



OCAK 2024

Yapay Zeka Dönem Ödevi Proje Raporu

CNN Modelleri Kullanarak Hayvan Sınıflandırması

HAZIRLAYAN

HAKAN TAHTA

204410062

HAZIRLAYAN

ABDULSAMET PALİTÇİ

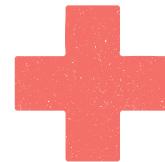
204410053

DERSİN HOCASI

Doç. Dr. KEMAL AKYOL



RAPOR İÇERİĞİ



GİRİŞ

1

Giriş kısmında proje hakkında bilgilendirme ve kullanılan teknolojiler ve projenin içeriğine dair genel bir bilgilendirme yapıldı

Kullanılan Sayfalar

2

Sonrasında ise projenin içinde kullanılan py uzantılı python file dosyalarımızın içeriği hakkında açıklamalar ve yazılan kodların anlatımı yapıldı

3

Fotoğraflar

Fotoğraflar kısmında projemizin kodlarının açıklamalı yorum satırlarının bulunduğu fotoğrafları eklendi. Ayrıca projemizi çalıştırduğumda kullanıcıyı karşılayan arayüzümüzün de fotoğrafları dahil edildi.

3

Ödevin Gereklilikleri

Son kısımda ise proje ödevinin gerekliliklerini projemizin nasıl karşıladığı, bu gereklilikleri karşılamak için projemizde neler yaptığımızı hangi yöntemleri nerede nasıl kullandığımızı ayrıntılı bir şekilde açıkladık.

GİRİŞ

Bu projede çeşitli hayvan türlerine (yengeç, yunus, penguen) ait görüntüleri sınıflandırmak için farklı derin öğrenme modelleri kullanılacaktır.

Model olarak VGG16, VGG19, ResNet50 ve proje kapsamında geliştirilen özel bir CNN modeli kullanılacaktır.

Modellere hem ham veri kümesi hem de veri artırımı uygulanmış eğitim verileri ile eğitim verilecektir.

Ayrıca model tahminlerinin ortalaması alınarak hibrit bir model oluşturulacaktır.

Yeni görüntüleri sınıflandırmak için gerçek zamanlı bir arayüz sağlanacaktır.

Ana Dosyalar:

odev.py: Tüm modellerin çalıştırılmasını, veri işlemesini ve sonuçların değerlendirilmesini sağlayan ana dosya

odevkfold.py: hayvan sınıflandırma görevi için farklı sinir ağları modeli (VGG16, VGG19, ResNet50 ve özel bir CNN modeli) oluşturmayı ve eğitmeyi amaçlar. Eğitimde Kfold doğrulama tekniği kullanılır.

ana_pencere.py: Model eğitimi ve sonuçların görüntüülendiği GUI arayüzü

islem_yapilmis_veriler.py: Artırılmış verilerin görüntüülendiği GUI

gercek_veriler.py: Ham veri kümесinin görüntüülendiği GUI

canlitest.py: Yeni görüntülerin sınıflandırılması için GUI

Veri İşleme:

Farklı hayvan türlerine ait resimler klasörlerden çekilip ön işleme tabi tutulur

Görüntüler 229x229 boyutunda normalize edilir

Etiketler kodlanır ve birlikte kodlanması yapılır

Eğitim ve test kümelerine ayırma işlemi gerçekleştirilir

Veri artırımı yapılarak daha fazla eğitim örneği elde edilir.

Model Eğitimi:

VGG16, VGG19, ResNet50 modelleri hazır ağırlıklarla yüklenir

Özel CNN modeli tanımlanır

Modeller derlenir ve artırılmış veriyle eğitilir

Epoch, batch boyutu gibi parametreler ayarlanabilir

Doğrulama doğrulu takip edilerek en iyi model kaydedilir

Değerlendirme:

Test kümesi üzerinden tahminler yapılır

Sonuçların analizi için sınıflandırma raporu üretilir

Tek tek model doğrulukları gösterilir

Hibrit Model:

Tüm model tahminlerinin ortalaması alınarak hibrit model oluşturulur

Hibrit modelin doğruluğu raporlanır

Gerçek Zamanlı Test:

Kullanıcının yeni görüntü seçebileceği GUI sağlanır

Seçilen görüntü işlenip, tahmin görüntülenir

Veri Kümesi

Projede kullanılan görüntü verileri üç farklı hayvan türüne ait resimlerden oluşmaktadır:

Yengeçler

Yunuslar

Penguenler

Her bir sınıfın yüzlerce adet görüntü elde edilmiştir. Görüntüler JPEG formatında olup, farklı boyut ve kalitede olabilmektedir.

ANA DOSYALAR KODLARIYLA BİRLİKTE AŞAĞIDA İŞLENECEKTİR

ana_pencere.py :

Kullanıcı Arayüzü Özellikleri:

- Arayüz Tasarımı:** Arayüz, farklı grup kutuları, etiketler, düğmeler ve metin alanları ile yapılandırılmıştır. Örneğin, VGG16, VGG19, ResNet50 gibi önceden eğitilmiş sinir ağlarının sonuçlarını göstermek için etiketler vardır.
- Veri Girişi:** Kullanıcılar, model eğitim parametrelerini (Epoch, Batch Size, Sabır Değeri) girebilirler. Bu parametreler, modelin nasıl eğitileceğini belirlemek için kullanılır.

- **İşlevsel Butonlar:** Arayüzde, model eğitimini başlatan, gerçek verileri gösteren, işlenmiş verileri gösteren ve canlı test sistemini başlatan butonlar bulunmaktadır.

İşlevsellik:

- **Modelleri Başlat Butonu:** Kullanıcı, belirttiği eğitim parametreleriyle birlikte model eğitimini başlatabilir. Eğitim süreci tamamlandığında, farklı sinir ağlarının sonuçları etiketlerde gösterilir.
- **Gerçek Verileri Göster Butonu:** Kullanıcı, gerçek verilere ait detayları içeren bir pencere açabilir.
- **İşlenmiş Verileri Göster Butonu:** Kullanıcı, işlenmiş verilere ait detayları içeren bir pencere açabilir.
- **Canlı Test Sistemi Butonu:** Kullanıcı, canlı test sistemini başlatmak için bu düğmeye tıklayabilir.

canlitest.py :

Bu sayfada, kullanıcının seçtiği bir görüntü dosyası üzerinde etiket tahmini yapabilen bir canlı test uygulaması geliştirilmiştir.

Bu uygulama, derin öğrenme modelinin gerçek dünya görüntülerini nasıl sınıflandıracağını anlamak için kullanılır.

Kodun Özellikleri ve İşlevselliği:

- **Veri Hazırlığı:** Önceden eğitilmiş modelin doğru bir şekilde çalışabilmesi için, farklı sınıf etiketlerine sahip görüntü dosyaları kullanılmıştır. Bu dosyalar, modelin sınıf etiketlerini tahmin edebilmesi için ön işleme adımlarından geçirilir.
- **Grafiksel Kullanıcı Arayüzü:** Tkinter kütüphanesi ile oluşturulan bir GUI, kullanıcının bir görüntü dosyası seçmesini ve bu dosya üzerinde modelin tahmin yapmasını sağlar.
- **Model Yükleme ve Tahmin:** Önceden eğitilmiş bir Keras modeli uygulamaya yüklenir. Kullanıcı bir görüntü dosyası seçtiğinde, model bu dosya üzerinde bir tahmin yapar ve sonucu ekranda gösterir.

Bu canlı test sayfasında, derin öğrenme modellerinin gerçek dünya verileriyle nasıl etkileşime gireceğini anlamak için kullanılır. Arayüzü sayesinde, kullanıcılar kolayca görüntü tahminleri yapabilir ve modelin performansını gözlemleyebilirler.

gercek_veriler.py :

Gerçek Verilerin Görüntülenmesi

Bu bölümde, kullanıcıya belirli sınıflara ait gerçek görüntü verilerini gösteren bir grafiksel kullanıcı arayüzü (GUI) oluşturulmuştur.

Bu arayüz sayesinde, farklı sınıf etiketlerine sahip gerçek dünya verileri kolayca gözlemlenebilir.

Kodun Özellikleri ve İşlevselliği:

- Grafiksel Arayüz:** PyQt5 kütüphanesi ile oluşturulan bir GUI, belirli klasörlerde bulunan görüntü dosyalarını listeler. Bu dosyalar, farklı sınıflandırma kategorilerine (Crabs, Dolphin, Penguin) aittir.
- Dosya Yollarının Toplanması:** ./girdiler/Crabs, ./girdiler/Dolphin, ve ./girdiler/Penguin klasörlerindeki uygun görüntü dosyaları belirlenir ve bu dosyaların yolları bir listeye eklenir.
- Görüntü Önizlemesi:** Her bir görüntü dosyası, GUI'de bir önizleme olarak gösterilir. Bu, kullanıcının her bir dosyaya kolayca erişebilmesini sağlar.

Bu arayüz, gerçek dünya verilerinin görsel bir şekilde incelenmesini kolaylaştırır. Kullanıcılar, belirli sınıf etiketlerine sahip görüntü dosyalarını bu arayüz üzerinden gözlemlayabilir ve analiz edebilirler.

islem_yapilmis_veriler.py :

İşlenmiş ve Çoğaltılmış Verilerin Görüntülenmesi

Bu bölümde, kullanıcıya belirli sınıflara ait işlenmiş ve çoğaltılmış görüntü verilerini gösteren bir grafiksel kullanıcı arayüzü (GUI) oluşturulmuştur. Bu arayüz sayesinde, farklı sınıf etiketlerine sahip işlenmiş ve çoğaltılmış veriler kolayca gözlemlenebilir.

