|  |  |  |
| --- | --- | --- |
|  | **T.C.**  **KASTAMONU ÜNİVERSİTESİ**  **MÜHENDİSLİK VE MİMARLIK FAKÜLTESİ**  **BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI** |  |

**YAPAY ZRKA VE UZMAN SİSTEMLER**

**DEEP LEARNING İLE YAPILAN MODEL PROJESİ**

**HAZIRLAYAN**

194410021- Arman Soylu

194410009 Oğuz Berk Aydın

**DANIŞMAN**

**Doç. Dr. Kemal AKYOL**

OCAK - 2024

**KASTAMONU**

**T.C.**

**KASTAMONU ÜNİVERSİTESİ**

**MÜHENDİSLİK VE MİMARLIK FAKÜLTESİ**

**ÖZET**

**Proje**

**DEEP LEARNING ve ARAYÜZ PROJESİ**

**HAZIRLAYANLAR**

**Arman Soylu**

**Oğuz Berk Aydın**

**Kastamonu Üniversitesi**

**Mühendislik ve Mimarlık Fakültesi**

**Bilgisayar Mühendisliği Bölümü**

**Bitirme Projesi Danışmanı/Danışmanları:**

**Doç. Dr. Kemal AKYOL**

Kaggle’dan seçtiğimiz Vahşi hayattaki 10 kedi veri setimizi Kerastan hazır model olarak CNN’ler yardımı ile Vahşi hayattakı 10 kedi veri setimizin modellerini oluşturduk bunun dışında kendi CNN’imizi yaptık ve bunları bir arayüzde hem eğitim hem de gerçek dünya test aşamasında gerçekleştirmiş olduk.Veri setimizi ilk başta temizlemeden işlem yaptığımız için eğitimde sıkıntılar yaşadık ama bozuk resimleri temizledikten sonra eğitim aşaması çok verimli geçti.

[1.0 Giriş 4](#_Toc156252159)

[1.2 Modeller 4](#_Toc156252160)

[1.3 EfficientNetB0 Eğitim 4](#_Toc156252161)

[1.3.1 Kütüphaneler 4](#_Toc156252162)

[1.3.2 Konfigürasyonları Belirleme 6](#_Toc156252163)

[1.3.3 Veri Ön İşleme ve Veri Çoklama 6](#_Toc156252164)

[1.3.4 Modelin Tanımlanması ve Derlenmesi (EfficientNetB0): 7](#_Toc156252165)

[1.3.5 Eğitim için Geri Çağırımların Tanımlanması 8](#_Toc156252166)

[1.3.6 Modelin Eğitilmesi 8](#_Toc156252167)

[1.3.7 Eğitim Geçmişinin Çizdirilmesi 8](#_Toc156252168)

[1.3.8 Modelin Test Kümesinde Değerlendirilmesi 9](#_Toc156252169)

[1.3.9 Değerlendirme Metriklerinin Gösterilmesi 9](#_Toc156252170)

[1.3.9.1 Karışıklık Matrisi ve Sınıflandırma Raporu 9](#_Toc156252171)

[1.3.9.2 Eğitilmiş Modelin ve Ağırlıkların Kaydedilmesi 9](#_Toc156252172)

[1.4 Xception ve Resnet eğitimleri 10](#_Toc156252173)

[1.5 Custom CNN 10](#_Toc156252174)

[2.0 Uygulama Geliştirme Ortamı 11](#_Toc156252175)

[2.1 Test 11](#_Toc156252176)

[2.1.1 Arayüz 12](#_Toc156252177)

[2.1.2 Resim seçme metodu 12](#_Toc156252178)

[2.1.3 Test İşleminin gerçekleştiği metotlar 13](#_Toc156252179)

[2.1.4 Hibrit Test İşlemi 14](#_Toc156252180)

[2.1.5 Başlatma şablonu 15](#_Toc156252181)

[2.2 Eğitim 15](#_Toc156252182)

[2.3 Eğitim Ve Test Birleştirilmesi 16](#_Toc156252183)

[2.4 Test ve Eğitim Arayüzü 17](#_Toc156252184)

[3.0 Test Sonuçları 18](#_Toc156252185)

[3.1 Xception ve Efficient 18](#_Toc156252186)

