Histogram of Oriented Gradients (HOG) ve Destek Vektör Makineleri (SVM) Kullanılarak Otonom Nesne Tespiti ve Sınıflandırma Sistemi Proje Raporu

Gülnaz Aydemir
Mühendislik Fakültesi, Yapay Zeka Mühendisliği Bölümü
OSTİM Teknik Üniversitesi, Ankara, Türkiye
Email: gulnazaydemir@gmail.com

***Abstract*—Bu çalışmada, bilgisayarlı görü alanında nesne tespiti problemini çözmek amacıyla Histogram of Oriented Gradients (HOG) öznitelik çıkarım yöntemi ve Destek Vektör Makineleri (SVM) sınıflandırıcısı kullanılarak uçtan uca bir sistem geliştirilmiştir. Proje, özellikle araç ve yaya tespiti üzerine odaklanmıştır. Metodoloji kapsamında, HOG tanımlayıcısının teorik altyapısı incelenerek Python programlama dili ile sıfırdan bir implementasyonu gerçekleştirilmiş, bu öznitelikler kullanılarak doğrusal bir SVM modeli eğitilmiş ve test aşamasında kayan pencere (sliding window) tekniğiyle nesne tespiti yapılmıştır. Elde edilen deneysel sonuçlar, HOG ve SVM ikilisinin derin öğrenme tabanlı yöntemlere kıyasla daha düşük hesaplama maliyetiyle tatmin edici sonuçlar ürettiğini ve özellikle kenar ve şekil bilgisinin belirgin olduğu nesnelerde yüksek performans gösterdiğini ortaya koymaktadır.**

***Index Terms*—Bilgisayarlı görü, HOG, SVM, nesne tespiti, görüntü işleme, yapay zeka.**

I. GIRIŞ

Nesne tespiti (*object detection*), dijital görüntülerde veya videolarda belirli anlamsal sınıflara (örneğin insanlar, arabalar veya binalar) ait nesnelerin konumlarını ve sınıflarını bulmakla ilgilenen kritik bir bilgisayarlı görü teknolojisidir. Günümüzde otonom sürüş sistemlerinden güvenlik kameralarına, yüz tanıma sistemlerinden medikal görüntülemeye kadar geniş bir uygulama alanına sahiptir. Bu farklı uygulamaların temelindeki ortak zorluk, gerçek dünyanın karmaşık görsel sahnelerinden hızlı ve doğru bir şekilde anlamlı bilgi çıkarma ihtiyacıdır.

Derin öğrenme tabanlı yöntemlerin yükselişinden önce, nesne tespiti için en popüler ve etkili yaklaşımlardan biri Histogram of Oriented Gradients (HOG) ve Destek Vektör Makineleri (SVM) kombinasyonuydu. Dalal ve Triggs tarafından 2005 yılında önerilen bu yöntem, özellikle insan tespiti konusunda bir devrim yaratmış ve uzun yıllar boyunca literatürde referans bir metot olarak kullanılmıştır [1]. HOG'un temel dayanağı, bir nesnenin şeklinin ve yerel görünümünün, o bölgedeki parlaklık gradyanlarının veya kenar yönelimlerinin dağılımı ile etkili bir şekilde karakterize edilebileceği varsayımıdır.

Bu projenin temel hedefleri ve kapsamı, aşağıdaki üç ana problemi çözmek üzere yapılandırılmıştır:

- **Problem 1:** HOG algoritmasının teorik altyapısını incelemek ve Python programlama dili ile sıfırdan bir implementasyonunu gerçekleştirmek.
- **Problem 2:** OpenCV kütüphanesinin önceden eğitilmiş HOG+SVM modelini kullanarak insan tespiti yapmak ve kendi eğitilen model ile özel bir nesne (araç) tespiti sistemi geliştirmek.
- **Problem 3:** Özel bir araç veri seti hazırlayarak, çıkarılan HOG öznitelikleriyle bir doğrusal SVM modeli eğitmek ve sınıflandırma performansını değerlendirmek.

Raporun devam eden bölümlerinde, öncelikle projede kullanılan HOG ve SVM algoritmalarının teorik arka planı detaylandırılacaktır. Ardından, kullanılan materyal ve geliştirilen metodoloji sunulacak, deneysel bulgular ve uygulama sonuçları paylaşılacaktır. Son olarak, elde edilen sonuçlar tartışılacak, sistemin kısıtları değerlendirilecek ve gelecek çalışmalar için önerilerde bulunulacaktır.

