger.warning(f"Ham görüntü için tahmin yapılamadı veya güven eşiğinin
altında: {filepath}")

            with processing_lock:

                latest_image_filename_for_tahmin_endpoint = filename

                latest_tahmin_filename_for_tahmin_endpoint = None

            return jsonify({'error': 'Tahmin yapılamadı veya güven eşiğinin altında'}),
500



    except Exception as e:

        logger.error(f"POST /upload_image: Ham görüntü işlenirken hata: {e}",
exc_info=True)

        return jsonify({'error': 'Görüntü işlenirken sunucuda bir hata oluştu'}), 500



@app.route('/tahmin', methods=['GET'])

def get_latest_tahmin_for_flutter():

    global latest_tahmin_filename_for_tahmin_endpoint
```

```
    global processing_lock

    tahmin_to_serve = None
    with processing_lock:
        tahmin_to_serve = latest_tahmin_filename_for_tahmin_endpoint

    if tahmin_to_serve:
        tahmin_filepath = os.path.join(TAHMIN_FOLDER_ABSOLUTE,
tahmin_to_serve)
        if os.path.exists(tahmin_filepath):
            try:
                with open(tahmin_filepath, 'r') as f:
                    lines = f.readlines()

                if not lines or len(lines) < 2:
                    logger.error(f"GET /tahmin: {tahmin_to_serve} dosyası beklenen
formatta değil.")
                    return jsonify({'error': 'Tahmin dosyası formatı geçersiz'}), 500

                predicted_class = lines[0].replace("Tahmin edilen sınıf:", "").strip()
                probabilities = {}
                try:
```

```
                prob_start_index = next(i for i, line in enumerate(lines) if
"Olasılıklar:" in line) + 1

                for line in lines[prob_start_index:]:

                    if ':' in line:

                        parts = line.strip().split(':')

                        if len(parts) == 2:

                            cls = parts[0].strip()

                            prob_str = parts[1].strip()

                            try:

                                probabilities[cls] = float(prob_str)

                            except ValueError:

                                logger.warning(f"GET /tahmin: {tahmin_to_serve}
dosyasında geçersiz olasılık değeri: {prob_str} for class {cls}")

                        else:

                            logger.warning(f"GET /tahmin: {tahmin_to_serve} dosyasında
geçersiz olasılık satırı formatı: {line.strip()}")

                    else:

                        logger.warning(f"GET /tahmin: {tahmin_to_serve} dosyasında ':'
içermeyen olasılık satırı: {line.strip()}")

            except StopIteration:

                logger.error(f"GET /tahmin: {tahmin_to_serve} dosyasında
'Olasılıklar:' satırı bulunamadı.")

                return jsonify({'error': "Tahmin dosyasında 'Olasılıklar:' başlığı
eksik"}), 500
```

```
            result = {
                "hasDisease": True,
                "diseaseName": predicted_class,
                "probabilities": probabilities
            }
            logger.info(f"GET /tahmin: {tahmin_to_serve} için sonuç başarıyla oluşturuldu.")
            return jsonify(result), 200
        except Exception as e:
            logger.error(f"GET /tahmin: {tahmin_to_serve} okunurken/işlenirken hata: {e}", exc_info=True)
            return jsonify({'error': 'Tahmin sonucu işlenirken sunucuda hata oluştu'}), 500
    else:
        logger.warning(f"GET /tahmin: En son tahmin dosyası ({tahmin_to_serve}) diskte bulunamadı.")
        return jsonify({'error': 'En son tahmin sonucu bulunamadı (dosya diskte yok)'}), 404
else:
    logger.warning("GET /tahmin: Sunucuda kayıtlı en son tahmin bulunmuyor.")
    return jsonify({'error': 'Henüz işlenmiş bir tahmin yok'}), 404


@app.route('/images', methods=['GET'])
def list_images():
```

```
    try:
        files = sorted([
            f for f in os.listdir(UPLOAD_FOLDER_ABSOLUTE)
            if os.path.isfile(os.path.join(UPLOAD_FOLDER_ABSOLUTE, f)) and
               re.match(r'.*\.(jpg|jpeg|png)$', f, re.IGNORECASE)
        ], key=lambda x:
os.path.getmtime(os.path.join(UPLOAD_FOLDER_ABSOLUTE, x)),
reverse=True)
        return jsonify({'files': files}), 200
    except Exception as e:
        logger.error(f"GET /images listesi alınırken hata: {e}", exc_info=True)
        return jsonify({'error': 'Dosya listesi alınamadı'}), 500


@app.route('/images/<path:filename>')
def serve_image(filename):
    logger.info(f"GET /images/{filename} isteği alındı.")
    try:
        return send_from_directory(UPLOAD_FOLDER_ABSOLUTE, filename,
as_attachment=False)
    except FileNotFoundError:
        logger.warning(f"GET /images/{filename}: Dosya bulunamadı.")
        return jsonify({'error': 'Dosya bulunamadı'}), 404
    except Exception as e:
```

```
        logger.error(f"GET /images/{filename} sunulurken hata: {e}", exc_info=True)
        return jsonify({'error': 'Dosya sunulurken hata'}), 500

@app.route('/tahmin/<path:filename>', methods=['GET'])
def serve_tahmin_file(filename):
    logger.info(f"GET /tahmin/{filename} isteği alındı.")
    try:
        return send_from_directory(TAHMIN_FOLDER_ABSOLUTE, filename, as_attachment=False, mimetype='text/plain')
    except FileNotFoundError:
        logger.warning(f"GET /tahmin/{filename}: Tahmin dosyası bulunamadı.")
        return jsonify({'error': 'Tahmin dosyası bulunamadı'}), 404
    except Exception as e:
        logger.error(f"GET /tahmin/{filename} sunulurken hata: {e}", exc_info=True)
        return jsonify({'error': 'Tahmin dosyası sunulurken hata'}), 500

LIVE_VIEW_HTML = """
<!DOCTYPE html>
<html>
<head>
    <title>Sunucu Canlı Görüntü</title>
    <style>
```