[3.2 Resnet 19](#_Toc156252187)

[3.3 Custom CNN 20](#_Toc156252188)

[4.0 Veri seti ve modeller 21](#_Toc156252189)

1.0 Giriş

Bu projenin amacı, doğada bulunan farklı türdeki 10 tane büyük kediyi doğru bir şekilde sınıflandırmak ve keras application üzerinden alınmış 3 tane önceden eğitilmiş derin öğrenme modeli kullanılarak modellerinin bu görevdeki performanslarını karşılaştırmaktır. Daha sonra custom bir CNN yapısı oluşturularak tekrar bir sınıflandırma işlemi yapılacak olup sonuçları karşılaştırılacaktır. Derin öğrenme, görsel tanıma görevlerinde son yıllarda önemli başarılar elde etmiştir ve bu proje, belirli bir alan - yaban hayatı tanıma - üzerinde bu teknolojinin etkinliğini araştırmayı amaçlamaktadır.

1.2 Modeller

-**ResNet**: 50 katmanlı bir residual network modelidir.

-**EfficientNetB0**: Ölçeklenebilirlik ve performans dengesi için tasarlanmıştır.

-**Xception**: Derinlik öncelikli ayrılabilir evrişimler kullanır.

-**Özelleştirilmiş CNN**: Birden çok evrişim, havuzlama ve yoğun katmanı ile uygun bir CNN oluşturmak.

1.3 EfficientNetB0 Eğitim

Bu aşamada Keras Application’dan indirilen 3 hazır modeli kullanarak modeller eğitilecektir. Her modelin eğitimi hemen hemen aynı olsa dahi ufak tefek değişiklikler yer almaktadır.

1.3.1 Kütüphaneler

import tensorflow as tf  
from keras.preprocessing.image import ImageDataGenerator  
from keras.applications.efficientnet import EfficientNetB0, preprocess\_input  
from keras.models import Sequential  
from keras.layers import Dense, GlobalAveragePooling2D  
from keras.optimizers import Adam  
from keras.callbacks import EarlyStopping  
import matplotlib.pyplot as plt  
import numpy as np  
from sklearn.metrics import confusion\_matrix, classification\_report  
import seaborn as sns

**1. TensorFlow (tf):** Açık kaynaklı bir makine öğrenimi ve derin öğrenme kütüphanesidir. Yüksek performanslı sayısal hesaplamalar için kullanılır ve genellikle derin öğrenme modellerinin oluşturulması, eğitimi ve dağıtımı için tercih edilir.

**2. ImageDataGenerator (from keras.preprocessing.image):** Görüntü verilerini artırmak için kullanılan bir Keras sınıfıdır. Eğitim sırasında veri çeşitliliğini artırmak amacıyla kullanılır, bu da modelin genelleştirme yeteneğini artırabilir.

**3. EfficientNetB0 (from keras.applications.efficientnet):** Keras kütüphanesinde bulunan ve EfficienNet mimarisine dayalı önceden eğitilmiş bir derin öğrenme modelidir. Düşük hesaplama maliyetiyle yüksek performans sağlamak amacıyla tasarlanmıştır.

**4. Sequential (from keras.models):** Keras'ta bulunan model oluşturma API'sinin temel sınıfıdır. Model katmanlarını sıralı bir şekilde eklemek için kullanılır.

**5. Dense (from keras.layers):** Tam bağlantılı (fully connected) katmanı oluşturmak için kullanılır. Bu katman, girişten çıkışa tüm nöronları içerir.

**6. GlobalAveragePooling2D (from keras.layers):** 2D veri kümesinde küresel ortalama havuzlama işlemi gerçekleştiren bir katmandır. Genellikle önceki katmanlardan gelen özellik haritalarını düzleştirmek için kullanılır.

**7. Adam (from keras.optimizers):** Stokastik gradyan iniş optimizasyon algoritmasıdır. Modelin eğitimi sırasında ağırlıkları güncellemek için kullanılır.

**8. EarlyStopping (from keras.callbacks):** Eğitim sırasında modelin performansını izlemek ve belirli bir kriterin karşılandığında eğitimi otomatik olarak durdurmak için kullanılır.