Kodun Özellikleri ve İşlevselliği:

- Grafiksel Arayüz:** PyQt5 kütüphanesi ile oluşturulan bir GUI, belirli klasörlerde bulunan işlenmiş ve çoğaltılmış görüntü dosyalarını listeler. Bu dosyalar, farklı sınıflandırma kategorilerine (Crabs, Dolphin, Penguin) aittir ve işlenmiş/çoğaltılmış hallerini yansıtmaktadır.
- Dosya Yollarının Toplanması:** ./cikti/Crabs, ./cikti/Dolphin, ve ./cikti/Penguin klasörlerindeki uygun görüntü dosyaları belirlenir ve bu dosyaların yolları bir listeye eklenir.
- Görüntü Önizlemesi:** Her bir görüntü dosyası, GUI'de bir önizleme olarak gösterilir. Bu, kullanıcının her bir dosyaya kolayca erişebilmesini ve işlenmiş/çoğaltılmış hallerini görmesini sağlar.

Bu arayüz, işlenmiş ve çoğaltılmış verilerin görsel bir şekilde incelenmesini kolaylaştırır. Kullanıcılar, belirli sınıf etiketlerine sahip işlenmiş ve çoğaltılmış görüntü dosyalarını bu arayüz üzerinden gözlemlleyebilir ve analiz edebilirler.

odev-test.py :

Model Oluşturma ve Eğitim Süreci

Bu bölüm, görüntü sınıflandırma modelinin oluşturulması ve eğitilmesi için gerekli olan işlemleri içerir. Aşağıda bu sürecin ana bileşenleri ve adımları açıklanmaktadır:

1. Veri Hazırlama:

- Belirli bir klasör yapısında bulunan görüntü dosyaları yüklenir ve işlenir.
- Görüntüler belirli bir boyuta yeniden boyutlandırılır ve normalleştirilir.
- Etiketler sınıflandırma için kategorik forma dönüştürülür.
- Eğitim ve test veri kümeleri oluşturulur.

1. Veri Artırma (Data Augmentation):

- Eğitim veri setini çeşitlendirmek için görüntü artırma teknikleri kullanılır. Bu, modelin daha genel ve güçlü olmasına yardımcı olabilir.

2. Model Eğitimi:

- Farklı mimarilere (VGG16, VGG19, ResNet50) sahip derin öğrenme modelleri oluşturulur ve bu modeller eğitilir.
- Eğitim sırasında erken durdurma (early stopping) ve model kontrol noktaları (model checkpointing) kullanılır.

3. Hybrid Model Değerlendirmesi:

- Farklı modellerden oluşturulan tahminler ortalama alınarak bir hybrid model oluşturulur. Bu modelin performansı değerlendirilir.

Bu süreç, görüntü sınıflandırma problemleri için etkili ve kapsamlı bir yaklaşım sunar. Farklı mimarilere sahip modellerin eğitilmesi ve bu modellerin birleştirilmesi, modelin genel performansını artırabilir.

odevkfold.py :

Veri Hazırlığı:

- `calistirModelleriKfold()` fonksiyonu: Veri yüklenir, ön işleme yapılır ve k-fold çapraz doğrulama ile model eğitimi gerçekleştirilir.
- Veri Yükleme: Belirli bir klasör yapısında bulunan görüntü dosyaları yüklenir ve işlenir.
- Veri Artırma (Data Augmentation): Eğitim veri setini çeşitlendirmek için görüntü artırma teknikleri kullanılır. Bu, modelin daha genel ve güçlü olmasına yardımcı olabilir.

Model Oluşturma ve Eğitim:

- VGG16, VGG19 ve ResNet50 gibi önceden eğitilmiş mimarilere sahip modeller tanımlanır.
- Model mimarileri belirlendikten sonra, eğitim verisi üzerinde eğitilir ve doğrulama verisi üzerinde değerlendirilir.
- Erken durdurma ve model kontrol noktaları gibi geri çağrılar kullanılarak eğitim süreci optimize edilir.

Değerlendirme:

- Her bir modelin performansı, doğruluk, hassasiyet, özgüllük gibi metriklerle değerlendirilir.
- Ayrıca, farklı modellerin tahminlerinin ortalaması alınarak bir hibrid model oluşturulur ve bu hibrid modelin performansı da değerlendirilir.

Notlar:

- Kod, görüntü sınıflandırma için kullanılan farklı derin öğrenme mimarilerinin (VGG16, VGG19, ResNet50) k-fold çapraz doğrulama ile eğitilmesini ve değerlendirilmesini içerir.
- Veri artırma, modelin daha genel ve performanslı olmasını sağlar.
- Her bir modelin eğitim ve değerlendirme süreçleri ayrı ayrı gerçekleştirilir ve sonuçlar raporlanır.

odev.py :

Bu sayfa, derin öğrenme modeli oluşturma ve eğitme sürecini içerir. Özellikle, farklı mimarilere (VGG16, VGG19, ResNet50 ve özel bir Convolutional Neural Network) sahip modelin performansını değerlendirmek için bir dizi adımı içerir.

İşte kodun temel bileşenleri ve işlevleri:

- **Gerekli Kütüphanelerin İçe Aktarılması:**
 - Model oluşturma ve eğitme işlemleri için gerekli kütüphaneler içe aktarılır.
- **olustur_hybrid_model Fonksiyonu:**
 - Farklı modellerin tahminlerini alır ve bu tahminlerin ortalama değerlerini hesaplar.
 - Hybrid tahminler oluşturulur ve doğruluk skorunu hesaplar.
- **calistirModelleri Fonksiyonu:**
 - Veri yüklenir, ön işleme yapılır ve eğitim/test seti olarak bölünür.
 - Veri artırma (data augmentation) işlemi gerçekleştirilir.
 - VGG16, VGG19, ResNet50 ve özel bir CNN modeli için eğitim gerçekleştirilir.
 - Her model için eğitim sonrası değerlendirme yapılır ve metrikler kaydedilir.
 - Her modelin en iyi ağırlıkları kaydedilir.

- **Model Mimarileri:**
 - VGG16, VGG19, ResNet50: Önceden eğitilmiş ağırlıklarla birlikte kullanılır. Global Average Pooling katmanı ve son birkaç yoğun katman eklenir.
 - Kendi CNN: Özel bir evrişimli sinir ağının mimarisini oluştururlar.
- **Sonuçların Kaydedilmesi:**
 - Her model için performans metrikleri (doğruluk, hassasiyet, özgüllük vb.) hesaplanır ve kaydedilir.
 - En iyi ağırlıklar kaydedilir ve model performansı raporlanır.
- **Veri Artırma ve Çıktı Oluşturma:**
 - Veri artırma teknikleri kullanılarak eğitim verisi çeşitlendirilir.
 - Eğitim verisinden rastgele seçilen görüntüler çıktı olarak kaydedilir.
- **Hybrid Model Oluşturma ve Değerlendirme:**
 - Tüm modellerin tahminlerini alarak birleşik (hybrid) bir model oluşturulur ve değerlendirilir.

Kodun bu bölümü, çeşitli derin öğrenme mimarilerini eğiterek ve değerlendirecek bir sınıflandırma görevi için en iyi modeli belirlemek üzere tasarlanmıştır. Ayrıca, bu modelin performansını artırmak için farklı mimarilerin kombinasyonunu (hybrid model) da denemektedir.

PROJE SAYFALARININ AÇIKLAMALI FOTOĞRAFLARI

ana_pencere.py-(Yorum satırı fotoğrafları) :

```
# Ana pencere arayüzü sınıfı.  
class Ui_AnaDialog(object):  
    def setupUi(self, Dialog):  
        # Arayüzün genel özelliklerini tanımlıyoruz.  
        # (Palette, butonlar, etiketler, vb.)  
        Dialog.setObjectName("Dialog")  
        Dialog.resize(488, 718)  
        palette = QtGui.QPalette()  
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 247))  
        brush.setStyle(QtCore.Qt.SolidPattern)  
        palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.LinkVisited, brush)  
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 247))  
        brush.setStyle(QtCore.Qt.SolidPattern)  
        palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.LinkVisited, brush)  
        brush = QtGui.QBrush(QtGui.QColor(0, 0, 247))  
        brush.setStyle(QtCore.Qt.SolidPattern)  
        palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.LinkVisited, brush)  
        Dialog.setPalette(palette)  
        Dialog.setAutoFillBackground(True)
```

```
# Butonlara tıklanma olaylarına karşılık gelen işlevleri bağlıyoruz.  
self.testgosterbutton = QtWidgets.QPushButton(Dialog)  
self.testgosterbutton.setEnabled(True)  
self.testgosterbutton.setGeometry(QtCore.QRect(20, 650, 211, 32))  
self.testgosterbutton.setObjectName("testgosterbutton")  
self.cogaltilmisveributon = QtWidgets.QPushButton(Dialog)  
self.cogaltilmisveributon.setGeometry(QtCore.QRect(260, 650, 211, 32))  
self.cogaltilmisveributon.setObjectName("cogaltilmisveributon")  
self.label_11 = QtWidgets.QLabel(Dialog)  
self.label_11.setGeometry(QtCore.QRect(850, 30, 121, 31))  
self.label_11.setObjectName("label_11")  
self.label_12 = QtWidgets.QLabel(Dialog)  
self.label_12.setGeometry(QtCore.QRect(850, 50, 171, 221))  
self.label_12.setObjectName("label_12")  
self.groupBox = QtWidgets.QGroupBox(Dialog)  
self.groupBox.setGeometry(QtCore.QRect(20, 240, 211, 191))  
palette = QtGui.QPalette()  
brush = QtGui.QBrush(QtGui.QColor(131, 255, 10))  
brush.setStyle(QtCore.Qt.SolidPattern)  
palette.setBrush(QtGui.QPalette.Active, QtGui.QPalette.LinkVisited, brush)  
brush = QtGui.QBrush(QtGui.QColor(131, 255, 10))  
brush.setStyle(QtCore.Qt.SolidPattern)  
palette.setBrush(QtGui.QPalette.Inactive, QtGui.QPalette.LinkVisited, brush)  
brush = QtGui.QBrush(QtGui.QColor(131, 255, 10))  
brush.setStyle(QtCore.Qt.SolidPattern)  
palette.setBrush(QtGui.QPalette.Disabled, QtGui.QPalette.LinkVisited, brush)  
self.groupBox.setPalette(palette)
```

```

# Ana uygulamanın başlatılması.
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_AnaDialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())