```
        body { font-family: Arial, sans-serif; margin: 0; padding: 20px; background-color: #f4f4f4; text-align: center; }

        h1 { color: #333; }

        #liveImage { border: 2px solid #ccc; border-radius: 8px; max-width: 800px; max-height: 600px; width: auto; height: auto; background-color: #fff; padding: 5px; }

        p { color: #666; }

        .upload-form { margin-top: 20px; padding:15px; background-color: #fff; border-radius: 8px; display: inline-block; }

    </style>

</head>

<body>

    <h1>Sunucu Anlık Görüntü Akışı</h1>

    <p>Bu sayfa, sunucuya en son yüklenen resmi otomatik olarak yeniler.</p>

    <img id="liveImage" src="/latest_image_feed" alt="Canlı Görüntü Yükleniyor...">

    <p id="timestamp">Son Güncelleme: Bekleniyor...</p>

    <div class="upload-form">

        <h2>Test için Resim Yükle (POST /images)</h2>

        <form id="uploadForm" action="/images" method="post" enctype="multipart/form-data">

            <input type="file" name="file" id="fileInput" accept="image/jpeg,image/png" required>

            <button type="submit">Yükle ve Tahmin Et</button>

        </form>
```

```
        <div id="uploadResult" style="margin-top:10px;"></div>
    </div>
    <script>
        const imageElement = document.getElementById('liveImage');
        const timestampElement = document.getElementById('timestamp');
        const refreshInterval = 2000;
        function refreshImage() {
            imageElement.src = "/latest_image_feed?" + new Date().getTime();
        }
        function updateTimestamp() {
            timestampElement.textContent = "Son Güncelleme: " + new
Date().toLocaleTimeString();
        }
        imageElement.onload = updateTimestamp;
        imageElement.onerror = function() {
            timestampElement.textContent = "Görüntü yüklenemedi veya henüz
görüntü yok.";
        };
        setInterval(refreshImage, refreshInterval);
        const uploadForm = document.getElementById('uploadForm');
        const uploadResultDiv = document.getElementById('uploadResult');
        if (uploadForm) {
            uploadForm.addEventListener('submit', async function(event) {
```

```
            event.preventDefault();
            const formData = new FormData(uploadForm);
            uploadResultDiv.textContent = 'Yükleniyor...';
            try {
                const response = await fetch('/images', {
                    method: 'POST',
                    body: formData
                });
                const result = await response.json();
                if (response.ok) {
                    uploadResultDiv.textContent = 'Yükleme Başarılı: ' +
JSON.stringify(result);
                } else {
                    uploadResultDiv.textContent = 'Yükleme Hatası: ' +
JSON.stringify(result);
                }
            } catch (error) {
                uploadResultDiv.textContent = 'Bir hata oluştu: ' + error;
            }
        });
    }
  </script>
</body>
```

```
</html>
"""

@app.route('/')
@app.route('/live_view')
def live_view_page():
    return render_template_string(LIVE_VIEW_HTML)

@app.route('/latest_image_feed')
def latest_image_feed_endpoint():
    global latest_image_filename_for_tahmin_endpoint
    global processing_lock

    current_image_filename_local = None
    with processing_lock:
        current_image_filename_local = latest_image_filename_for_tahmin_endpoint

    if current_image_filename_local:
        image_path_to_serve = os.path.join(UPLOAD_FOLDER_ABSOLUTE,
current_image_filename_local)
        if os.path.exists(image_path_to_serve):
            logger.debug(f"Feed: En son görüntü sunuluyor: {image_path_to_serve}")
```

```
        try:
            return send_file(image_path_to_serve, mimetype='image/jpeg',
                             as_attachment=False, download_name='live.jpg', max_age=0)
        except Exception as e:
            logger.error(f"Feed: {image_path_to_serve} gönderilirken hata: {e}", exc_info=True)
            return jsonify({'error': 'Görüntü dosyası gönderilirken hata'}), 500
    else:
        logger.warning(f"Feed: {image_path_to_serve} diskte bulunamadı ama adı kayıtlıydı.")

    logger.info("Feed: Gösterilecek kaydedilmiş görüntü bulunamadı veya bir hata oluştu. Şeffaf PNG dönülüyor.")
    from io import BytesIO
    import base64
    transparent_png_data = base64.b64decode("iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAQAAAC1HAwCAAAAC0lEQVR42mNkYAAAAAYAAjCB0C8AAAAASUVORK5CYII=")
    return send_file(BytesIO(transparent_png_data), mimetype='image/png', max_age=0)

if __name__ == '__main__':
    flask_port = 5001
    logger.info(f"Flask sunucusu HTTPS olarak başlatılıyor. https://0.0.0.0:{flask_port}")
```

```
    cert_path = os.path.join(APP_ROOT, 'cert.pem')
    key_path = os.path.join(APP_ROOT, 'key.pem')

    if not os.path.exists(cert_path) or not os.path.exists(key_path):
        logger.critical(f"KRİTİK HATA: SSL sertifika ({cert_path}) veya anahtar ({key_path}) dosyası bulunamadı! Sunucu başlatılamıyor.")
        print(f"HATA: SSL sertifika ({cert_path}) veya anahtar ({key_path}) dosyası bulunamadı!")
        exit()

    try:
        app.run(host='0.0.0.0', port=flask_port, debug=True, use_reloader=False,
                ssl_context=(cert_path, key_path))
    except Exception as e:
        logger.critical(f"Sunucu başlatılırken KRİTİK HATA: {e}", exc_info=True)
```

Ek-D: MobilnetV2 Model Eğitimi Kodları