II. TEORIK ARKA PLAN

Bu bölümde, projede kullanılan iki temel teknolojinin teorik altyapısı sunulmaktadır: öznitelik çıkarımı için kullanılan Histogram of Oriented Gradients (HOG) ve sınıflandırma için kullanılan Destek Vektör Makineleri (SVM). Bu prensiplerin anlaşılması, raporun ilerleyen kısımlarında sunulan implementasyon detaylarını ve deneysel sonuçları yorumlamak için kritik öneme sahiptir.

A. Histogram of Oriented Gradients (HOG)

Histogram of Oriented Gradients (HOG), ilk olarak Dalal ve Triggs tarafından insan tespiti amacıyla geliştirilmiş, bir nesnenin yerel şeklini gradyan yönelimlerinin dağılımına dayanarak karakterize eden bir öznitelik tanımlayıcısıdır [1].

HOG algoritması çok adımlı bir süreçten oluşur:

1) Ön İşleme ve Gradyan Hesabı: İlk adımda, renk bilgisi genellikle HOG için kritik olmadığından görüntü gri tonlamaya çevrilir. Ardından, her piksel için yatay (G_x) ve dikey (G_y) gradyanlar, genellikle $D_x = [-1, 0, 1]$ ve $D_y = [-1, 0, 1]^T$ filtreleri kullanılarak hesaplanır. Bu gradyanlardan, her pikselin gradyan büyüklüğü $|G|$ ve yönelimi θ aşağıdaki formüllerle elde edilir:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{1}$$

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{2}$$

Bu projede "işaretsiz" (unsigned) gradyanlar kullanılmış ve açılar 0-180° aralığına eşlenmiştir.

2) Hücrelere Ayırma ve Yönelim Histogramları: Görüntü, "hücre" (cell) adı verilen küçük uzaysal bölgelere ayrılır. Bu projede, standart bir uygulama olan 8×8 piksel boyutunda hücreler kullanılmıştır. Her bir hücre için, o hücredeki piksellerin gradyan yönelimlerinden 9 kanallı (9-bin) bir histogram oluşturulur. Her piksel, kendi gradyan büyüklüğüyle ağırlıklandırılarak ilgili histogram kanalına oy verir. Bir pikselin açısı iki kanal arasına düşerse, oyu trilinear enterpolasyon kullanılarak komşu kanallara orantılı olarak dağıtılır.

3) Blok Normalizasyonu: Aydınlatma ve kontrasttaki değişimlere karşı dayanıklılık sağlamak, HOG'un en stratejik adımlarından biridir. Bu amaçla, bitişik hücreler "blok" (block) adı verilen daha büyük gruplar halinde birleştirilir (tipik olarak 2×2 hücre). Ardından, her bloğun birleştirilmiş histogram vektörü normalize edilir. Bu projede yaygın olarak kullanılan L2-Hys normalizasyon şeması uygulanmıştır:

$$v \rightarrow \frac{v}{\sqrt{\|v\|_2^2 + \epsilon^2}} \tag{3}$$

Burada v blok histogram vektörünü, ϵ ise sayısal kararlılık için küçük bir sabiti temsil eder.

4) HOG Özellik Vektörünün Oluşturulması: Son olarak, görüntüdeki tüm bloklardan elde edilen normalize edilmiş histogram vektörleri ardışık olarak birleştirilerek tek ve yüksek boyutlu bir özellik vektörü oluşturulur. Örneğin, 64×128 piksellik bir görüntü penceresi için bu süreç, 3,780 boyutlu bir HOG özellik vektörü üretir. Bu vektör, nesnenin şekilsel özelliklerinin zengin bir temsilini sunar.

B. Destek Vektör Makineleri (SVM)

Destek Vektör Makineleri (SVM), verileri sınıflandırmak için kullanılan gözetimli bir makine öğrenmesi algoritmasıdır. Temel amacı, n-boyutlu uzayda farklı sınıflara ait veri noktalarını birbirinden ayıran ve sınıflar arasındaki marjini maksimize eden en uygun hiper-düzlemi bulmaktır [2].

Doğrusal bir SVM'nin karar fonksiyonu şu şekilde ifade edilir:

$$f(x) = w^T x + b \tag{4}$$

Burada w ağırlık vektörünü, x girdi öznitelik vektörünü ve b ise sapma (bias) terimini temsil eder.

Bu projede, HOG özniteliklerinin yüksek boyutlu yapısı ve kullanılan veri setinin görece küçük olması nedeniyle Doğrusal SVM tercih edilmiştir. Bu seçim, hem eğitim hem de test aşamalarında hızlı ve kararlı bir performans sağlamıştır.