```

canlitest.py-(Yorum satırı fotoğrafları) :

```

# Bu kod, modeli gerçek zamanlı olarak test etmek için bir arayüz sağlar.
def canliTest():
    # Önceden eğitilmiş bir Keras modeli yüklenir.
    model = load_model("model2.h5")
    resimler = []
    labeller = []

    # Ana dizin ve alt dizinlerdeki resimler taranır.
    ana_dizin = './girdiler'

    for hayvan in tqdm(os.listdir(ana_dizin)):
        gecerli_dizin = os.path.join(ana_dizin, hayvan)

        if os.path.isdir(gecerli_dizin):
            for i in range(len(os.listdir(gecerli_dizin))):
                # Resimler okunur, boyutlandırılır ve listelere eklenir.
                resim_yolu = os.path.join(gecerli_dizin, os.listdir(gecerli_dizin)[i])
                resim = cv2.imread(resim_yolu)
                resim_yeniden_boyutlandir = cv2.resize(resim, (229, 229))
                resim_yeniden_boyutlandir = resim_yeniden_boyutlandir / 255.0
                resimler.append(resim_yeniden_boyutlandir)
                labeller.append(hayvan)

```

```

# Resimleri ve etiketleri uygun formatta dönüştürür.
resimler = np.array(resimler, dtype='float32')

le = preprocessing.LabelEncoder()
le.fit(labeller)
class_isimleri = le.classes_
labeller = le.transform(labeller)

lb = LabelBinarizer()
labeller = lb.fit_transform(labeller)

# Dönüştürülen etiketleri bir dosyaya kaydeder.
with open("lb.pickle", "wb") as f:
    f.write(pickle.dumps(lb))

# TKinter kullanarak bir pencere oluşturulur.
kok = tk.Tk()
kok.title("Gerçek Zamanlı Test")

pencere_genisligi = 600
pencere_yuksekligi = 400

ekran_genisligi = kok.winfo_screenwidth()
ekran_yuksekligi = kok.winfo_screenheight()
x_kordinati = (ekran_genisligi - pencere_genisligi) / 2
y_kordinati = (ekran_yuksekligi - pencere_yuksekligi) / 2
kok.geometry(f"{pencere_genisligi}x{pencere_yuksekligi}+{int(x_kordinati)}+{int(y_kordinati)}")
liste_kutusu = tk.Listbox(kok, selectmode=tk.SINGLE, width=100, height=10)
liste_kutusu.pack(pady=10)

```

```

# Bu fonksiyon, kullanıcının seçtiği resim dosyasını yükler, model üzerinde tahminde bulunur ve sonucu gösterir.
def resmi_tahmin_et():

    # Kullanıcının seçtiği resim dosyasının yolunu alır.
    secilen_dosya = liste_kutusu.get(tk.ACTIVE)

    # Seçilen resmi belirlenen boyutlarda (229x229 piksel) yükler.
    resim = load_img(secilen_dosya, target_size=(229, 229))

    # Yüklenen resmi numpy dizisine dönüştürür ve normalize eder.
    resim = img_to_array(resim) / 255.0

    # Resmi, modelin beklediği forma uygun hale getirir.
    resim = np.expand_dims(resim, axis=0)

    # Model üzerinde tahminde bulunur ve en yüksek tahmin değerine sahip sınıfın indeksini alır.
    tahminler = model.predict(resim)
    tahmin_class_indexi = np.argmax(tahminler, axis=1)[0]
    # En yüksek tahmin değerine sahip sınıfın adını alır.
    tahmin_sinifi = class_isimleri[tahmin_class_indexi]
    # Sonuç etiketini günceller ve tahmin edilen sınıfı gösterir.
    sonuc_label.config(text=f"Görüntü Tahmin Edilen Sınıf: {tahmin_sinifi}")

    # Tahmin yapma butonunu ve onunla ilişkilendirilmiş işlevi (resmi_tahmin_et) oluşturur.
    tahmin_butonu = tk.Button(kok, text="Tahmin Yap", command=resmi_tahmin_et)
    tahmin_butonu.pack(pady=10)

    sonuc_label = tk.Label(kok, text="")
    sonuc_label.pack(pady=10)

    kok.mainloop()

```

canlitest.py-(Yorum satırı fotoğrafları) :

```
# UI tasarımını tanımlayan sınıf oluşturulur.
class Ui_Dialog_Gerçek_Veriler(object):
    def __init__(self) -> None:
        super().__init__()

    # UI bileşenlerini oluşturan fonksiyon.
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(490, 400)

        # Resimleri listelemek için bir liste widget oluşturulur.
        self.listWidget = QtWidgets.QListWidget(Dialog)
        self.listWidget.setGeometry(QtCore.QRect(20, 40, 441, 291))
        self.listWidget.setObjectName("listWidget")
        # Başlık etiketi eklenir.
        self.label = QtWidgets.QLabel(Dialog)
        self.label.setGeometry(QtCore.QRect(190, 10, 191, 21))
        self.label.setObjectName("label")
        # Belirli klasörlerdeki resim dosyaları listelenir ve liste widget'a eklenir.
        yengecлер = [os.path.join('./girdiler/Crabs', dosya) for dosya in os.listdir('./girdiler/Crabs') if dosya.lower().endswith('.png')]
        yunuslar = [os.path.join('./girdiler/Dolphin', dosya) for dosya in os.listdir('./girdiler/Dolphin') if dosya.lower().endswith('.png')]
        penguenler = [os.path.join('./girdiler/Penguin', dosya) for dosya in os.listdir('./girdiler/Penguin') if dosya.lower().endswith('.png')]

        self.listWidget.clear()
```

```
    self.listWidget.clear()
    # Resimler liste widget'a eklenirken boyutlandırılır ve simgeleri ayarlanır.
    for resim in yengecлер + yunuslar + penguenler:
        item = QListWidgetItem()
        item.setText(resim)
        pixmap = QPixmap(resim)

        boyut = QSize(300, 300)
        pixmap = pixmap.scaled(boyut)

        item.setIcon(QIcon(pixmap))
        item.setSizeHint(QSize(200, 40))
        self.listWidget.addItem(item)

    self.retranslateUi(Dialog)
    QtCore.QMetaObject.connectSlotsByName(Dialog)

def retranslateUi(self, Dialog):
    _translate = QtCore.QCoreApplication.translate
    Dialog.setWindowTitle(_translate("Dialog", "Donem Odevi | ASP & HT"))
    self.label.setText(_translate("Dialog", "Gerçek Veriler"))

# Ana uygulama başlangıç noktası.
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog_Gerçek_Veriler()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

islem_yapilmis_veriler.py-(Yorum satırı fotoğrafları) :

```
class Ui_Dialog_IslemYapilmisVeriler(object):

    # UI bileşenlerini oluşturan fonksiyon.
    def setupUi(self, Dialog):
        # Pencerenin özellikleri belirlenir.
        Dialog.setObjectName("Dialog")
        Dialog.resize(490, 400)
        # Resimleri listelemek için bir liste widget oluşturulur.
        self.listWidget = QtWidgets.QListWidget(Dialog)
        self.listWidget.setGeometry(QtCore.QRect(20, 40, 441, 291))
        self.listWidget.setObjectName("listWidget")
        # Başlık etiketi eklenir.
        self.label = QtWidgets.QLabel(Dialog)
        self.label.setGeometry(QtCore.QRect(160, 10, 191, 21))
        self.label.setObjectName("label")
        # Belirli klasörlerdeki işlenmemiş resim dosyaları listelenir ve liste widget'a eklenir.
        yengeceler = [os.path.join('./cikti/Crabs', dosya) for dosya in os.listdir('./cikti/Crabs') if dosya.lower().endswith('.png', '.j')]
        yunuslar = [os.path.join('./cikti/Dolphin', dosya) for dosya in os.listdir('./cikti/Dolphin') if dosya.lower().endswith('.png', '.j')]
        penguenler = [os.path.join('./cikti/Penguin', dosya) for dosya in os.listdir('./cikti/Penguin') if dosya.lower().endswith('.png', '.j')]