```
#!/usr/bin/env python3
```

```
# -*- coding: utf-8 -*-


import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

os.environ["QT_LOGGING_RULES"] = "qt5ct.debug=false"


import tensorflow as tf

import numpy as np

import pandas as pd

import matplotlib

matplotlib.use('Agg')

import matplotlib.pyplot as plt

import seaborn as sns

sns.set_style('darkgrid')

from tqdm import tqdm

import warnings

from matplotlib.backends.backend_pdf import PdfPages

import cv2


from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D

from tensorflow.keras.optimizers import Adam

from tensorflow.keras import regularizers

from tensorflow.keras.models import Model
```

```
from tensorflow.keras.applications import MobileNetV3Large

from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping, Callback


from sklearn.metrics import f1_score, confusion_matrix, classification_report, roc_curve, roc_auc_score, precision_recall_curve


try:

    import scikitplot as skplt

    SCIKITPLOT_AVAILABLE = True

except ImportError:

    SCIKITPLOT_AVAILABLE = False

    print("scikit-plot not found. Lift and cumulative gains curves will be skipped.")


warnings.simplefilter("ignore")

pd.set_option('display.max_columns', None)

pd.set_option('display.max_rows', None)

pd.set_option('display.max_colwidth', None)


try:

    from tensorflow_addons.losses import SigmoidFocalCrossEntropy

    TFA_AVAILABLE = True

    print("TensorFlow Addons found (SigmoidFocalCrossEntropy check successful).")

except ImportError:

    TFA_AVAILABLE = False
```

```
    print("TensorFlow Addons not found. Custom Categorical Focal Loss will be used.")

gpus = tf.config.list_physical_devices('GPU')
if gpus:
    print(f"GPU(s) found: {[gpu.name for gpu in gpus]}")
    print("TensorFlow will use GPU.")
else:
    print("No GPU found! TensorFlow will use CPU.")

def categorical_focal_loss(gamma=2.0, alpha=0.25):
    def focal_loss(y_true, y_pred):
        epsilon = tf.keras.backend.epsilon()
        y_pred = tf.clip_by_value(y_pred, epsilon, 1. - epsilon)
        pt = tf.reduce_sum(y_true * y_pred, axis=-1)
        focal_loss_val = -alpha * tf.pow(1. - pt, gamma) * tf.math.log(pt)
        return tf.reduce_mean(focal_loss_val)
    return focal_loss

print('All modules have been imported')
print(f"NumPy version: {np.__version__}")
print(f"Pandas version: {pd.__version__}")
print(f"TensorFlow version: {tf.__version__}")

def print_in_color(txt_msg, fore_tupple=(0,255,255), back_tupple=(100,100,100)):
```

```
    rf, gf, bf = fore_tupple

    rb, gb, bb = back_tupple

    msg = '{0}' + txt_msg

    mat = '\33[38;2;' + str(rf) +';' + str(gf) + ';' + str(bf) + ';48;2;' + str(rb) + ';' + str(gb) + ';' + str(bb) +
'm'

    print(msg.format(mat), flush=True)

    print('\33[0m', flush=True)


msg = 'Test of default colors for console output.'

print_in_color(msg)


# --- CALLBACK: Per-Class metrics and logging ---

class PerClassMetrics(Callback):

    def __init__(self, train_generator, validation_generator, classes, eval_frequency=1,
sample_size=1000):

        super(PerClassMetrics, self).__init__()

        self.train_gen = train_generator

        self.valid_gen = validation_generator

        self.classes = classes

        self.num_classes = len(classes)

        self.class_indices = train_generator.class_indices

        self.eval_frequency = eval_frequency

        self.sample_size = sample_size


    def on_epoch_end(self, epoch, logs=None):
```

```
if (epoch + 1) % self.eval_frequency != 0:
    return
logs = logs or {}
self.train_gen.reset()
self.valid_gen.reset()

train_y_true, train_y_pred = [], []
valid_y_true, valid_y_pred = [], []

train_steps = min(len(self.train_gen), max(1, self.sample_size // self.train_gen.batch_size))
for _ in range(train_steps):
    X, y = next(self.train_gen)
    y_pred = self.model.predict(X, verbose=0)
    train_y_true.extend(np.argmax(y, axis=1))
    train_y_pred.extend(np.argmax(y_pred, axis=1))

valid_steps = min(len(self.valid_gen), max(1, self.sample_size // self.valid_gen.batch_size))
for _ in range(valid_steps):
    X, y = next(self.valid_gen)
    y_pred = self.model.predict(X, verbose=0)
    valid_y_true.extend(np.argmax(y, axis=1))
    valid_y_pred.extend(np.argmax(y_pred, axis=1))

train_y_true = np.array(train_y_true)
```

```
train_y_pred = np.array(train_y_pred)
valid_y_true = np.array(valid_y_true)
valid_y_pred = np.array(valid_y_pred)

train_class_acc = []
valid_class_acc = []
train_class_f1 = []
valid_class_f1 = []

for i, cls in enumerate(self.classes):
    train_mask = train_y_true == i
    if train_mask.sum() > 0:
        train_class_acc.append(np.mean(train_y_pred[train_mask] == i))
    else:
        train_class_acc.append(0.0)

    valid_mask = valid_y_true == i
    if valid_mask.sum() > 0:
        valid_class_acc.append(np.mean(valid_y_pred[valid_mask] == i))
    else:
        valid_class_acc.append(0.0)

    train_f1 = f1_score(train_y_true, train_y_pred, labels=[i], average='micro', zero_division=0)
    valid_f1 = f1_score(valid_y_true, valid_y_pred, labels=[i], average='micro', zero_division=0)
```

```
            train_class_f1.append(train_f1)
            valid_class_f1.append(valid_f1)

        for i, cls in enumerate(self.classes):
            logs[f'train_{cls}_accuracy'] = train_class_acc[i]
            logs[f'valid_{cls}_accuracy'] = valid_class_acc[i]
            logs[f'train_{cls}_f1'] = train_class_f1[i]
            logs[f'valid_{cls}_f1'] = valid_class_f1[i]