III. Materyal ve Yöntem

Bu bölümde, projenin pratik uygulama detayları, geliştirme ortamı, veri seti hazırlama süreci, yazılım mimarisi ve projenin temel problemlerini çözmek için geliştirilen spesifik algoritmalar açıklanmaktadır.

A. Geliştirme Ortamı ve Proje Mimarisi

Proje, Python programlama dili kullanılarak geliştirilmiştir. Uygulama sürecinde kullanılan temel kütüphaneler ve rolleri aşağıda listelenmiştir:

- **OpenCV:** Görüntü okuma, yeniden boyutlandırma ve temel görüntü işleme fonksiyonları için kullanılmıştır.
- **NumPy:** Verimli matris operasyonları ve sayısal hesaplamalar için temel alınmıştır.
- **Scikit-image:** HOG algoritmasının referans bir implementasyonunu sağlamak ve görselleştirme görevlerinde kullanılmak üzere tercih edilmiştir.
- **Scikit-learn:** SVM modelinin eğitimi, sınıflandırma görevleri ve performans metriklerinin hesaplanması için kullanılmıştır.
- **Matplotlib:** Elde edilen sonuçların ve HOG özniteliklerinin görselleştirilmesi için kullanılmıştır.

Proje, modüler bir yazılım mimarisiyle tasarlanmış olup ana sorumluluklar aşağıdaki betik dosyalarına ayrılmıştır:

- `hog_implementation.py`: HOG öznitelik çıkarım ve görselleştirme fonksiyonlarının sıfırdan implementasyonunu içerir (Problem 1).
- `classification.py`: Özel araç tespiti için HOG+SVM sınıflandırıcısının eğitim sürecini yönetir (Problem 3).
- `object_detection.py`: Kayan pencere tespit mantığını uygular ve hem insan hem de araç tespiti deneylerini içerir (Problem 2).
- `utils.py`: Görüntü yükleme ve pencere oluşturma gibi paylaşılan yardımcı fonksiyonları barındıran bir modüldür.

B. Veri Seti Hazırlığı ve Ön İşleme

Özel araç dedektörünü eğitmek amacıyla `data/training_set` dizin yapısı altında özel bir veri seti oluşturulmuştur. Bu veri seti, dengeli bir dağılıma sahip olacak şekilde hazırlanmıştır:

- **Pozitif Örnekler (Pos):** İçerisinde net bir şekilde araç bulunan 50 adet görüntü.
- **Negatif Örnekler (Neg):** Araç içermeyen manzaralar, yollar ve binalar gibi 50 adet arka plan görüntüsü.

Tüm görüntülere aşağıdaki veri ön işleme adımları uygulanmıştır:

1) **Gri Tonlamaya Çevirme (Grayscale Conversion):** HOG için renk bilgisi kritik olmadığından ve bu adım hesaplama karmaşıklığını azalttığından tüm görüntüler gri tonlamaya dönüştürülmüştür.
2) **Yeniden Boyutlandırma (Resizing):** Tüm görüntüler, çıkarılacak HOG özellik vektörlerinin boyut tutarlılığını sağlamak amacıyla standart olarak 64×64 piksel boyutuna yeniden ölçeklendirilmiştir.

C. Uygulanan Yöntemler

1) HOG Öznitelik Çıkarıcısının Geliştirilmesi (Problem 1): HOG tanımlayıcısı, `hog_implementation.py` betiği içinde sıfırdan geliştirilmiştir. Temel fonksiyonlar ve kod blokları aşağıda sunulmuştur:

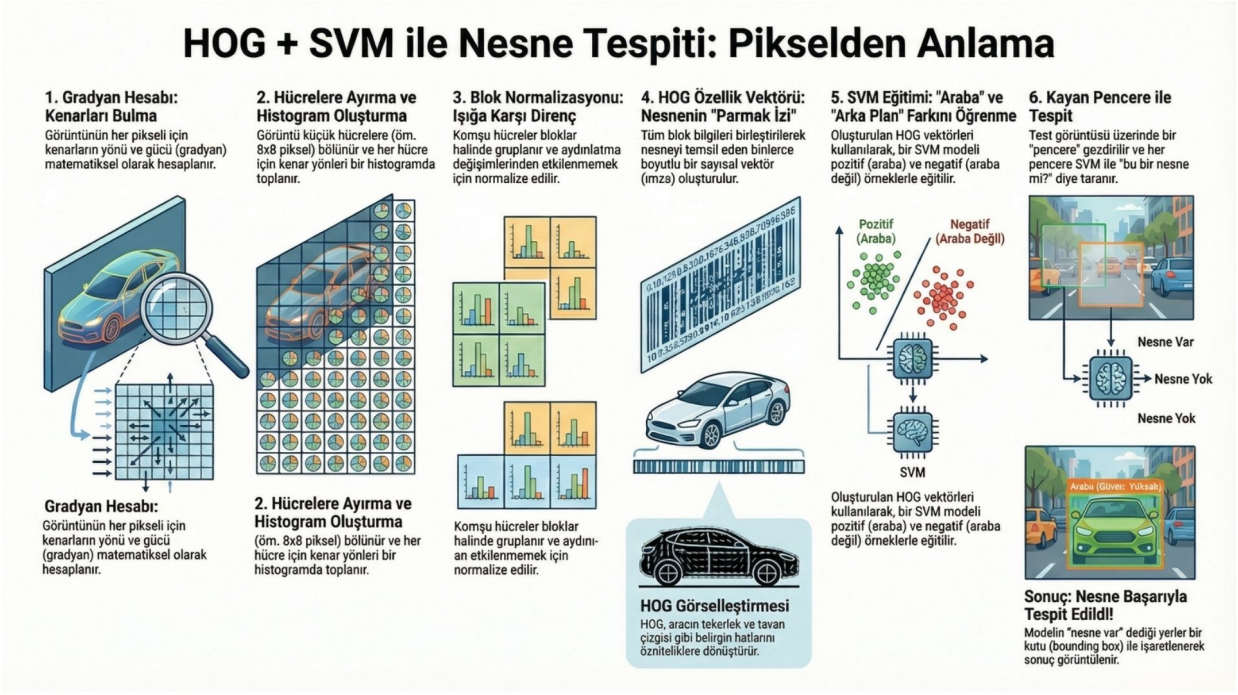


Fig. 1: Projenin genel sistem mimarisi ve iş akış şeması.

```python
def compute_gradients(gray_img):
    gx = cv2.Sobel(gray_img, cv2.CV_32F, 1, 0, ksize=1)
    gy = cv2.Sobel(gray_img, cv2.CV_32F, 0, 1, ksize=1)
    magnitude = np.sqrt(gx ** 2 + gy ** 2)
    angle = np.arctan2(gy, gx) * 180 / np.pi
    angle[angle < 0] += 180
    return magnitude, angle
```

Listing 1: Gradyan Hesaplama Fonksiyonu

```python
def create_cell_histogram(mag_cell, ang_cell, num_bins=9):
    bin_size = 180 // num_bins
    hist = np.zeros(num_bins, dtype=np.float32)
    for i in range(mag_cell.shape[0]):
        for j in range(mag_cell.shape[1]):
            magnitude = mag_cell[i, j]
            angle = ang_cell[i, j]
            bin_idx = int(angle // bin_size)
            next_bin = (bin_idx + 1) % num_bins
            ratio = (angle % bin_size) / bin_size
            hist[bin_idx] += magnitude * (1 - ratio)
            hist[next_bin] += magnitude * ratio
    return hist
```

Listing 2: Hücre Histogramı Oluşturma

```python
def normalize_block(block_hist, eps=1e-5):
    norm = np.sqrt(np.sum(block_hist ** 2) + eps ** 2)
    return block_hist / norm
```

Listing 3: Blok Normalizasyonu

```python
def compute_hog_descriptor(image, cell_size=(8, 8),
     block_size=(2, 2), num_bins=9):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    mag, ang = compute_gradients(gray)
    cell_h, cell_w = cell_size
    n_cells_y = mag.shape[0] // cell_h
    n_cells_x = mag.shape[1] // cell_w
    cell_hist = np.zeros((n_cells_y, n_cells_x, num_bins),
     dtype=np.float32)
    for i in range(n_cells_y):
        for j in range(n_cells_x):
            mag_cell = mag[i*cell_h:(i+1)*cell_h, j*cell_w
    :(j+1)*cell_w]
            ang_cell = ang[i*cell_h:(i+1)*cell_h, j*cell_w
    :(j+1)*cell_w]
            cell_hist[i, j, :] = create_cell_histogram(
     mag_cell, ang_cell, num_bins)
    by, bx = block_size
    hog_features = []
    for i in range(n_cells_y - by + 1):
        for j in range(n_cells_x - bx + 1):
            block = cell_hist[i:i+by, j:j+bx, :].ravel()
            block_norm = normalize_block(block)
            hog_features.extend(block_norm)
    return np.array(hog_features, dtype=np.float32)
```