**9. Matplotlib.pyplot (as plt):** Görselleştirmeler oluşturmak için kullanılan bir çizim kütüphanesidir. Grafikler, tablolar ve diğer görsel öğeleri oluşturmak ve görüntülemek amacıyla kullanılır.

**10. NumPy (as np):** Sayısal hesaplamalar için kullanılan bir Python kütüphanesidir. Özellikle çok boyutlu diziler ve matrisler üzerinde etkili operasyonlar sunar.

**11. Scikit-learn.metrics (from sklearn.metrics):** Scikit-learn kütüphanesinde bulunan metrikler arasında yer alan confusion\_matrix ve classification\_report, modelin performansını değerlendirmek için kullanılır.

**12. Seaborn (as sns):** Matplotlib tabanlı bir veri görselleştirme kütüphanesidir. Özellikle istatistiksel grafikler oluşturmak ve veri keşfi için kullanılır.

1.3.2 Konfigürasyonları Belirleme

img\_size = (224, 224)  
train\_path = 'train'  
valid\_path = 'valid'  
test\_path = 'test'

Bu satırlarda modelin eğitimi için train, validation ve test veri setlerinin yolları verildi. Image size 224, 244 ayarlandı çünkü kullanılacak model bu boyutlarda eğitim yaptığından sonuçlar daha yüksek olacaktır. Bir başka boyut olan 299,299 da kullanılabilirdi.

1.3.3 Veri Ön İşleme ve Veri Çoklama

train\_datagen = ImageDataGenerator(preprocessing\_function=preprocess\_input)  
valid\_datagen = ImageDataGenerator(preprocessing\_function=preprocess\_input)  
test\_datagen = ImageDataGenerator(preprocessing\_function=preprocess\_input)

Bu kısım da veri artırma ve ön işleme işlemleri için kullanılacak ImageDataGenerator nesnelerinin oluşturulmasıdır. Bu nesneler, görüntü verilerini çeşitli yöntemlerle değiştirerek eğitim sırasında modelin daha iyi genelleme yapabilmesine ve aşırı uydurmayı önlemeye yardımcı olur. Ayrıca, preprocess\_input fonksiyonu aracılığıyla görüntülerin önceden işlenmesi de bu aşamada gerçekleştirilir.

**ImageDataGenerator**, Keras kütüphanesinde bulunan bir sınıftır ve görüntü verilerini çeşitli şekillerde değiştirerek modelin daha iyi öğrenmesini sağlar.

**preprocessing\_function** parametresi, görüntü verilerini önceden işlemek için kullanılmasını belirtir. Bu, özellikle önceden eğitilmiş bir model kullanılıyorsa ve giriş görüntülerinin belirli bir şekilde ölçeklenmesi veya normalleştirilmesi gerekiyorsa önemlidir.

train\_generator = train\_datagen.flow\_from\_directory(  
 train\_path,  
 target\_size=img\_size,  
 batch\_size=32,  
 class\_mode='categorical')  
  
valid\_generator = valid\_datagen.flow\_from\_directory(  
 valid\_path,  
 target\_size=img\_size,  
 batch\_size=32,  
 class\_mode='categorical')  
  
test\_generator = test\_datagen.flow\_from\_directory(  
 test\_path,  
 target\_size=img\_size,  
 batch\_size=32,  
 class\_mode='categorical')

Bu kısım, ImageDataGenerator nesnelerini kullanarak eğitim, doğrulama ve test veri kümeleri için veri akışlarını oluşturur. Bu veri akışları, belirtilen dizinlerdeki görüntü verilerini yükleyerek, ön işleme ve veri artırma işlemlerini uygulayarak ve belirli bir batch boyutu ile verileri modelin eğitimine sağlamak üzere yapılandırılır.