# --- CALLBACK: Per-epoch plotting and extra analyses ---
class PerEpochPlotter(tf.keras.callbacks.Callback):
    def __init__(self, classes, valid_gen, out_dir="epoch_plots", pr_roc_frequency=5):
        super().__init__()
        self.classes = classes
        self.valid_gen = valid_gen
        self.out_dir = out_dir
        self.pr_roc_frequency = pr_roc_frequency
        os.makedirs(out_dir, exist_ok=True)
        self.history = {}

    def on_train_begin(self, logs=None):
        self.history = {}

    def on_epoch_end(self, epoch, logs=None):
```

```
        logs = logs or {}
        # Learning rate kaydı
        if hasattr(self.model.optimizer, 'lr'):
            try:
                lr_val = float(tf.keras.backend.get_value(self.model.optimizer.lr))
            except Exception:
                lr_val = self.model.optimizer.lr if isinstance(self.model.optimizer.lr, float) else 0.0
            logs['lr'] = lr_val

        for k, v in logs.items():
            self.history.setdefault(k, []).append(v)
        self.plot_epoch(epoch)

    def plot_epoch(self, epoch):
        pdf_path = f"{self.out_dir}/epoch_{epoch+1:03d}_plots.pdf"
        with PdfPages(pdf_path) as pdf:
            history = self.history
            epochs_range = range(1, len(history.get('accuracy', [])) + 1)
            # Aggregate
            fig, axes = plt.subplots(2, 3, figsize=(20,12))
            # Accuracy
            if 'accuracy' in history and 'val_accuracy' in history:
                axes[0,0].plot(epochs_range, history['accuracy'], label='Train Accuracy')
                axes[0,0].plot(epochs_range, history['val_accuracy'], label='Val Accuracy')
```

```
        axes[0,0].set_title('Aggregate Accuracy')
        axes[0,0].legend()
    # Loss
    if 'loss' in history and 'val_loss' in history:
        axes[0,1].plot(epochs_range, history['loss'], label='Train Loss')
        axes[0,1].plot(epochs_range, history['val_loss'], label='Val Loss')
        axes[0,1].set_title('Aggregate Loss')
        axes[0,1].legend()
    # Precision & Recall
    for metric, val_metric, label in [('precision', 'val_precision', 'Precision'), ('recall', 'val_recall',
'Recall')]:
        if metric in history and val_metric in history:
            axes[0,2].plot(epochs_range, history[metric], label=f'Train {label}')
            axes[0,2].plot(epochs_range, history[val_metric], label=f'Val {label}')
    axes[0,2].set_title('Aggregate Precision & Recall')
    axes[0,2].legend()
    # Learning rate
    if 'lr' in history:
        axes[1,0].plot(epochs_range, history['lr'], label='Learning Rate', color='g')
        axes[1,0].set_title('Learning Rate')
        axes[1,0].legend()
    # Top-3 accuracy
    if 'top3_acc' in history and 'val_top3_acc' in history:
        axes[1,1].plot(epochs_range, history['top3_acc'], label='Train Top-3 Acc')
        axes[1,1].plot(epochs_range, history['val_top3_acc'], label='Val Top-3 Acc')
```

```
    axes[1,1].set_title('Top-3 Accuracy')
    axes[1,1].legend()
# Overfitting/Underfitting Analizi
if 'accuracy' in history and 'val_accuracy' in history:
    acc_gap = np.array(history['accuracy']) - np.array(history['val_accuracy'])
    loss_gap = np.array(history['val_loss']) - np.array(history['loss'])
    axes[1,2].plot(epochs_range, acc_gap, label='Acc Gap (Train-Val)')
    axes[1,2].plot(epochs_range, loss_gap, label='Loss Gap (Val-Train)')
    axes[1,2].axhline(0, color='grey', linestyle='--')
    axes[1,2].set_title('Overfitting/Underfitting Analysis')
    axes[1,2].legend()
plt.tight_layout()
pdf.savefig(fig)
plt.close(fig)
# Per-class
for cls in self.classes:
    fig, axes = plt.subplots(1, 3, figsize=(18,6))
    if f'train_{cls}_accuracy' in history and f'valid_{cls}_accuracy' in history:
        axes[0].plot(epochs_range, history[f'train_{cls}_accuracy'], label='Train Accuracy')
        axes[0].plot(epochs_range, history[f'valid_{cls}_accuracy'], label='Val Accuracy')
        axes[0].set_title(f'{cls} Accuracy')
        axes[0].legend()
    if f'train_{cls}_f1' in history and f'valid_{cls}_f1' in history:
        axes[1].plot(epochs_range, history[f'train_{cls}_f1'], label='Train F1')
```

```
            axes[1].plot(epochs_range, history[f'valid_{cls}_f1'], label='Val F1')
            axes[1].set_title(f'{cls} F1 Score')
            axes[1].legend()
        axes[2].text(0.5, 0.5, 'Per-Class Loss Not Computed', ha='center', va='center')
        axes[2].set_title(f'{cls} Loss')
        plt.tight_layout()
        pdf.savefig(fig)
        plt.close(fig)
    # Her 5 epoch'ta bir ROC, PR, Confusion Matrix, Class Distribution, Precision/Recall/F1 Heatmap
    if (epoch + 1) % self.pr_roc_frequency == 0:
        print(f"Epoch {epoch+1}: Computing validation ROC, PR, confusion matrix, class distribution, metrics heatmap.")
        y_true = []
        y_pred_prob = []
        self.valid_gen.reset()
        for i in range(len(self.valid_gen)):
            X, y = next(self.valid_gen)
            y_pred = self.model.predict(X, verbose=0)
            y_true.append(y)
            y_pred_prob.append(y_pred)
        y_true = np.concatenate(y_true)
        y_pred_prob = np.concatenate(y_pred_prob)
        y_pred_cls = np.argmax(y_pred_prob, axis=1)
        y_true_cls = np.argmax(y_true, axis=1)
```