        # Resimler liste widget'a eklenirken boyutlandırılır ve simgeleri ayarlanır.
        self.listWidget.clear()
        for resim in yengeceler + yunuslar + penguenler:
            item = QListWidgetItem()
            item.setText(resim)
            pixmap = QPixmap(resim)

            boyut = QSize(300, 300)
            pixmap = pixmap.scaled(boyut)

            item.setIcon(QIcon(pixmap))
            item.setSizeHint(QSize(200, 40))
            self.listWidget.addItem(item)
```

```
# UI'nın dillerini (başlık vb.) ayarlayan fonksiyon çağırılır.
self.retranslateUi(Dialog)
QtCore.QMetaObject.connectSlotsByName(Dialog)

# UI'nın dillerini (başlık vb.) ayarlayan fonksiyon.
def retranslateUi(self, Dialog):
    _translate = QtCore.QCoreApplication.translate
    Dialog.setWindowTitle(_translate("Dialog", "Donem Odevi | ASP & HT"))
    self.label.setText(_translate("Dialog", "Cogaltilmis ve işlenmiş veriler"))

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog_IslemYapilmisVeriler()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

odev-test.py-(Yorum satırı fotoğrafları) :

```
# Hybrid modelin oluşturulması için kullanılan fonksiyon
def olustur_hybrid_model(model, model2, model3, model4, x_test_resimler, y_kat_test):
    modeller = [model, model2, model3, model4]
    tahminler = np.array([model.predict(x_test_resimler) for model in modeller])
    ortalama_tahminler = np.mean(tahminler, axis=0)
    hybrid_tahminler = np.argmax(ortalama_tahminler, axis=1)
    return accuracy_score(np.argmax(y_kat_test, axis=1), hybrid_tahminler)

# Model eğitimi için kullanılan ana fonksiyon.
def calistirModelleri(epochs, batchSize, patience):
    resimler = []
    labeller = []

    ana_dizin = './girdiler'

    for hayvan in tqdm(os.listdir(ana_dizin)):
        gecerli_dizin = os.path.join(ana_dizin, hayvan)

        if os.path.isdir(gecerli_dizin):
            for i in range(len(os.listdir(gecerli_dizin))):
                resim_yolu = os.path.join(gecerli_dizin, os.listdir(gecerli_dizin)[i])
                resim = cv2.imread(resim_yolu)
                resim_yeniden_boyutlandir = cv2.resize(resim, (229, 229))
                resim_yeniden_boyutlandir = resim_yeniden_boyutlandir / 255
                resimler.append(resim_yeniden_boyutlandir)
                labeller.append(hayvan)

    resimler = np.array(resimler, dtype='float32')

    le = preprocessing.LabelEncoder()
    le.fit(labeller)
    class_isimleri = le.classes_
    labeller = le.transform(labeller)

    x_train_coklanmis_resimler = np.concatenate((x_train_resimler, coklanmis_resimler))
    y_cat_train_coklanmis = np.concatenate((y_kat_train, coklanmis_labeller))

    cikti_dizini = "./cikti"
    if not os.path.exists(cikti_dizini):
        os.makedirs(cikti_dizini)
    for i in range(len(x_train_coklanmis_resimler)):
        boyutlandir_resim = cv2.resize(x_train_coklanmis_resimler[i], (229, 229))
        normalize_resim = cv2.normalize(boyutlandir_resim, None, 0, 1, cv2.NORM_MINMAX)

        label = y_cat_train_coklanmis[i]
        label_isim = class_isimleri[np.argmax(label)]

        yeni_dosyaismi = f"{label_isim}_{i}.jpg"

        dizin_yolu = cikti_dizini + "/" + label_isim

        if not os.path.exists(dizin_yolu):
            os.makedirs(dizin_yolu)

        cikti_dosyasi = os.path.join(dizin_yolu, yeni_dosyaismi)
        cv2.imwrite(cikti_dosyasi, (normalize_resim * 255).astype(np.uint8))

    # Eğitilen model isimleri döndürülür.
    return "vgg16ModelYazi", "vgg19ModelYazi", "resNetModelYazi", "kendiCnnYazi", "hybridModelYazi"
```

odev.py-(Yorum satırı fotoğrafları) :

```
# Uyarıları filtrelemek için gerekli kütüphaneyi kullanıyoruz.
import warnings
warnings.filterwarnings("ignore", category=UserWarning)

# Sklearn ve Keras'tan gerekli kütüphaneleri ve araçları içe aktarıyoruz.
from sklearn.metrics import accuracy_score
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.models import Sequential, Model
from keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16, VGG19, ResNet50

# Sınıflandırma raporları için gerekli metrikleri ve araçları içe aktarıyoruz.
from sklearn.metrics import classification_report

# Veri ön işleme için sklearn ve diğer gerekli kütüphaneleri içe aktarıyoruz.
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import tensorflow as tf

import logging
logging.getLogger('tensorflow').disabled = True
# Gerekli matematik ve veri işleme kütüphanelerini içe aktarıyoruz.

import numpy as np
import plotly.express as px
import cv2

# İterasyon ilerlemesini takip etmek için gerekli kütüphaneyi içe aktarıyoruz.
from tqdm import tqdm
import os
```

```
# Hybrid model için tahminlerin ortalama değerini hesaplamak için bir fonksiyon tanımlıyoruz.
def olustur_hybrid_model(model, model2, model3, model4, x_test_resimler, y_kat_test):
    modeller = [model, model2, model3, model4]
    tahminler = np.array([model.predict(x_test_resimler) for model in modeller])
    ortalama_tahminler = np.mean(tahminler, axis=0)
    hybrid_tahminler = np.argmax(ortalama_tahminler, axis=1)
    return accuracy_score(np.argmax(y_kat_test, axis=1), hybrid_tahminler)
```

```
# Model eğitiminin ve değerlendirmesini gerçekleştiren fonksiyon.
def calistirModelleri(epochs, batchSize, patience):
    resimler = []
    labeller = []

    ana_dizin = './girdiler'

    # Girdi dizinindeki her bir sınıf için veri yükleme
    for hayvan in tqdm(os.listdir(ana_dizin)):
        gecerli_dizin = os.path.join(ana_dizin, hayvan)

        if os.path.isdir(gecerli_dizin):
            for i in range(len(os.listdir(gecerli_dizin))):
                resim_yolu = os.path.join(gecerli_dizin, os.listdir(gecerli_dizin)[i])
                resim = cv2.imread(resim_yolu)
                resim_yeniden_boyutlandir = cv2.resize(resim, (229, 229))
                resim_yeniden_boyutlandir = resim_yeniden_boyutlandir / 255
                resimler.append(resim_yeniden_boyutlandir)
                labeller.append(hayvan)

    # Veriyi numpy dizisine dönüştürme
    resimler = np.array(resimler, dtype='float32')

    # Etiketlerin sayısallaştırılması
    le = preprocessing.LabelEncoder()
    le.fit(labeller)
    class_isimleri = le.classes_
    labeller = le.transform(labeller)

    labeller = np.array(labeller, dtype='uint8')
    labeller = np.resize(labeller, (len(labeller), 1))
```

```
# Eğitim ve test setlerini ayırma
x_train_resimler, x_test_resimler, y_train_labeller, y_test_labeller = train_test_split(resimler,
                                                                                         labeller,
                                                                                         test_size=0.25,
                                                                                         stratify=labeller)

# Kategorik etiketleri one-hot encoding yapma
y_kat_train = to_categorical(y_train_labeller, len(class_isimleri))
y_kat_test = to_categorical(y_test_labeller, len(class_isimleri))

# Veri artırma için ImageDataGenerator oluşturma
dataolustur = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```

# Eğitim için veri artırma
coklanmis_resimler = []
coklanmis_labeller = []

for i in range(len(x_train_resimler)):
    resim = x_train_resimler[i]
    label = y_kat_train[i]

    resim = np.reshape(resim, (1,) + resim.shape)

    for batch in dataolustur.flow(resim, batch_size= batchSize):
        coklanmis_resimler.append(batch[0])
        coklanmis_labeller.append(label)
        break

# Veri artırılmış veriyi numpy dizisine dönüştürme
coklanmis_resimler = np.array(coklanmis_resimler)
coklanmis_labeller = np.array(coklanmis_labeller)

# Eğitim verisini birleştirme
x_train_coklanmis_resimler = np.concatenate((x_train_resimler, coklanmis_resimler))
y_cat_train_coklanmis = np.concatenate((y_kat_train, coklanmis_labeller))

# Sınıfların bilgisini toplama
classlar_bilgiler = {}
classlar = sorted(os.listdir(ana_dizin))

for isim in classlar:
    class_yolu = os.path.join(ana_dizin, isim)

    if os.path.isdir(class_yolu):
        classlar_bilgiler[isim] = len(os.listdir(class_yolu))