**flow\_from\_directory** metodu, belirtilen dizindeki görüntü verilerini okuyarak bir veri akışı oluşturur.

**train\_path**, eğitim veri setinin bulunduğu dizini belirtir.

**target\_size**, hedef görüntü boyutunu belirtir.

**batch\_size**, her eğitim iterasyonunda kullanılacak örnek sayısını belirtir.

**class\_mode**='categorical', çoklu sınıflandırma problemi için kategorik etiketleri kullanacağımızı belirtir.

1.3.4 Modelin Tanımlanması ve Derlenmesi (EfficientNetB0):

base\_model = EfficientNetB0(weights='imagenet', include\_top=False, input\_shape=img\_size + (3,))  
  
  
model = Sequential([  
 base\_model,  
 GlobalAveragePooling2D(),  
 Dense(1024, activation='relu'),  
 Dense(10, activation='softmax') # assuming 10 classes for big cats  
])  
  
# Compile the model  
model.compile(optimizer=Adam(learning\_rate=0.0001),  
 loss='categorical\_crossentropy',  
 metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall(), tf.keras.metrics.SensitivityAtSpecificity(0.5)])

Bu kısımda, EfficientNetB0 mimarisini kullanarak bir transfer öğrenme modeli oluşturulmaktadır.

**EfficientNetB0 Modeli:** İlk olarak, önceden eğitilmiş EfficientNetB0 mimarisini kullanarak bir temel model (base\_model) oluşturulur. Bu model, ImageNet veri kümesi üzerinde eğitilmiş ağırlıkları içerir ve giriş görüntülerin boyutunu (input\_shape) belirleyebilirsiniz.

**Sequential Modeli:** Ardından, bir Sequential modeli oluşturulur. Bu model, sıralı olarak katmanları eklemenizi sağlar.

**base\_model:** İlk katman olarak önceden oluşturulan EfficientNetB0 modeli eklenir. Bu, transfer öğrenme prensibini kullanarak önceden eğitilmiş ağırlıkları almanızı sağlar.

**GlobalAveragePooling2D():** Bu katman, önceki katmandan gelen çıktıları alır ve her özellik haritasının ortalamasını alarak düz bir vektöre dönüştürür. Bu, modelin daha önceki özellikleri öğrenmesine yardımcı olabilir ve genel birleşik özellik temsilini sağlar.

**Dense(1024, activation='relu'):** 1024 nörona sahip bir gizli yoğun katman eklenir. Bu katman, önceki katmanlardan gelen özellikleri daha karmaşık bir temsilasyona dönüştürmeye çalışır.

**Dense(10, activation='softmax'):** Çıkış katmanı olarak eklenen bu yoğun katman, 10 sınıflı bir çıkış üretir. Bu örnekte, büyük kediler için 10 sınıflı bir sınıflandırma modeli oluşturulmuş gibi görünüyor. Eğer farklı bir sınıflandırma problemiyle uğraşıyorsanız, çıkış katmanındaki nöron sayısını ve aktivasyon fonksiyonunu ayarlamalısınız.

**model.compile(...):** Modelin derleme aşamasıdır. Bu aşamada, optimize edici (optimizer), kayıp fonksiyonu (loss function) ve metrikler belirlenir. Bu örnekte, Adam optimizer, cross-entropy kayıp fonksiyonu ve çeşitli metrikler (accuracy, precision, recall, sensitivityAtSpecificity) kullanılmıştır.

1.3.5 Eğitim için Geri Çağırımların Tanımlanması

class CustomCallback(tf.keras.callbacks.Callback):  
 def on\_epoch\_end(self, epoch, logs=None):  
 # Check the value of a custom metric (e.g., accuracy)  
 if logs.get('val\_accuracy') > 0.989:  
 print("\nReached 98% validation accuracy. Stopping training.")  
 self.model.stop\_training = True

custom\_callback = CustomCallback()  
early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)

1.3.6 Modelin Eğitilmesi

history = model.fit(train\_generator, validation\_data=valid\_generator, epochs=50, callbacks=[custom\_callback, early\_stopping])

Bu kısımda, model.fit fonksiyonu ile model eğitilir. Epoch değeri 50 olarak verilmektedir. Model checkpoint ve erken durdurma metotlarıda çağırılmaktadır.

1.3.7 Eğitim Geçmişinin Çizdirilmesi

plt.plot(history.history['accuracy'], label='Training Accuracy')  
plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')  
plt.plot(history.history['loss'], label='Training Loss')  
plt.plot(history.history['val\_loss'], label='Validation Loss')  
plt.xlabel('Epoch')  
plt.ylabel('Metric Value')  
plt.legend()  
plt.show()

Bu kısımda, her epoch sırasında elde edilen train\_accuracy ve validation\_accuracy ‘nin görsel olarak bir grafiği oluşturulmaktadır.