```
            # Confusion matrix
            fig, ax = plt.subplots(figsize=(8,6))
            cm = confusion_matrix(y_true_cls, y_pred_cls)
            sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=self.classes,
yticklabels=self.classes, ax=ax)
            ax.set_title(f'Confusion Matrix (Epoch {epoch+1})')
            ax.set_xlabel('Predicted')
            ax.set_ylabel('True')
            plt.tight_layout()
            pdf.savefig(fig)
            plt.close(fig)
            # ROC curves
            fig, ax = plt.subplots(figsize=(8,6))
            for i, cls in enumerate(self.classes):
                fpr, tpr, _ = roc_curve(y_true[:, i], y_pred_prob[:, i])
                try:
                    roc_auc = roc_auc_score(y_true[:, i], y_pred_prob[:, i])
                except Exception:
                    roc_auc = 0.0
                ax.plot(fpr, tpr, label=f'{cls} (AUC={roc_auc:.2f})')
            ax.plot([0, 1], [0, 1], linestyle='--', color='gray')
            ax.set_title(f'ROC Curves (Epoch {epoch+1})')
            ax.set_xlabel('False Positive Rate')
            ax.set_ylabel('True Positive Rate')
            ax.legend()
```

```
plt.tight_layout()
pdf.savefig(fig)
plt.close(fig)
# Precision-Recall curves
fig, ax = plt.subplots(figsize=(8,6))
for i, cls in enumerate(self.classes):
    prec, rec, _ = precision_recall_curve(y_true[:, i], y_pred_prob[:, i])
    ax.plot(rec, prec, label=cls)
ax.set_title(f'Precision-Recall Curves (Epoch {epoch+1})')
ax.set_xlabel('Recall')
ax.set_ylabel('Precision')
ax.legend()
plt.tight_layout()
pdf.savefig(fig)
plt.close(fig)
# Class distribution
fig, ax = plt.subplots(figsize=(8,4))
sns.histplot(y_true_cls, bins=len(self.classes), discrete=True, ax=ax)
ax.set_xticks(range(len(self.classes)))
ax.set_xticklabels(self.classes)
ax.set_title(f'Validation Class Distribution (Epoch {epoch+1})')
plt.tight_layout()
pdf.savefig(fig)
plt.close(fig)
```

```
            # Precision/Recall/F1 heatmap
            report = classification_report(y_true_cls, y_pred_cls, target_names=self.classes,
output_dict=True, zero_division=0)
            metric_mat = np.zeros((len(self.classes), 3))
            for i, cls in enumerate(self.classes):
                metric_mat[i, 0] = report[cls]['precision']
                metric_mat[i, 1] = report[cls]['recall']
                metric_mat[i, 2] = report[cls]['f1-score']
            fig, ax = plt.subplots(figsize=(8,6))
            sns.heatmap(metric_mat, annot=True, fmt=".2f", cmap="YlGnBu",
                        xticklabels=['Precision', 'Recall', 'F1'], yticklabels=self.classes, ax=ax)
            ax.set_title(f'Per-Class PRF1 Heatmap (Epoch {epoch+1})')
            plt.tight_layout()
            pdf.savefig(fig)
            plt.close(fig)
        print(f"Epoch {epoch+1} plots saved to {pdf_path}")


# --- DATASET, MODEL, GENERATOR KISMI ---


def make_dataframes(base_dir, use_undersampling=False, train_target_count=1960,
valid_target_count=420):
    splits = ['train', 'validation', 'test']
    dataframes = {}
    if not os.path.isdir(base_dir):
        print_in_color(f"Error: Base directory {base_dir} does not exist.", fore_tupple=(255,0,0)); exit(1)
```

```
    classlist = sorted([d for d in os.listdir(base_dir) if os.path.isdir(os.path.join(base_dir, d))])

    if not classlist:

        print_in_color(f"Error: No class subdirectories found in {base_dir}.", fore_tupple=(255,0,0));
exit(1)



    for split in splits:

        filepaths, labels = [], []

        for klass in classlist:

            split_dir = os.path.join(base_dir, klass, split)

            if not os.path.exists(split_dir):

                print_in_color(f"Warning: Dir {split_dir} not found for class {klass}. Skipping.",
fore_tupple=(255,165,0)); continue

            flist = sorted(os.listdir(split_dir))

            desc = f'{split:10s}-{klass:25s}'

            for f in tqdm(flist, ncols=130, desc=desc, unit='files', colour='green'):

                filepaths.append(os.path.join(split_dir, f)); labels.append(klass)

        df = pd.DataFrame({'filepaths': filepaths, 'labels': labels}) if filepaths else
pd.DataFrame(columns=['filepaths', 'labels'])

        if df.empty: print_in_color(f"Warning: No files for {split} set.", fore_tupple=(255,165,0))

        dataframes[split] = df



    train_df_orig = dataframes.get('train', pd.DataFrame(columns=['filepaths', 'labels']))

    valid_df_orig = dataframes.get('validation', pd.DataFrame(columns=['filepaths', 'labels']))

    test_df = dataframes.get('test', pd.DataFrame(columns=['filepaths', 'labels']))



    if use_undersampling:
```

```
        print_in_color("Applying undersampling...", fore_tupple=(255,255,0))

        train_dfs_sampled = [df_class.sample(n=min(train_target_count, len(df_class)),
random_state=42) for lbl, df_class in train_df_orig.groupby('labels')]

        train_df = pd.concat(train_dfs_sampled).sample(frac=1,
random_state=42).reset_index(drop=True) if train_dfs_sampled else
pd.DataFrame(columns=['filepaths', 'labels'])

        valid_dfs_sampled = [df_class.sample(n=min(valid_target_count, len(df_class)),
random_state=42) for lbl, df_class in valid_df_orig.groupby('labels')]

        valid_df = pd.concat(valid_dfs_sampled).sample(frac=1,
random_state=42).reset_index(drop=True) if valid_dfs_sampled else
pd.DataFrame(columns=['filepaths', 'labels'])

    else:

        print_in_color("Using full dataset (no undersampling).", fore_tupple=(0,255,0))

        train_df = train_df_orig.sample(frac=1, random_state=42).reset_index(drop=True)

        valid_df = valid_df_orig.sample(frac=1, random_state=42).reset_index(drop=True)



    classes_list = sorted(train_df['labels'].unique()) if not train_df.empty else classlist

    class_count_val = len(classes_list)

    return train_df, test_df, valid_df, classes_list, class_count_val



def make_gens(batch_size, train_df, test_df, valid_df, img_size_tuple):

    ycol = 'labels'

    trgen = ImageDataGenerator(horizontal_flip=True, rotation_range=25, width_shift_range=0.15,
height_shift_range=0.15,