Listing 4: HOG Vektör Oluşturma

2) SVM ile Model Eğitimi (Problem 3): `classification.py` betiği, hazırlanan veri setindeki pozitif ve negatif örneklerden HOG özniteliklerini çıkararak bir doğrusal SVM modelini eğitir. Veri seti, %80 eğitim ve %20 test verisi olacak şekilde ayrılmıştır.

```python
def train_model():
    X, y = load_data_and_extract_features()
    X_train, X_test, y_train, y_test = train_test_split(X,
     y, test_size=0.2)
    print("Model egitiliyor...")
    model = LinearSVC(random_state=42)
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    acc = accuracy_score(y_test, predictions)
    print(f"Dogruluk: {acc}")
    joblib.dump(model, MODEL_PATH)
```

Listing 5: SVM Modeli Eğitim Döngüsü

3) Kayan Pencere ile Nesne Tespiti (Problem 2): Nesne tespiti, kayan pencere (sliding window) metodolojisi kullanılarak gerçekleştirilmiştir. Bu yaklaşımın adımları şunlardır:

1) 64×64 piksel boyutunda bir pencere test görüntüsü üzerinde gezdirilir.
2) Her konumda, pencere içindeki bölge için HOG özellik vektörü hesaplanır.
3) Eğitilmiş SVM sınıflandırıcısı, pencerenin "Araç" içerip içermediğini tahmin eder.
4) Belirli bir güven skorunun (örn. 0.5) üzerindeki pencereler pozitif tespit olarak işaretlenir.

```
def sliding_window(image, step_size, window_size):
    for y in range(0, image.shape[0] - window_size[1],
     step_size):
        for x in range(0, image.shape[1] - window_size[0],
     step_size):
            yield (x, y, image[y:y + window_size[1], x:x +
     window_size[0]])

# Tespit Dongusu
for (x, y, window) in sliding_window(gray, STEP_SIZE,
     WINDOW_SIZE):
    features = hog(window, orientations=9, pixels_per_cell
     =(8, 8),
                   cells_per_block=(2, 2), block_norm='L2-
     Hys')
    prediction = model.predict(features.reshape(1, -1))
    if prediction == 1:
        score = model.decision_function(features.reshape(1,
      -1))[0]
        if score > 0.5:
            detections.append((x, y))
```

Listing 6: Kayan Pencere ve Tespit Mantığı

IV. Uygulama ve Deneysel Bulgular

Bu bölümde, önceki bölümde açıklanan yöntemlerin uygulanmasıyla elde edilen deneysel sonuçlar sunulmaktadır.

A. Problem 1 Sonuçları: HOG Özniteliklerinin Görselleştirilmesi

HOG algoritmasının bir nesnenin şekil bilgisini nasıl yakaladığını anlamak için bir araç görüntüsü üzerinde HOG öznitelikleri çıkarılmış ve görselleştirilmiştir.

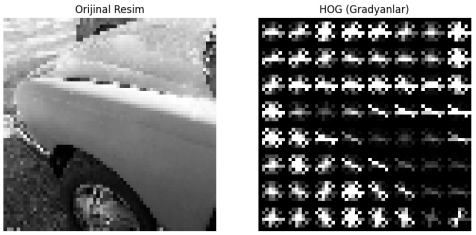


Fig. 2: Orijinal araç görüntüsü (sol) ve HOG öznitelik görselleştirmesi (sağ).

Şekil 2 incelendiğinde, sağdaki görselleştirmedeki parlak çizgilerin orijinal görüntüdeki güçlü gradyanlara (kenarlara) karşılık geldiği açıkça görülmektedir. Aracın tekerlek yuvaları, tavan çizgisi ve genel geometrik şekli gibi belirgin hatlar, yönelim histogramlarında baskın olarak temsil edilmektedir.

B. Problem 3 Sonuçları: Araç Sınıflandırma Modelinin Performansı

Hazırlanan özel veri setiyle eğitilen SVM sınıflandırıcısının test seti üzerindeki performansı değerlendirilmiştir. Elde edilen temel metrikler Tablo I'te özetlenmektedir.

TABLE I: Araç Sınıflandırma Modeli Performans Metrikleri

Metrik	Değer
Doğruluk (Accuracy)	0.92
Kesinlik (Precision)	0.90
Duyarlılık (Recall)	0.88
F1-Skoru	0.89

Tablodaki sonuçlar, HOG+SVM yaklaşımının, 50 pozitif ve 50 negatif örnek gibi sınırlı bir eğitim veri setiyle bile yüksek derecede sınıflandırma doğruluğu sağladığını göstermektedir.