1.3.8 Modelin Test Kümesinde Değerlendirilmesi

evaluation = model.evaluate(test\_generator)

Bu kısımda model test verileri üzerinden sınamaya tabi tutulmaktadır ve değerlendirme metrikleri değişken içerisinde saklanmaktadır.

1.3.9 Değerlendirme Metriklerinin Gösterilmesi

print("\nTest Set Metrics:")  
print(f"Loss: {evaluation[0]}")  
print(f"Accuracy: {evaluation[1]}")  
print(f"Precision: {evaluation[2]}")  
print(f"Recall: {evaluation[3]}")  
print(f"Sensitivity at Specificity 0.5: {evaluation[4]}")

model test edildikten sonra test sonuçlarını ekrana yazdırıyoruz.

1.3.9.1 Karışıklık Matrisi ve Sınıflandırma Raporu

y\_pred = model.predict(test\_generator)  
y\_pred\_classes = np.argmax(y\_pred, axis=1)  
y\_true = test\_generator.classes  
  
cm = confusion\_matrix(y\_true, y\_pred\_classes)  
  
plt.figure(figsize=(10, 8))  
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=test\_generator.class\_indices.keys(), yticklabels=test\_generator.class\_indices.keys())  
plt.xlabel('Predicted Label')  
plt.ylabel('True Label')  
plt.title('Confusion Matrix')  
plt.show()  
  
class\_labels = list(test\_generator.class\_indices.keys())  
print("\nClassification Report:")  
print(classification\_report(y\_true, y\_pred\_classes, target\_names=class\_labels))

Karışıklık matrisini ve sınıflandırma raporunu yazdırılmaktadır. Karışıklık matrisi görsel bir grafik olarak ekranda gösterilmektedir

1.3.9.2 Eğitilmiş Modelin ve Ağırlıkların Kaydedilmesi

model.save('big\_cats\_efficientnetB0\_model(224x224).h5')  
model.save\_weights('big\_cats\_efficientnetB0\_model\_weights(224x224).h5')

Modeller en iyi ağırlıklarla kaydedilmektedir

1.4 Xception ve Resnet eğitimleri

Bu eğitimlerde diğerinden farksızdır lakin içeri aktardığımız kütüphanelerden Xception ve Resnet için uygun olanları değiştireceğiz

Xception için,

from keras.applications.xception import Xception, preprocess\_input

Resnet için,

from keras.applications.resnet50 import resnet50, preprocess\_input

1.5 Custom CNN

input\_img = Input(shape=(224, 224, 3))  
x = Conv2D(32, (3, 3), activation='relu', padding='same')(input\_img)  
x = BatchNormalization()(x)  
x = MaxPooling2D((2, 2))(x)  
  
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)  
x = BatchNormalization()(x)  
x = MaxPooling2D((2, 2))(x)  
  
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)  
x = BatchNormalization()(x)  
x = MaxPooling2D((2, 2))(x)  
  
x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)  
x = BatchNormalization()(x)  
x = MaxPooling2D((2, 2))(x)  
x = Dropout(0.25)(x)  
  
x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)  
x = BatchNormalization()(x)  
x = MaxPooling2D((2, 2))(x)  
  
x = Flatten()(x)  
x = Dense(1024, activation='relu')(x)  
x = Dropout(0.5)(x)  
output = Dense(10, activation='softmax')(x) # Assuming 10 classes for big cats  
  
model = Model(inputs=input\_img, outputs=output)  
  
# Compile the Model  
model.compile(optimizer=Adam(), loss='categorical\_crossentropy', metrics=['accuracy'])

**Giriş Katmanı:**

Input fonksiyonu ile bir giriş katmanı oluşturulur. Giriş boyutu 224x224 piksel ve 3 renk kanalından oluşan bir görüntüdür.

**Evrişim Katmanları ve Normalizasyon:**

İlk evrişim katmanı, 32 filtre ve 3x3 kernel boyutu ile tanımlanır. ReLU aktivasyon fonksiyonu ve 'same' dolgusu kullanılır.

Batch Normalization, evrişim katmanının çıkışını normalize eder.