                               zoom_range=0.15, shear_range=0.15, brightness_range=[0.8, 1.2],
fill_mode='nearest',

                               preprocessing_function=tf.keras.applications.mobilenet_v3.preprocess_input)

    t_and_v_gen =
ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet_v3.preprocess_input)
```

```
    print_in_color('Creating train generator...')

    train_gen = trgen.flow_from_dataframe(train_df, x_col='filepaths', y_col=ycol,
target_size=img_size_tuple,

                                          class_mode='categorical', color_mode='rgb', shuffle=True,
batch_size=batch_size)

    print_in_color('Creating valid generator...')

    valid_gen = t_and_v_gen.flow_from_dataframe(valid_df, x_col='filepaths', y_col=ycol,
target_size=img_size_tuple,

                                          class_mode='categorical', color_mode='rgb', shuffle=False,
batch_size=batch_size)

    test_gen, test_steps_eff = None, 0

    if not test_df.empty:

        test_batch_size_eff = min(batch_size, len(test_df))

        test_steps_eff = int(np.ceil(len(test_df) / test_batch_size_eff))

        print_in_color('Creating test generator...')

        test_gen = t_and_v_gen.flow_from_dataframe(test_df, x_col='filepaths', y_col=ycol,
target_size=img_size_tuple,

                                          class_mode='categorical', color_mode='rgb', shuffle=False,
batch_size=test_batch_size_eff)

        print(f'Test batch size: {test_batch_size_eff}, Test steps: {test_steps_eff}')

    else:

        print_in_color("Test DataFrame empty, test generator not created.", fore_tupple=(255,165,0))



    # Get classes from generator

    classes_from_gen = list(train_gen.class_indices.keys())

    print(f'Train classes from generator: {classes_from_gen}')
```

```
    # Save class indices to labels.txt
    labels_file_path = 'labels.txt'
    try:
        with open(labels_file_path, 'w', encoding='utf-8') as f:
            for class_name, index in sorted(train_gen.class_indices.items(), key=lambda x: x[1]):
                f.write(f"{index} {class_name}\n")
        print_in_color(f"Saved class indices to {labels_file_path}", fore_tupple=(0,255,0))
    except Exception as e:
        print_in_color(f"Error saving labels.txt: {e}", fore_tupple=(255,0,0))
        raise

    return train_gen, test_gen, valid_gen, test_steps_eff, classes_from_gen

def make_model(img_size_tuple, num_classes, dr_rate, l2_val, initial_lr, loss_type, focal_gamma,
focal_alpha):
    base = MobileNetV3Large(
        input_shape=(img_size_tuple[0], img_size_tuple[1], 3),
        include_top=False,
        weights='imagenet'
    )
    print_in_color("MobileNetV3Large loaded successfully.")
    base.trainable = False
    x = GlobalAveragePooling2D()(base.output)
    x = Dense(256, activation='relu', kernel_regularizer=regularizers.l2(l2_val))(x)
```

```
    x = Dropout(dr_rate)(x)
    outputs = Dense(num_classes, activation='softmax')(x)
    model_instance = Model(inputs=base.input, outputs=outputs)
    loss_name_for_print = loss_type
    metrics_list = [
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        tf.keras.metrics.TopKCategoricalAccuracy(k=3, name='top3_acc')
    ]
    if loss_type == 'focal_loss':
        selected_loss = categorical_focal_loss(gamma=focal_gamma, alpha=focal_alpha)
        loss_name_for_print = f"Custom Categorical Focal Loss (g={focal_gamma},a={focal_alpha})"
    else:
        selected_loss = 'categorical_crossentropy'
    model_instance.compile(optimizer=Adam(learning_rate=initial_lr), loss=selected_loss,
metrics=metrics_list)
    print_in_color(f"Model created. LR={initial_lr}, Loss: {loss_name_for_print}")
    return model_instance, base


# === HAZIRLIK ===
BASE_DIR = './data2'
USE_UNDERSAMPLING = False
TRAIN_TARGET_COUNT_IF_UNDERSAMPLING = 2000
VALID_TARGET_COUNT_IF_UNDERSAMPLING = 500
```

```
IMG_SIZE = (224, 224)

BATCH_SIZE = 64

INITIAL_EPOCHS = 5

FINE_TUNE_EPOCHS = 1

FINE_TUNE_AT_PERCENT = 0.4

LEARNING_RATE_INITIAL = 1e-4

LEARNING_RATE_FINETUNE = 1e-5

DROPOUT_RATE = 0.45

L2_REGULARIZATION = 0.001

LOSS_FUNCTION_TYPE = 'focal_loss'

FOCAL_LOSS_GAMMA = 2.0

FOCAL_LOSS_ALPHA = 0.25



train_df, test_df, valid_df, classes, class_count = make_dataframes(

    BASE_DIR, USE_UNDERSAMPLING, TRAIN_TARGET_COUNT_IF_UNDERSAMPLING, VALID_TARGET_COUNT_IF_UNDERSAMPLING

)

if train_df.empty or valid_df.empty or class_count == 0:

    print_in_color("Critical error: DataFrames empty or no classes. Exiting.", fore_tupple=(255,0,0)); exit(1)



train_gen, test_gen, valid_gen, test_steps, classes_from_generator = make_gens(

    BATCH_SIZE, train_df, test_df, valid_df, IMG_SIZE

)

classes = classes_from_generator
```

```
class_count = len(classes)


model, base_model_obj = make_model(IMG_SIZE, class_count, DROPOUT_RATE,
L2_REGULARIZATION, LEARNING_RATE_INITIAL,