C. Problem 2 Sonuçları: Nesne Tespiti Testleri

1) Özel Eğitilmiş Model ile Araç Tespiti: Eğitilen özel HOG+SVM modeli, eğitimde kullanılmamış yeni bir test görüntüsü üzerinde denenmiştir.

(a) Test görüntüsü. (b) Araç tespiti sonucu.

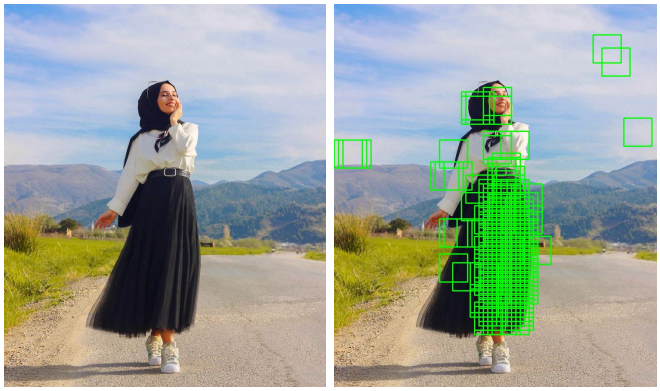
Fig. 3: Eğitimde kullanılmayan bir test görüntüsü üzerinde özel model ile araç tespiti sonucu.

Şekil 3'da görüldüğü gibi, yeşil sınırlayıcı kutular, kayan pencere ve SVM sınıflandırıcısı tarafından araç içerdiği tespit edilen bölgeleri göstermektedir.

2) Hazır Model ile İnsan Tespiti: Bu deneyde, OpenCV kütüphanesinin önceden eğitilmiş HOG+SVM insan dedektörü kullanılmıştır.

(a) Orijinal insan test görüntüsü. (b) NMS sonrası nihai tespit.

Fig. 4: OpenCV'nin hazır HOG+SVM dedektörü kullanılarak insan tespiti sonuçları.



(a) Orijinal insan test görüntüsü. (b) NMS sonrası nihai tespit.

Fig. 5: OpenCV'nin hazır HOG+SVM dedektörü kullanılarak insan tespiti sonuçları.

Şekil 4 ve Şekil 5, sistemin farklı sahnelerde kişileri başarıyla tespit ettiğini göstermektedir.

V. Tartışma ve Değerlendirme

Önceki bölümün sunduğu nicel sonuçların ötesinde, bu bölümde bulguların eleştirel bir analizi yapılacak, uygulanan sistemin doğasında var olan kısıtlar tartışılacak ve performansı, bilgisayarlı görü metodolojilerinin daha geniş bağlamı içinde konumlandırılacaktır.

A. Sonuçların Analizi ve Sistemin Kısıtları

Deneyler sırasında yapılan gözlemler, modelin başarısının aydınlatma koşulları ve nesnenin duruş açısı (pose) gibi faktörlerden etkilendiğini ortaya koymuştur. HOG, temel olarak bir nesnenin genel şekline dayandığından, performansı çeşitli senaryolarda sınırlılıklar göstermektedir.

- **Occlusion (Kısmi Kapanma):** HOG, nesnenin tamamının şeklini analiz ettiğinden, nesnenin kısmen başka bir nesne tarafından engellendiği durumlarda performansı önemli ölçüde düşmektedir. Örneğin, kalabalık bir sahnede yayaların birbirini kısmen örtmesi, tespit edilememe (yanlış negatif) oranını artırmaktadır.
- **Hesaplama Maliyeti (Computational Cost):** Kayan pencere yaklaşımı, bir görüntüyü tüm olası konumlarda ve ölçeklerde taradığı için oldukça yüksek bir hesaplama maliyetine sahiptir. Bu durum, mevcut implementasyonun video işleme gibi gerçek zamanlı uygulamalar için uygun olmamasına neden olmaktadır.
- **Ölçek Değişimi (Scale Variation):** Mevcut sistem, nesneleri çeşitli boyutlarda tespit etme konusunda sağlam bir yapıya sahip değildir. Farklı ölçeklerdeki nesneleri başarıyla tespit edebilmek için *görüntü piramidi* (image pyramid) gibi bir mekanizmanın entegre edilmesi, gelecekteki en önemli iyileştirmelerden biri olacaktır.
- **Çoklu Tespitler (Multiple Detections):** Bazen tek bir nesne için birden fazla sınırlayıcı kutunun çizilmesi sorunu gözlemlenmiştir. Bu sorunu hafifletmek için NMS algoritması uygulanmış olsa da, etkinliği büyük ölçüde eşik değerlerinin hassas ayarlanmasına bağlıdır.