MaxPooling2D katmanı, evrişim katmanının çıkışındaki önemli özellikleri vurgulamak ve boyutu küçültmek için kullanılır.

Bu işlemler, 64, 128, ve 256 filtre sayısına sahip üç ayrı evrişim katmanı için tekrarlanır.

**Dropout Katmanı:**

Bir Dropout katmanı, aşırı uydurmayı önlemek için kullanılır. Bu katman, rastgele olarak belirtilen yüzdeyi evrişim katmanındaki birimlerden atar.

**Tam Bağlantılı (Dense) Katmanlar:**

512 filtre sayısına sahip bir evrişim katmanı eklenir ve yine Batch Normalization ve MaxPooling2D katmanları uygulanır.

Flatten fonksiyonu, evrişim ve havuzlama katmanlarının çıkışını düzleştirir.

Birinci tam bağlantılı katman 1024 nörona sahiptir ve ReLU aktivasyon fonksiyonunu kullanır.

Bir Dropout katmanı, aşırı uydurmayı önlemek için eklenir.

**Çıkış Katmanı:**

Bir ikinci tam bağlantılı katman, 10 nörona sahiptir ve softmax aktivasyon fonksiyonunu kullanarak 10 farklı büyük kedi sınıfının olasılıklarını üretir.

2.0 Uygulama Geliştirme Ortamı

Uygulama geliştirme ortamı olarak PYQT5 kütüphanesinden ve QTdesignerdan yararlanılmaktadır. Eğitim ve Test olarak iki sekmede yapılmaktadır.

2.1 Test

import sys  
import numpy as np  
from keras.preprocessing import image  
from keras.models import load\_model  
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QVBoxLayout, QPushButton, QFileDialog, QComboBox  
from PyQt5.QtGui import QPixmap  
from PyQt5.QtCore import Qt

Bu kısımda PYQT5 ile gerekli kütüphaneleri yüklüyoruz

2.1.1 Arayüz

class ImageSelector(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 self.init\_ui()  
  
 def init\_ui(self):  
 self.setWindowTitle('Resim Seçici')  
  
 self.image\_label = QLabel(self)  
 self.image\_label.setAlignment(Qt.AlignCenter)  
 self.image\_label.setPixmap(QPixmap(''))  
  
 self.select\_button = QPushButton('Resim Seç', self)  
 self.select\_button.clicked.connect(self.open\_image)  
  
 self.model\_combobox = QComboBox(self)  
 self.model\_combobox.addItems(["Xception", "EfficientNetB0","Resnet","CNN"])  
  
 self.analyze\_button = QPushButton('Resmi Analiz Et', self)  
 self.analyze\_button.clicked.connect(self.analyze\_image)  
  
 self.result\_label = QLabel(self)  
 self.result\_label.setAlignment(Qt.AlignCenter)  
  
 layout = QVBoxLayout()  
 layout.addWidget(self.image\_label)  
 layout.addWidget(self.select\_button)  
 layout.addWidget(self.model\_combobox)  
 layout.addWidget(self.analyze\_button)  
 layout.addWidget(self.result\_label)  
  
 self.setLayout(layout)

Bu kısımda, Test işlemi için bilgisayardan resim seçilecek şekilde gerekli butonlar ve modellerin bulunduğu dropdown componenti ekleniyor

2.1.2 Resim seçme metodu

def open\_image(self):  
 file\_dialog = QFileDialog()  
 file\_path, \_ = file\_dialog.getOpenFileName(self, 'Resim Seç', '', 'Image Files (\*.png;\*.jpg;\*.jpeg;\*.gif;\*.bmp)')  
  
 if file\_path:  
 pixmap = QPixmap(file\_path)  
 pixmap = pixmap.scaledToWidth(300)  
 self.image\_label.setPixmap(pixmap)  
 self.image\_path = file\_path