```

```

# Erken durdurma ve model kontrolü için callback'leri tanımlama
early_stop = EarlyStopping(monitor='val_loss', patience=patience)

model_path = "model.h5"
model_checkpoint_callback = ModelCheckpoint(
    filepath=model_path,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

# VGG16 modelini oluşturma ve eğitme
tf.config.run_functions_eagerly(True)
vgg16_model = VGG16(weights='imagenet', input_shape=(229,229, 3), include_top=False)
x_train_resimler_boyutlandirildi = np.array([cv2.resize(img, (229, 229)) for img in x_train_coklanmis_resimler])

# VGG16 modelinin eğitimine başlama
vgg16_model.trainable = False # Önceden eğitilmiş ağırlıkların güncellenmesini engelleme
global_average_katmani = GlobalAveragePooling2D()(vgg16_model.output)
tahmin_katmani = Dense(len(class_isimleri), activation='softmax')(global_average_katmani)
model = Model(inputs=vgg16_model.input, outputs=tahmin_katmani)

model.compile(optimizer=tf.keras.optimizers.Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

```

```

# Modeli eğitme
model.fit(x_train_resimler_boyutlandirildi, y_cat_train_coklanmis, epochs=epochs,
           validation_data=(x_test_resimler, y_kat_test),
           batch_size=batchSize,
           callbacks=[early_stop, model_checkpoint_callback])

model.load_weights(model_path) # Model ağırlıklarını yükleme

# Test verisi üzerinde tahmin yapma
y_tahmin = model.predict(x_test_resimler)
y_tahmin_classlari = np.argmax(y_tahmin, axis=1)

y_true = np.argmax(y_kat_test, axis=1)

# Sınıflandırma raporunu oluşturma ve değerlendirme
class_sonuc_raporu = classification_report(y_true, y_tahmin_classlari, target_names=class_isimleri, output_dict=True)
basarim = class_sonuc_raporu['accuracy']
# Başarı ve diğer metrikleri yazdırma
vgg16ModelYazi = "Başarı: " + str(round(basarim, 2)) + "\n" + "Hassasiyet (Yengeç): " + str(round(class_sonuc_raporu["Crabs"]["recall"], 2))

vgg19_model = VGG19(weights='imagenet', input_shape=(229,229, 3), include_top=False)
vgg19_model.trainable = False
global_average_katmani2 = GlobalAveragePooling2D()(vgg19_model.output)
tahmin_katmani2 = Dense(len(class_isimleri), activation='softmax')(global_average_katmani2)
model2 = Model(inputs=vgg19_model.input, outputs=tahmin_katmani2)

model2.compile(optimizer=tf.keras.optimizers.legacy.Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

model2_yolu = "model2.h5"

model_checkpoint_callback2 = ModelCheckpoint(
    filepath=model2_yolu,
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

```

```

model3.fit(x_train_resimler_boyutlandirildi, y_cat_train_coklanmis, epochs=epochs,
           validation_data=(x_test_resimler, y_kat_test),
           batch_size=batchSize,
           callbacks=[early_stop, model_checkpoint_callback3])
model3.summary()

model3.load_weights(model3_yolu)

y_tahmin3 = model3.predict(x_test_resimler)
y_tahmin_classlari3 = np.argmax(y_tahmin3, axis=1)

y_true = np.argmax(y_kat_test, axis=1)

class_sonuc_raporu3 = classification_report(y_true, y_tahmin_classlari3, target_names=class_isimleri, output_dict=True)
basarim3 = class_sonuc_raporu3['accuracy']
resNetModelYazi = "Başarı: " + str(round(basarim3, 2)) + "\n" + "Hassasiyet (Yengeç): " + str(round(class_sonuc_raporu3["Crabs"]["recall"], 2))

model4 = Sequential()
model4.add(Conv2D(32, (11, 11), activation='relu', kernel_initializer='he_uniform', padding='same',
                 input_shape=(229, 229, 3)))
model4.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model4.add(MaxPooling2D((2, 2)))
model4.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model4.add(Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_uniform', padding='same'))
model4.add(MaxPooling2D((2, 2)))
model4.add(Flatten())
model4.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
model4.add(Dense(units=len(class_isimleri), activation='softmax'))

model4.compile(optimizer=tf.keras.optimizers.legacy.Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
model4.summary()
model4_yolu = "model4.h5"