                                   LOSS_FUNCTION_TYPE, FOCAL_LOSS_GAMMA, FOCAL_LOSS_ALPHA)


callbacks_initial_phase = [

    ModelCheckpoint('initial_best_weights_mnv3.weights.h5', monitor='val_loss', save_best_only=True,
save_weights_only=True, mode='min', verbose=1),

    EarlyStopping(monitor='val_loss', patience=12, restore_best_weights=True, verbose=1),

    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=6, min_lr=1e-7, verbose=1),

    PerClassMetrics(train_gen, valid_gen, classes, eval_frequency=1, sample_size=1000),

    PerEpochPlotter(classes, valid_gen, out_dir="epoch_plots_initial", pr_roc_frequency=5)

]


print_in_color(f"--- Initial Training (Top Layers) for {INITIAL_EPOCHS} epochs ---",
fore_tupple=(0,255,0))

try:

    history_initial_phase = model.fit(

        train_gen,

        epochs=INITIAL_EPOCHS,

        validation_data=valid_gen,

        callbacks=callbacks_initial_phase,

        verbose=1

    )

except Exception as e:
```

```
    print_in_color(f"Initial training failed: {e}", fore_tupple=(255,0,0))
    # Model ve verileri kurtar!
    model.save_weights("emergency_initial_weights.h5")
    model.save("emergency_initial_model.keras")
    raise

if os.path.exists('initial_best_weights_mnv3.weights.h5'):
    try:
        model.load_weights('initial_best_weights_mnv3.weights.h5')
        print_in_color("Loaded best initial weights for fine-tuning.", fore_tupple=(255,255,0))
    except Exception as e:
        print_in_color(f"Could not load best initial weights: {e}", fore_tupple=(255,0,0))

base_model_obj.trainable = True
fine_tune_from_layer = int(len(base_model_obj.layers) * (1 - FINE_TUNE_AT_PERCENT))
for layer in base_model_obj.layers[:fine_tune_from_layer]: layer.trainable = False

model.compile(
    optimizer=Adam(learning_rate=LEARNING_RATE_FINETUNE),
    loss=categorical_focal_loss(gamma=FOCAL_LOSS_GAMMA, alpha=FOCAL_LOSS_ALPHA),
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
```

```
        tf.keras.metrics.TopKCategoricalAccuracy(k=3, name='top3_acc')
    ]
)

callbacks_finetune_phase = [
    ModelCheckpoint('finetune_best_weights_mnv3.weights.h5', monitor='val_loss',
save_best_only=True, save_weights_only=True, mode='min', verbose=1),
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True, verbose=1),
    ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=5, min_lr=1e-8, verbose=1),
    PerClassMetrics(train_gen, valid_gen, classes, eval_frequency=1, sample_size=1000),
    PerEpochPlotter(classes, valid_gen, out_dir="epoch_plots_finetune", pr_roc_frequency=5)
]

start_epoch_ft = history_initial_phase.epoch[-1] + 1 if history_initial_phase and
hasattr(history_initial_phase, 'epoch') else INITIAL_EPOCHS
total_fine_tune_epochs = start_epoch_ft + FINE_TUNE_EPOCHS

print_in_color(f"--- Fine-Tuning for {FINE_TUNE_EPOCHS} epochs ---", fore_tupple=(0,255,0))
try:
    history_finetune_phase = model.fit(
        train_gen,
        epochs=total_fine_tune_epochs,
        initial_epoch=start_epoch_ft,
        validation_data=valid_gen,
        callbacks=callbacks_finetune_phase,
```

```
        verbose=1
    )
except Exception as e:
    print_in_color(f"Fine-tuning failed: {e}", fore_tupple=(255,0,0))
    model.save_weights("emergency_finetune_weights.h5")
    model.save("emergency_finetune_model.keras")
    raise

# === EĞİTİM SONRASI GÜVENLİ KAYIT ===
try:
    model.save_weights('final_mnv3_weights.h5')
    model.save('final_mnv3_model.keras')
    print_in_color("Final model and weights saved securely.", fore_tupple=(0,255,0))
except Exception as e:
    print_in_color(f"Error saving final model: {e}", fore_tupple=(255,0,0))

print_in_color("--- Script Execution Completed ---", fore_tupple=(0,255,0))
```

Ek-F Model Kullanimi ve Tflite Donusturme

```
#!/usr/bin/env python3
```

```
# -*- coding: utf-8 -*-


import tensorflow as tf

import numpy as np

import os

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import pandas as pd

from tqdm import tqdm


def print_in_color(txt_msg, fore_tupple=(0,255,255), back_tupple=(100,100,100)):

    rf, gf, bf = fore_tupple

    rb, gb, bb = back_tupple

    msg = '{0}' + txt_msg

    mat = '\33[38;2;' + str(rf) +';' + str(gf) + ';' + str(bf) + ';48;2;' + str(rb) + ';' + str(gb) + ';' + str(bb) +
'm'

    print(msg.format(mat), flush=True)

    print('\33[0m', flush=True)


# Define the custom categorical focal loss function

def categorical_focal_loss(gamma=2.0, alpha=0.25):

    """

    Softmax aktivasyonu ile çok sınıflı odak kaybı.

    """

    def focal_loss(y_true, y_pred):

        epsilon = tf.keras.backend.epsilon()
```

```
        y_pred = tf.clip_by_value(y_pred, epsilon, 1. - epsilon)
        pt = tf.reduce_sum(y_true * y_pred, axis=-1)
        focal_loss_val = -alpha * tf.pow(1. - pt, gamma) * tf.math.log(pt)
        return tf.reduce_mean(focal_loss_val)
    return focal_loss

# Settings
MODEL_PATH = 'final_dermatology_model_FULL_v3.keras'
TFLITE_PATH = 'dermatology_model_v3.tflite'
IMG_SIZE = (224, 224)
BATCH_SIZE = 32
DATASET_DIR = '../dataset_balanced_20004'
NUM_CALIBRATION_SAMPLES = 100

# Create representative dataset for quantization
def make_representative_dataset():
    print_in_color("Preparing representative dataset for quantization...")