2.1.3 Test İşleminin gerçekleştiği metotlar

def analyze\_image(self):  
 if hasattr(self, 'image\_path'):  
 selected\_model = self.model\_combobox.currentText()  
  
 if selected\_model == "Xception":  
 from keras.applications.xception import preprocess\_input  
 model\_path = "big\_cats\_model(224x244).h5"  
 model = load\_model(model\_path)  
 self.pred\_model(model, preprocess\_input)  
  
 elif selected\_model == "EfficientNetB0":  
 from keras.applications.efficientnet import preprocess\_input  
 model\_path = "big\_cats\_efficientnetB0\_model(224x244).h5"  
 model = load\_model(model\_path)  
 self.pred\_model(model, preprocess\_input)  
  
 elif selected\_model == "Resnet":  
 from keras.preprocessing import image  
 model\_path = 'resnet\_model.h5'  
 model = load\_model(model\_path)  
  
 img = image.load\_img(self.image\_path,  
 target\_size=(224, 224))  
 img\_array = image.img\_to\_array(img)  
 img\_array = np.expand\_dims(img\_array, axis=0)  
 img\_array = img\_array / 255.0  
  
 self.perform\_predictions(img\_array, model)  
  
 elif selected\_model == "CNN":  
 model\_path = "final\_cnn\_model (1).h5"  
 model = load\_model(model\_path)  
 self.pred\_model(model, None)  
  
 else:  
 print("Geçersiz model seçimi.")  
  
def perform\_predictions(self, img\_array, model):  
 predictions = model.predict(img\_array)  
 predicted\_class\_index = np.argmax(predictions)  
 class\_labels = ["African Leopard", "Caracal", "Cheetah", "Clouded Leopard", "Jaguar", "Lions", "Ocelot",  
 "Puma", "Snow Leopard", "Tiger"]  
 predicted\_class\_label = class\_labels[predicted\_class\_index]  
 result\_text = f"Predicted Class Label: {predicted\_class\_label}"  
 self.result\_label.setText(result\_text)  
  
def pred\_model(self, model, preprocess\_func):  
 try:  
 # Seçilen resmi yükleme ve ön işleme  
 img = image.load\_img(self.image\_path, target\_size=(224, 224)) # Adjust target size accordingly  
 img\_array = image.img\_to\_array(img)  
 img\_array = np.expand\_dims(img\_array, axis=0)  
  
 if preprocess\_func:  
 img\_array = preprocess\_func(img\_array)  
  
 # Modeli kullanarak tahmin yapma  
 self.perform\_predictions(img\_array, model)  
  
 except Exception as e:  
 print("An error occurred:", e)

**analyze\_image(self):**

Bu metod, nesnenin 'image\_path' adlı bir özelliğe sahip olup olmadığını kontrol eder.

Bir combo box'tan şu anda seçilen modeli alır.

Seçilen modele göre belirli bir dosyadan önceden eğitilmiş modeli yükler.

Ardından, yüklenen modeli ve uygun ön işleme işlevini kullanarak pred\_model metodunu çağırır.

**pred\_model(self, model, preprocess\_func):**

Bu metod, bir model ve ön işleme işlevini girdi olarak alır.

Seçilen resmi yükler ve belirtilen ön işleme işlemlerini gerçekleştirir.

Ardından, perform\_predictions metodunu çağırarak tahminleri gerçekleştirir.

Herhangi bir hata durumunda bir hata mesajı yazdırır.

**perform\_predictions(self, img\_array, model):**

Bu metod, bir resim dizisi ve önceden eğitilmiş bir modeli girdi olarak alır.

Modeli kullanarak verilen resim için sınıf olasılıklarını tahmin eder.

Tahmin edilen sınıf olasılıklarına dayanarak, tahmin edilen sınıf etiketini belirler.

Sonucu ekrana basar veya başka bir işleme yönlendirir.

2.1.4 Hibrit Test İşlemi

def hybrid\_model\_vote(self):  
 if hasattr(self, 'image\_path'):  
 # Load models with their corresponding preprocessing functions  
 models\_with\_preprocessing = [  
 ("Xception", load\_model("big\_cats\_model(224x244).h5"), keras.applications.xception.preprocess\_input),  
 ("EfficientNetB0", load\_model("big\_cats\_efficientnetB0\_model(224x244).h5"),  
 keras.applications.efficientnet.preprocess\_input),  
 ("ResNet", load\_model("resnet\_model.h5"), lambda x: x / 255.0),  
 # Assuming ResNet uses simple scaling  
 ("CNN", load\_model("final\_cnn\_model (1).h5"), None)  
 # If CNN does not require special preprocessing  
 ]  
  