```

```
y_tahmin4 = model4.predict(x_test_resimler)
y_tahmin_classlari4 = np.argmax(y_tahmin4, axis=1)

y_true = np.argmax(y_kat_test, axis=1)

class_sonuc_raporu4 = classification_report(y_true, y_tahmin_classlari4, target_names=class_isimleri, output_dict=True)
basarim4 = class_sonuc_raporu4['accuracy']
kendiCnnYazi = "Başarı%: " + str(round(basarim4, 2)) + "\n" + "Hassasiyet (Yengeç): " + str(round(class_sonuc_raporu4["Crabs"])[
cikti_dizini = "./cikti"
if not os.path.exists(cikti_dizini):
    os.makedirs(cikti_dizini)

for i in range(len(x_train_coklanmis_resimler)):
    boyutlandir_resim = cv2.resize(x_train_coklanmis_resimler[i], (229, 229))
    normalize_resim = cv2.normalize(boyutlandir_resim, None, 0, 1, cv2.NORM_MINMAX)

    label = y_cat_train_coklanmis[i]
    label_isim = class_isimleri[np.argmax(label)]

    yeni_dosyaismi = f"{label_isim}_{i}.jpg"

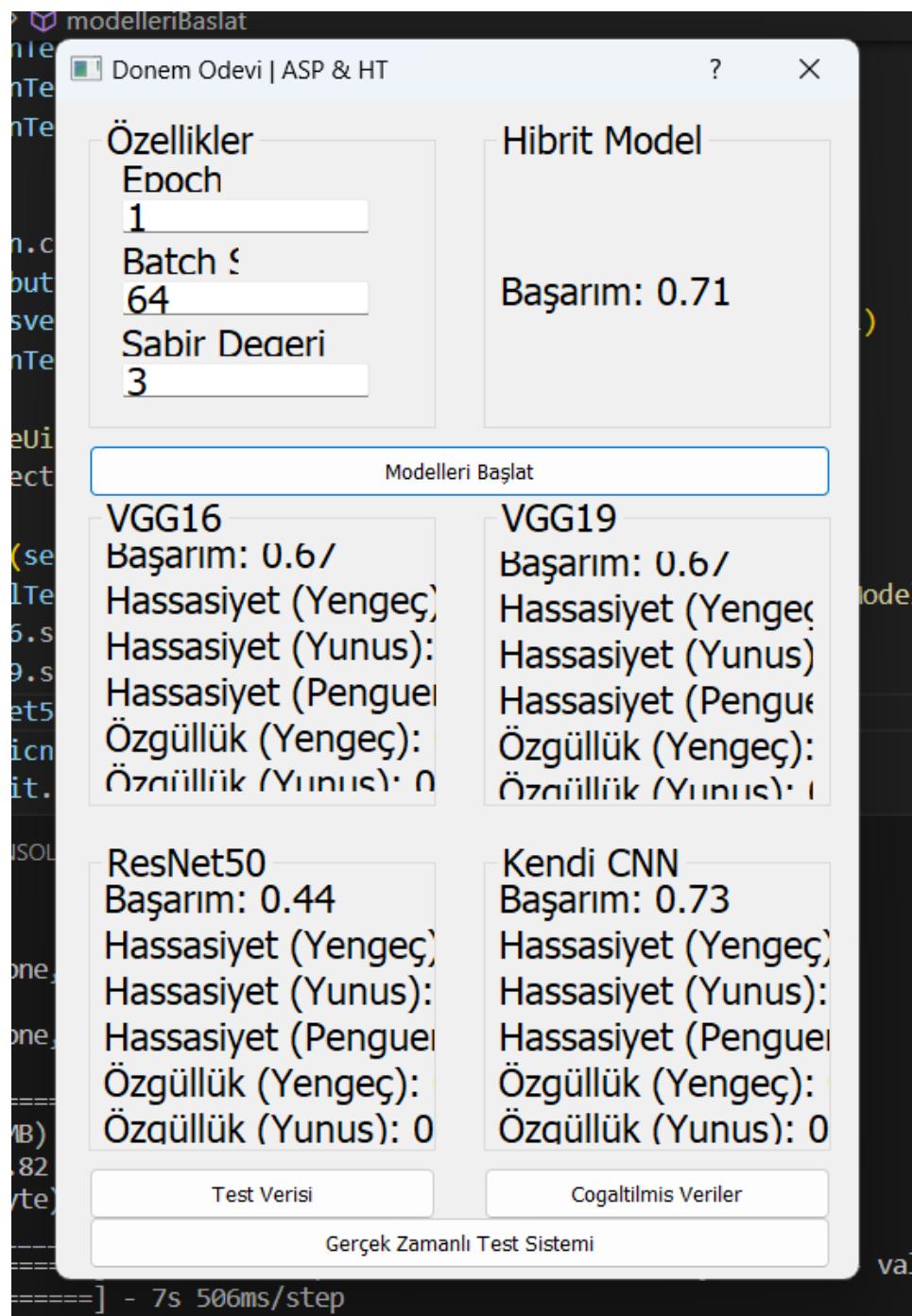
    dizin_yolu = cikti_dizini + "/" + label_isim

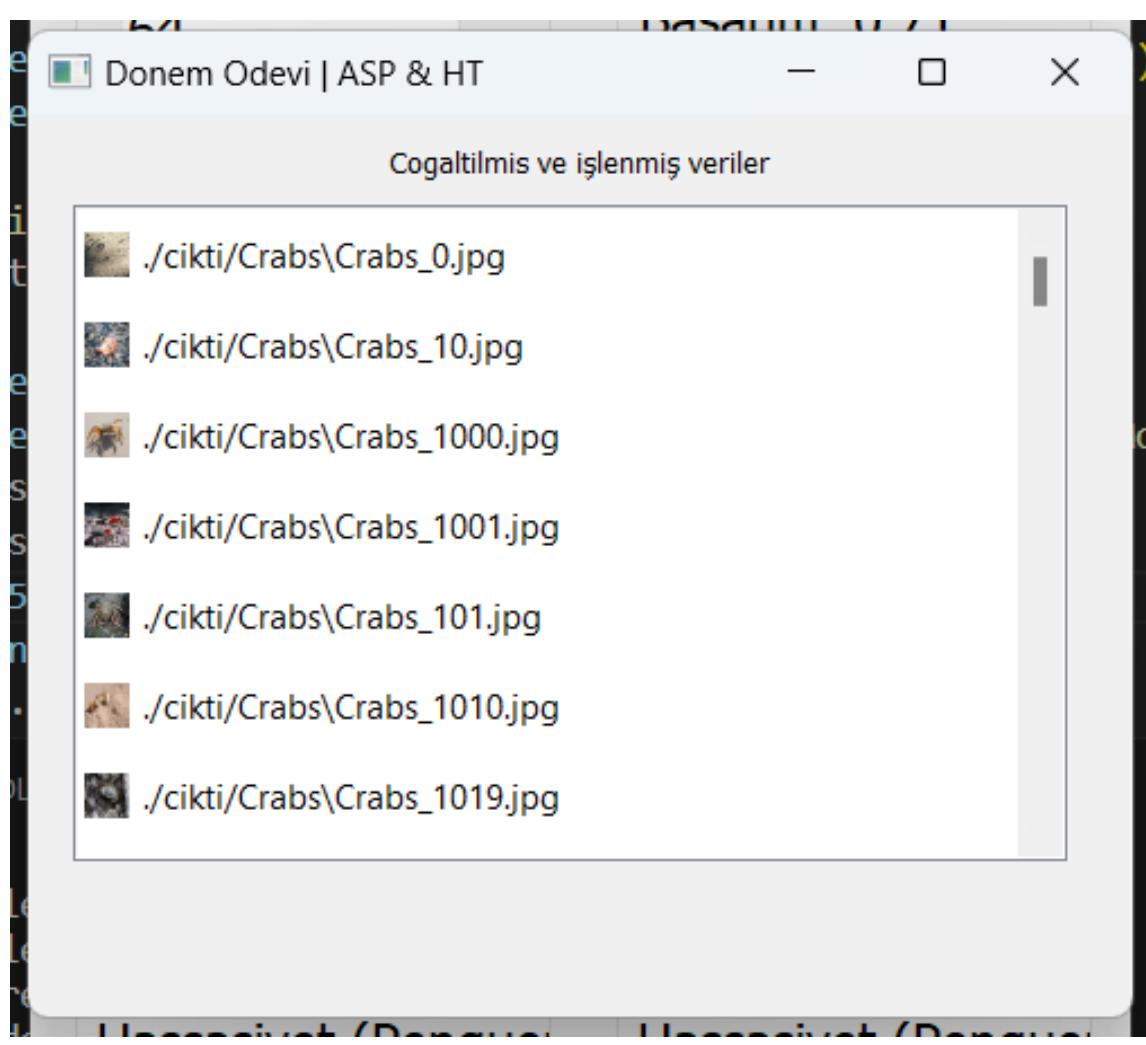
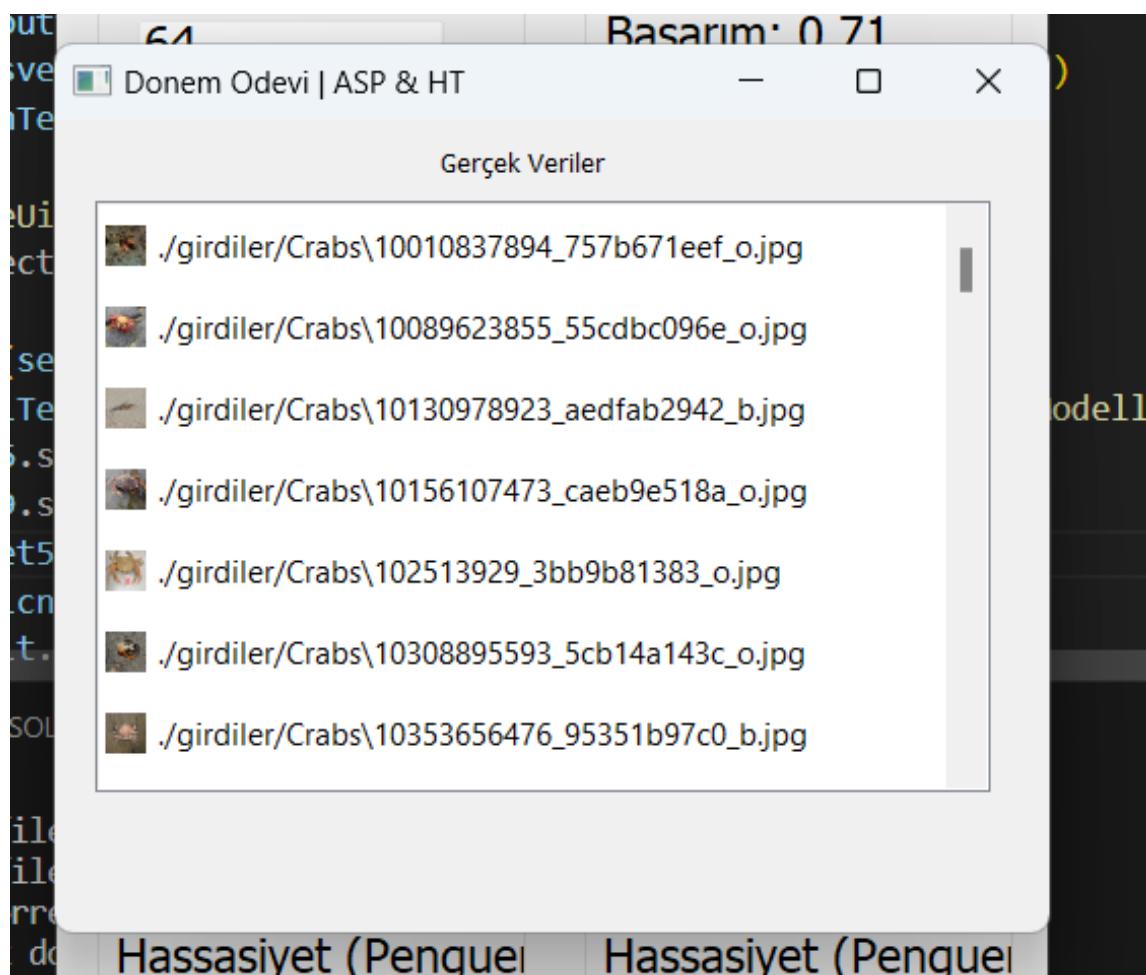
    if not os.path.exists(dizin_yolu):
        os.makedirs(dizin_yolu)

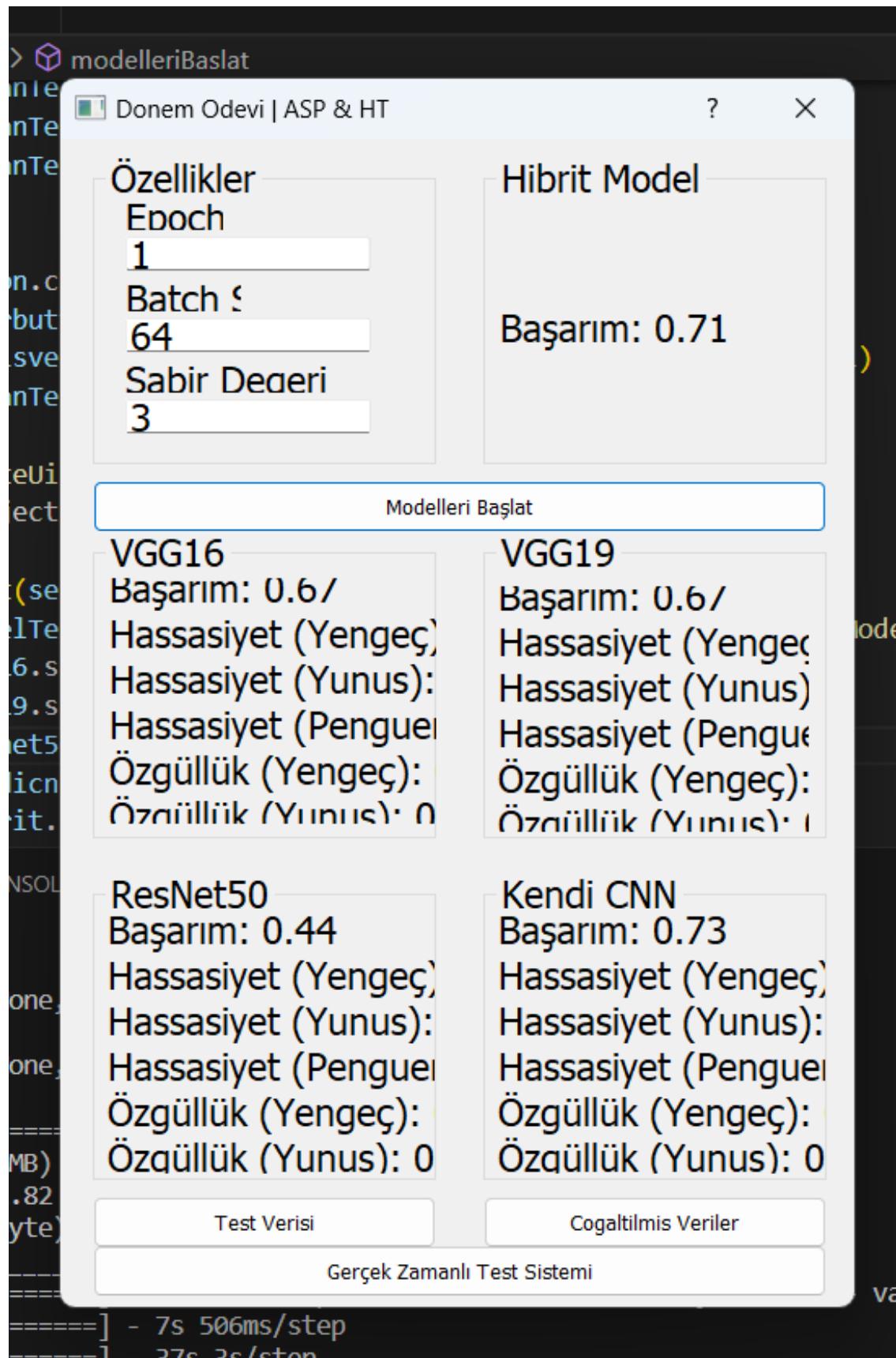
    cikti_dosyasi = os.path.join(dizin_yolu, yeni_dosyaismi)
    cv2.imwrite(cikti_dosyasi, (normalize_resim * 255).astype(np.uint8))

# Son olarak, tüm modellerden tahminleri alarak hibrid bir model oluşturma ve değerlendirme
hybridModelBasarim = olustur_hybrid_model(model, model2, model3, model4, x_test_resimler, y_kat_test)
hybridModelYazi = "Başarı%: " + str(round(hybridModelBasarim, 2))
return vgg16ModelYazi, vgg19ModelYazi, resNetModelYazi, kendiCnnYazi, hybridModelYazi
```