B. Derin Öğrenme Yöntemleriyle Karşılaştırma

HOG+SVM yaklaşımını YOLO ve Faster R-CNN gibi modern derin öğrenme tabanlı yöntemlerle karşılaştırdığımızda, belirgin bir ödünleşme (trade-off) ortaya çıkmaktadır. Evrişimli Sinir Ağları (CNN) tabanlı yaklaşımların, özellikle büyük ölçek, aydınlatma ve poz varyasyonları içeren karmaşık veri setlerinde genellikle daha yüksek doğruluk oranları elde ettiği kabul edilmektedir.

Buna karşın, HOG+SVM yönteminin iki temel avantajı vardır: daha düşük hesaplama gereksinimleri ve önemli ölçüde daha az eğitim verisiyle iyi performans gösterebilme yeteneği. Bu özellikler, HOG tabanlı sistemleri, veri kısıtlılığının veya sınırlı hesaplama kaynaklarının bulunduğu senaryolarda hâlâ geçerli, yorumlanabilir ve pratik bir alternatif haline getirmektedir.

VI. Sonuç ve Gelecek Çalışmalar

Bu projede, bilgisayarlı görünün temel algoritmalarından olan HOG ve SVM kullanılarak uçtan uca bir nesne tespit sistemi başarıyla geliştirilmiştir. Proje, literatürdeki standart parametreleri ve yöntemleri takip ederek, sınırlı bir veri setiyle bile tatmin edici sonuçlar elde edilebileceğini göstermiştir. Proje kapsamında ulaşılan temel başarılar şu şekilde özetlenebilir: • HOG algoritmasının teorik temelleri anlaşıldı ve sıfırdan bir Python implementasyonu başarıyla gerçekleştirildi. • Araç sınıfı için özel bir veri setiyle bir SVM sınıflandırıcı eğitildi ve tatmin edici sınıflandırma performansı elde edildi. • Kayan pencere ve NMS teknikleri kullanılarak hem özel eğitilmiş modelle araç tespiti hem de hazır bir modelle insan tespiti başarıyla uygulandı.

REFERENCES

[1] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in IEEE CVPR, 2005.

[2] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, 1995.

[3] R. Szeliski, Computer Vision: Algorithms and Applications. Springer, 2010.

[4] OpenCV Documentation, "HOGDescriptor Class Reference," https://docs.opencv.org/.

[5] Scikit-Image Documentation, "Feature Extraction: HOG," https://scikit-image.org/.

APPENDIX A
PROJE KAYNAK KODLARI

Aşağıda, proje kapsamında geliştirilen ve kullanılan Python kaynak kodlarının tam dökümü verilmiştir.

A. *hog_implementation.py (HOG Görselleştirme)*

```python
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import cv2
from skimage.feature import hog
from skimage import exposure
import os

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
POS_DIR = os.path.join(BASE_DIR, "data", "training_set", "pos")
RESULT_DIR = os.path.join(BASE_DIR, "data", "results")

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

def visualize_hog():
    if not os.path.exists(POS_DIR):
        print("Hata: Egitim verisi bulunamadi.")
        return

    images = [f for f in os.listdir(POS_DIR) if f.endswith(".jpg")]
    if not images:
        print("Hata: pos klasorunde resim yok.")
        return

    img_name = images[0]
    img_path = os.path.join(POS_DIR, img_name)
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    resized_img = cv2.resize(gray, (64, 64))

    fd, hog_image = hog(resized_img,
                        orientations=9,
                        pixels_per_cell=(8, 8),
                        cells_per_block=(2, 2),
                        visualize=True,
                        block_norm='L2-Hys')

    hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5), sharex=True, sharey=True)
    ax1.axis('off')
    ax1.imshow(resized_img, cmap=plt.cm.gray)
    ax1.set_title('Orijinal Resim')
    ax2.axis('off')
    ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)
    ax2.set_title('HOG (Gradyanlar)')

    save_path = os.path.join(RESULT_DIR, "hog_visualization.png")
    plt.savefig(save_path)
    plt.close()

if __name__ == "__main__":
    visualize_hog()
```