    # Load a small subset of the training data
    classlist = sorted([d for d in os.listdir(DATASET_DIR) if os.path.isdir(os.path.join(DATASET_DIR,
d))])
    filepaths, labels = [], []
    for klass in classlist:
        train_dir = os.path.join(DATASET_DIR, klass, 'train')
        if not os.path.exists(train_dir):
```

```
            print_in_color(f"Warning: Training directory {train_dir} not found.", fore_tupple=(255,165,0))
            continue
        flist = sorted(os.listdir(train_dir))
        for f in flist[:10]:  # Limit to 10 images per class for efficiency
            filepaths.append(os.path.join(train_dir, f))
            labels.append(klass)

    df = pd.DataFrame({'filepaths': filepaths, 'labels': labels})
    if df.empty:
        print_in_color("Error: No data found for representative dataset.", fore_tupple=(255,0,0))
        return None

    datagen =
ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input)
    gen = datagen.flow_from_dataframe(
        df,
        x_col='filepaths',
        y_col='labels',
        target_size=IMG_SIZE,
        class_mode='categorical',
        color_mode='rgb',
        shuffle=True,
        batch_size=BATCH_SIZE
    )
```

```
    def representative_data_gen():
        for _ in tqdm(range(NUM_CALIBRATION_SAMPLES // BATCH_SIZE + 1), desc="Generating representative data"):
            images, _ = next(gen)
            for image in images:
                yield [image[np.newaxis, ...].astype(np.float32)]

    return representative_data_gen

# Load the Keras model
print_in_color("Loading Keras model...")
try:
    model = tf.keras.models.load_model(MODEL_PATH, custom_objects={
        'focal_loss': categorical_focal_loss(gamma=2.0, alpha=0.25)
    })
    print_in_color(f"Model loaded successfully from {MODEL_PATH}")
except Exception as e:
    print_in_color(f"Error loading model: {str(e)}", fore_tupple=(255,0,0))
    exit(1)

# Convert to TFLite with full integer quantization
print_in_color("Converting model to TFLite with full integer quantization...")
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = make_representative_dataset()
```

```
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8

try:
    tflite_model = converter.convert()
    print_in_color("Model converted successfully")
except Exception as e:
    print_in_color(f"Error during conversion: {str(e)}", fore_tupple=(255,0,0))
    exit(1)

# Save the TFLite model
try:
    with open(TFLITE_PATH, 'wb') as f:
        f.write(tflite_model)
    print_in_color(f"TFLite model saved to {TFLITE_PATH}")
except Exception as e:
    print_in_color(f"Error saving TFLite model: {str(e)}", fore_tupple=(255,0,0))
    exit(1)

print_in_color("--- TFLite Conversion Finished ---", fore_tupple=(0,255,0))

import cv2
import numpy as np
```

```
import tensorflow as tf
import os
from lesion_filter import get_lesion_bounding_boxes

# Configuration
SCORE_THRESHOLD = 0.7  # Minimum confidence score for displaying predictions
model_path = 'dermatology_model_v3.tflite'
label_path = 'labels.txt'

# Verify files
if not os.path.exists(model_path):
    print(f"Error: {model_path} not found!")
    exit(1)
if not os.path.exists(label_path):
    print(f"Error: {label_path} not found!")
    exit(1)

# Load model
try:
    interpreter = tf.lite.Interpreter(model_path=model_path)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    print("Model loaded successfully!")
```

```
except Exception as e:
    print(f"Error loading model: {e}")
    exit(1)

# Load labels
labels = {}
with open(label_path, 'r') as f:
    for line in f:
        index, label = line.strip().split(' ', 1)
        labels[int(index)] = label

# Start webcam
cap = cv2.VideoCapture(0)  # Adjust index if needed
if not cap.isOpened():
    print("Error: Could not open webcam!")
    exit()

# Get input size
input_shape = input_details[0]['shape']
input_size = (input_shape[1], input_shape[2])  # (224, 224)
scale, zero_point = input_details[0]['quantization']

while True:
    ret, frame = cap.read()
```

```
if not ret:
    print("Error: Could not read frame!")
    break

# Preprocess frame for classification
resized_frame = cv2.resize(frame, input_size)
rgb_frame = cv2.cvtColor(resized_frame, cv2.COLOR_BGR2RGB)
input_frame = tf.keras.applications.mobilenet_v2.preprocess_input(rgb_frame)
input_frame = np.expand_dims(input_frame, axis=0)
if scale != 0:
    input_frame = (input_frame / scale + zero_point).astype(np.int8)

# Run inference
interpreter.set_tensor(input_details[0]['index'], input_frame)
interpreter.invoke()
predictions = interpreter.get_tensor(output_details[0]['index'])
if output_details[0]['quantization'][0] != 0:
    output_scale, output_zero_point = output_details[0]['quantization']
    predictions = (predictions.astype(np.float32) - output_zero_point) * output_scale

# Get prediction
predicted_index = np.argmax(predictions[0])
predicted_label = labels.get(predicted_index, 'Unknown')
score = predictions[0][predicted_index]
```

```
    # Apply threshold
    if score >= SCORE_THRESHOLD:
        print(f"Prediction: {predicted_label}, Score: {score:.2f}")
        cv2.putText(frame, f"{predicted_label}: {score:.2f}", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
        # Get and draw bounding boxes
        boxes = get_lesion_bounding_boxes(frame)
        for (x, y, w, h) in boxes:
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
    else:
        print(f"No confident prediction (Score: {score:.2f} < {SCORE_THRESHOLD})")
        cv2.putText(frame, "No confident prediction", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    cv2.imshow('Dermatology Model', frame)

    # Exit on 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Cleanup
cap.release()
cv2.destroyAllWindows()
```