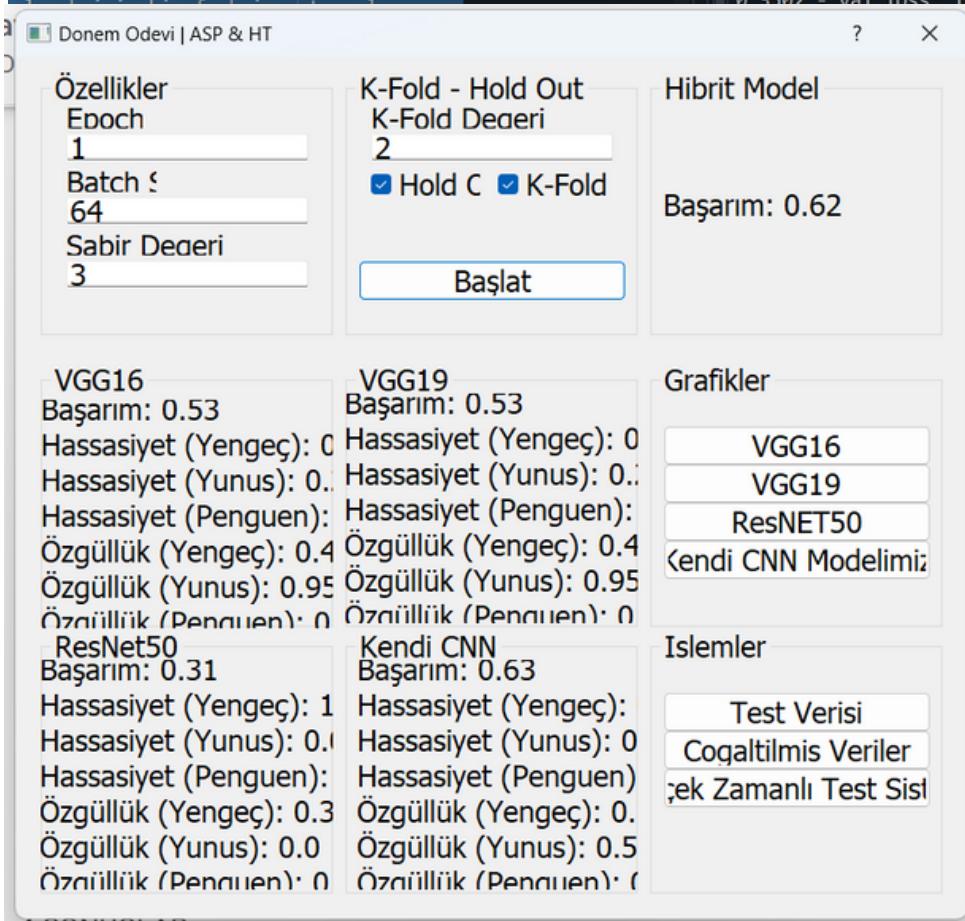
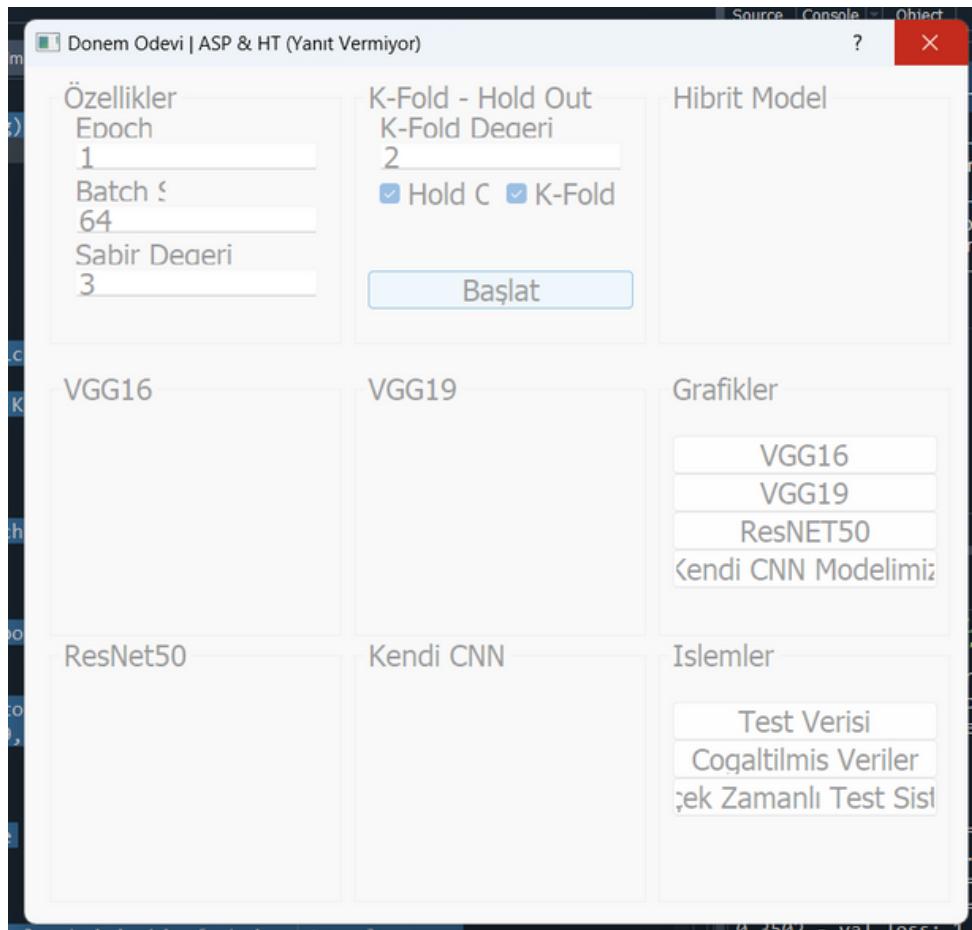
Arayüz Fotoğrafları (Eklemelerden Önceki Halleri)