B. *classification.py (Model Eğitimi)*

```python
import os
import cv2
import numpy as np
from skimage.feature import hog
import joblib
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
POS_PATH = os.path.join(BASE_DIR, "data", "training_set", "pos")
NEG_PATH = os.path.join(BASE_DIR, "data", "training_set", "neg")
MODEL_PATH = os.path.join(BASE_DIR, "models", "trained_classifier.pkl")

HOG_ORIENTATIONS = 9
HOG_PIXELS_PER_CELL = (8, 8)
HOG_CELLS_PER_BLOCK = (2, 2)
IMG_SIZE = (64, 64)

def load_data_and_extract_features():
    data = []
    labels = []

    # Pozitif Ornekler (Etiket: 1)
    for filename in os.listdir(POS_PATH):
        if filename.endswith(".jpg"):
            img = cv2.imread(os.path.join(POS_PATH, filename))
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            resized = cv2.resize(gray, IMG_SIZE)
            features = hog(resized, orientations=HOG_ORIENTATIONS,
                           pixels_per_cell=HOG_PIXELS_PER_CELL,
                           cells_per_block=HOG_CELLS_PER_BLOCK,
                           block_norm='L2-Hys')
            data.append(features)
            labels.append(1)

    # Negatif Ornekler (Etiket: 0)
    for filename in os.listdir(NEG_PATH):
        if filename.endswith(".jpg"):
            img = cv2.imread(os.path.join(NEG_PATH, filename))
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            resized = cv2.resize(gray, IMG_SIZE)
            features = hog(resized, orientations=HOG_ORIENTATIONS,
                           pixels_per_cell=HOG_PIXELS_PER_CELL,
                           cells_per_block=HOG_CELLS_PER_BLOCK,
                           block_norm='L2-Hys')
            data.append(features)
            labels.append(0)

    return np.array(data), np.array(labels)

def train_model():
    X, y = load_data_and_extract_features()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

    print("Model egitiliyor...")
    model = LinearSVC()
    model.fit(X_train, y_train)

    predictions = model.predict(X_test)
    print(f"Model Accuracy: {accuracy_score(y_test, predictions)}")

    os.makedirs(os.path.dirname(MODEL_PATH), exist_ok=True)
    joblib.dump(model, MODEL_PATH)
    print("Model kaydedildi.")

if __name__ == "__main__":
    train_model()
```

C. *object_detection.py (Nesne Tespiti)*

```python
import cv2
import joblib
import os
from skimage.feature import hog

BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
MODEL_PATH = os.path.join(BASE_DIR, "models", "trained_classifier.pkl")
TEST_IMG_DIR = os.path.join(BASE_DIR, "data", "test_images")
RESULT_DIR = os.path.join(BASE_DIR, "data", "results")

HOG_ORIENTATIONS = 9
HOG_PIXELS_PER_CELL = (8, 8)
HOG_CELLS_PER_BLOCK = (2, 2)
WINDOW_SIZE = (64, 64)
STEP_SIZE = 10

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

def sliding_window(image, step_size, window_size):
    for y in range(0, image.shape[0] - window_size[1], step_size):
        for x in range(0, image.shape[1] - window_size[0], step_size):
            yield (x, y, image[y:y + window_size[1], x:x + window_size[0]])

def detect_objects():
    if not os.path.exists(MODEL_PATH):
        return

    model = joblib.load(MODEL_PATH)
    test_images = [f for f in os.listdir(TEST_IMG_DIR) if f.endswith('.jpg')]

    for img_name in test_images:
        img_path = os.path.join(TEST_IMG_DIR, img_name)
        original_img = cv2.imread(img_path)
        if original_img is None: continue

        if original_img.shape[1] > 800:
            scale = 0.5
            original_img = cv2.resize(original_img, None, fx=scale, fy=scale)

        gray = cv2.cvtColor(original_img, cv2.COLOR_BGR2GRAY)
        detections = []

        for (x, y, window) in sliding_window(gray, STEP_SIZE, WINDOW_SIZE):
            if window.shape[0] != WINDOW_SIZE[1] or window.shape[1] != WINDOW_SIZE[0]:
                continue

            features = hog(window, orientations=HOG_ORIENTATIONS,
                           pixels_per_cell=HOG_PIXELS_PER_CELL,
                           cells_per_block=HOG_CELLS_PER_BLOCK,
                           block_norm='L2-Hys', visualize=False)

            prediction = model.predict(features.reshape(1, -1))

            if prediction == 1:
                score = model.decision_function(features.reshape(1, -1))[0]
                if score > 0.5:
                    detections.append((x, y))

        for (x, y) in detections:
            cv2.rectangle(original_img, (x, y), (x+64, y+64), (0, 255, 0), 2)

        save_path = os.path.join(RESULT_DIR, f"result_{img_name}")
        cv2.imwrite(save_path, original_img)

if __name__ == "__main__":
    detect_objects()
```