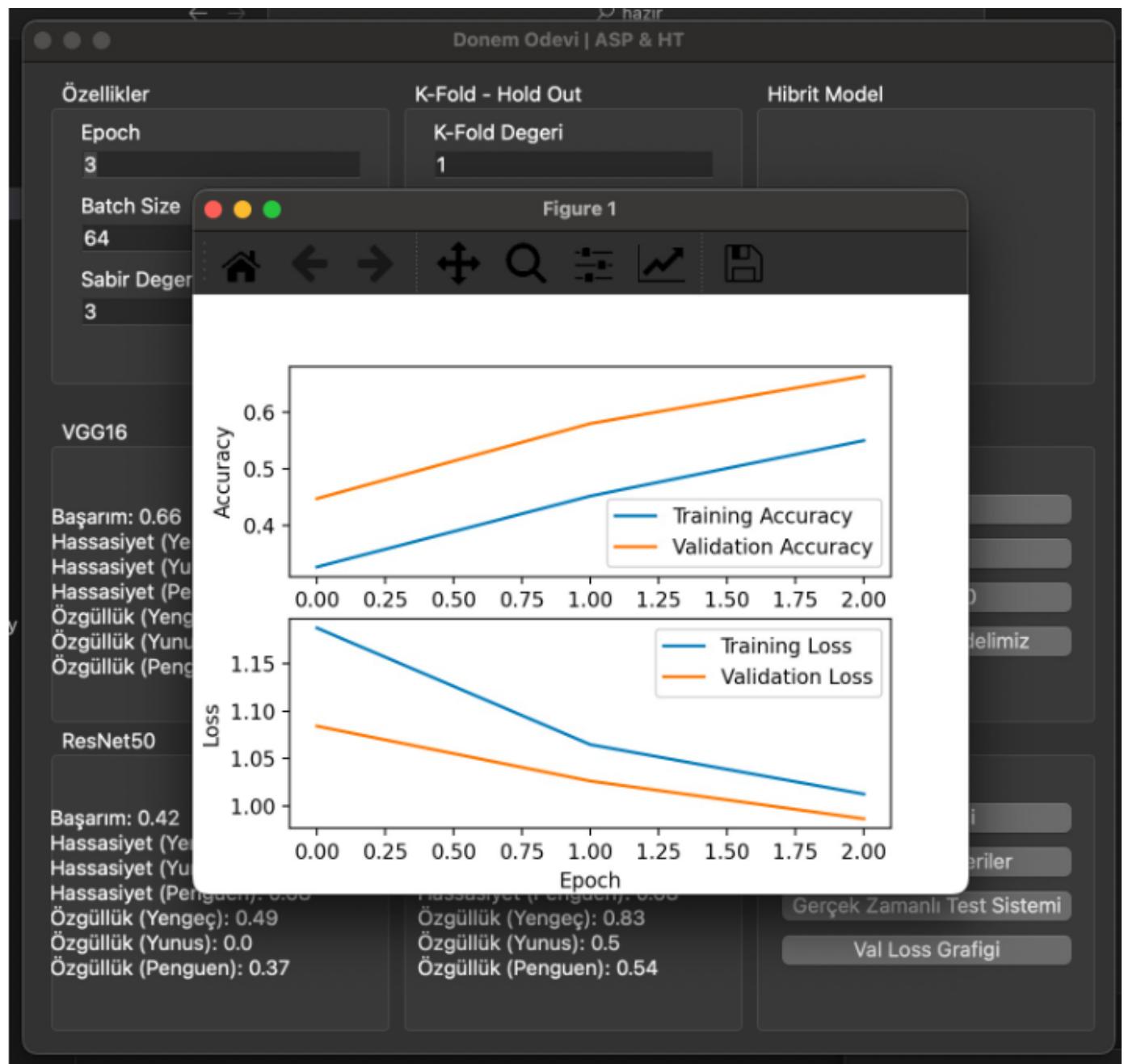






Eklenen Değişiklikler Sonrası Arayüzdeki Değişim ile Birlikte Fotoğrafları (K-Fold ve Hold Out) + (Grafik Eklemeleri)





Projede Geçen Bazı İşlemlerin Açıklamaları

VERİ ÇOKLAMA İŞLEMİ :

Veri çoklama Keras ImageDataGenerator sınıfı kullanılarak gerçekleştirildi. DataGenerator nesnesi oluşturulup resimlere uygulanan çeşitli dönüşümler ayarlandı:

Rotation_range: Resimleri 20 dereye kadar döndürme

Width/Height_shift_range: Resimlerde yatay/dikey kaydırma (%20 oranında)

Shear_range: Kaydırma açısını değiştirme (%20)

Zoom_range: Yakınlaştırma (%20)

Horizontal_flip: Yatay yansıtma

Bunun ardından eğitim verisi olan x_train_resimler tek tek bu data generatora verilerek çok sayıda artırılmış resim elde ettik.

Eğitim verisini böylece orijinalinden çok daha büyük bir hale getirerek, modelin daha genelleşebilir olmasını sağladık.

TRAIN VALIDATION TEST İŞLEMİ :

Projede verileri train-validation-test şeklinde bölüp kullandık.
odev.py dosyasında şu satırlarda :
resimler ve labeller listelerinden oluşturulan verisetini
train_test_split ile eğitim, doğrulama ve test kümelerine böldük.

x_train_resimler, x_val_resimler, x_test_resimler gibi
değişkenlere atadık.

Aynı şekilde y_train_labeller, y_val_labeller, y_test_labeller
değişkenlerini de oluşturduk.

Model eğitiminde bu kümeler şu şekilde kullanılıyor:

Model x_train_resimler ve y_train_labeller ile eğitiliyor.

Her epoch sonunda x_val_resimler kullanılarak doğrulama skoru
hesaplanıyor.

Model eğitiminin durması için doğrulama skorunda belirli bir
düşüş koşulu aranıyor.

Sonunda eğitim tamamlandıktan sonra test skoru
x_test_resimler kullanılarak ölçülüyor.

Yani projede train-validation-test bölünmesi ve kullanımı yaptık.

MODEL EĞİTİM İŞLEMLERİ :

Projede model eğitim işlemleri için kullanılan bazı ana parametreler şu şekildedir:

I. epoch değeri:

odev.py'de model.fit() fonksiyonunda epochs parametresine ana_pencere.py üzerinden girilen epoch değeri geçiliyor.

II. batch size değeri:

Model fit işleminde batch_size parametresine ana_pencere.py'den alınan batch boyutu değeri yazılıyor.

III. Early stopping:

EarlyStopping callback'i tanımlanıyor ve patience değeri olarak ana_pencere.py'den alınan sabır değeri kullanılıyor.

Doğrulama skorunda bu kadar düşüş olursa eğitim duruyor.

IV. Model checkpoint:

Model hafızasına en iyi durumu kaydetmek için ModelCheckpoint callback'i tanımlanıyor.
Dolayısıyla bu temel parametre değerleri kullanıcıdan alınarak eğitim parametrelenebilir hale getirilmiş. Böylece farklı değerler ile hızlı testler yapılabiliyor..

MODELİN EĞİTİM SONUÇLARI VE İSTENEN DİĞER ÖZELLİKLER :

I. Eğitim sonuçları:

odev.py dosyasında model.fit() fonksiyonu kullanılarak model eğitilmekte.

Her epoch sonunda doğruluk skorları print edilerek eğitim ilerlemesi izlenmektedir.

II. Karışıklık matrisi sonuçları ve doğruluk-duyarlılık-özgülük metrikleri:

Model.evaluate() kullanılarak test verileriyle modelin performansı ölçülmekte.

Skorlar print edilmekte ve metrik değerleri döndürülmektedir.

III. Real time çalışan test sistemi:

canlitest.py dosyasında Tkinter kullanılarak bir arayüz tasarlandı.

Kullanıcı resim yükleme butonuyla resim seçebiliyor.

Seçilen resim tahmin ediliyor ve sonuç ekranda gösteriliyor.

Böylece kullanıcı canlı olarak modeli test edebiliyor.

Ayrıca Keras uygulamalarından 3 model ve kendi CNN'nin eğitim sonuçları karşılaştırılarak en iyi model seçilebiliyor.

Dolayısıyla tüm bu metrikler ve canlı test özelliği projede kullanıldı. Verimli bir şekilde model geliştirme süreci izlenebiliyor.

DOĞRULUK VE KAYIP GRAFİKLERİ (VAL LOSS GRAPHICS):

ana_pencere.py dosyasında gosterValLossGrafigi adlı bir fonksiyon tanımlanmıştır. Bu fonksiyon matplotlib kütüphanesi kullanılarak validation loss grafiğini çizdirir.

Fonksiyon eğitim ve validation loss değerlerini almak için ana_pencere sınıfından val_loss_epoch, train_loss ve val_loss değişkenlerini kullanır.

Bu değişkenler model eğitimi sırasında odev.py/odevkfold.py dosyalarında tutulan verileri içerir.

Grafik çiziminde validation loss değerleri mavi renk, eğitim loss değerleri ise siyah noktalar kullanılarak gösterilir.

Grafiğin x eksenini epoch değerlerini, y eksenini ise loss değerlerini temsil etmektedir.

Böylece farklı modellerin eğitim ve validation performansları karşılaştırılabilmektedir.

K-Fold ve Hold Out (Sonradan eklenen kısımlardan):

Model eğitimleri için hem holdout hem de k-fold doğrulama yöntemleri kullanılabilir.

Holdout/k-fold seçimi ana_pencere.py dosyasındaki groupBox_7 içindeki checkbox'lardan yapılır.

Eğer kfold_checkbox seçili ise modeller odevkfold.py dosyasında, aksi halde odev.py dosyasında eğitilir.

odevkfold.py dosyası model eğitimini k-fold Cross Validation yöntemiyle gerçekleştirir. Veriler k katlama ayrıştırılır.

odev.py ise geleneksel holdout yöntemini kullanarak verileri eğitim ve test kümelerine ayırır.

Böylece kullanıcı isteğine göre farklı doğrulama yöntemleri denenebilir.

VERİ SETİ LİNKİ

<https://www.kaggle.com/datasets/vencerlanz09/sea-animals-image-dataste>

**DOLPHİN PENGUİN CRABS
Kullanılmıştır**