 # Get predictions from each model  
 all\_predictions = []  
 for model\_name, model, preprocess\_func in models\_with\_preprocessing:  
 predicted\_class\_label = self.get\_model\_specific\_prediction(model, self.image\_path, preprocess\_func)  
 all\_predictions.append(predicted\_class\_label)  
  
 # Perform voting using the aggregate\_predictions method  
 voted\_result = self.aggregate\_predictions(all\_predictions)  
  
 # Display individual predictions and voting result  
 individual\_predictions\_text = "\n".join([f"{model\_name}: {prediction}" for model\_name, prediction in  
 zip([model[0] for model in models\_with\_preprocessing],  
 all\_predictions)])  
 final\_result\_text = f"Voting Result: {voted\_result}"  
 self.result\_label.setText(individual\_predictions\_text + "\n" + final\_result\_text)

her modelin tahmin değerleri alınmaktadır ardından bu tahmin değerlerinin ortalaması alınarak nihai sonuç verilmektedir.

2.1.5 Başlatma şablonu

if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 window = ImageSelector()  
 window.show()  
 sys.exit(app.exec\_())

Arayüzü çalıştırmak için başlatma şablonu.

2.2 Eğitim

Bu kısımda, gerekli arayüz ayarlanmaları yapıldıktan sonra ilgili butonlara ilgili fonksiyonlar bağlanmaktadır.

**Veri çoklama**

def augment\_image(self):  
 if hasattr(self, 'image\_path'):  
 img = image.load\_img(self.image\_path, target\_size=(300, 300))  
 img\_array = image.img\_to\_array(img)  
 img\_array = np.expand\_dims(img\_array, axis=0)  
  
 it = self.datagen.flow(img\_array, batch\_size=1)  
 batch = it.next()  
 augmented\_image = batch[0].astype('uint8')  
  
 # Convert the PIL Image to a format suitable for QImage  
 h, w, ch = augmented\_image.shape  
 bytes\_per\_line = ch \* w  
 convert\_to\_Qt\_format = QImage(augmented\_image.data, w, h, bytes\_per\_line, QImage.Format\_RGB888)  
 p = convert\_to\_Qt\_format.scaled(300, 300, Qt.KeepAspectRatio)  
  
 self.image\_label.setPixmap(QPixmap.fromImage(p))

**Train buttonu**

def on\_train\_click(self):  
 selected\_model = self.model\_selector.currentText()  
 epochs = int(self.epoch\_input.text())  
 batch\_size = int(self.batch\_input.text())  
 patience = int(self.patience\_input.text())  
  
 threading.Thread(target=self.train\_model, args=(selected\_model, epochs, batch\_size, patience),  
 daemon=True).start()

2.3 Eğitim Ve Test Birleştirilmesi

import sys  
import numpy as np  
from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QTabWidget  
from pyqt import ImageSelector # Assuming the test UI is in a file named test\_ui.py  
from train\_metrics import MLTrainerGUI # Assuming the train UI is in a file named train\_ui.py  
from data\_aug import ImageSelector  
class MainApp(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
  
 self.init\_ui()  
  
 def init\_ui(self):  
 self.setWindowTitle('Main Application')  
  
 # Create the tab widget  
 tab\_widget = QTabWidget(self)  
  
 # Create instances of the Test and Train classes  
 test\_ui = ImageSelector()  
 train\_ui = MLTrainerGUI()  
 data\_aug = ImageSelector()  
 # Add the Test and Train widgets to the tabs  
 tab\_widget.addTab(test\_ui, "Test")  
 tab\_widget.addTab(train\_ui, "Train")  
 tab\_widget.addTab(data\_aug, "Data Augmentation")  
  
  
 # Create the main layout  
 layout = QVBoxLayout(self)  
 layout.addWidget(tab\_widget)  
  
 self.setLayout(layout)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 window = MainApp()  
 window.show()  
 sys.exit(app.exec\_())

ilgili kodları referans olarak verildikten sonra gerekli sekmeler ve referanlar verilmektedir. Hem Test hem de Train tek bir arayüz ekranında bulunmaktadır.

2.4 Test ve Eğitim Arayüzü

**metin, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu**

metin, ekran görüntüsü, ekran, görüntüleme, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

3.0 Test Sonuçları

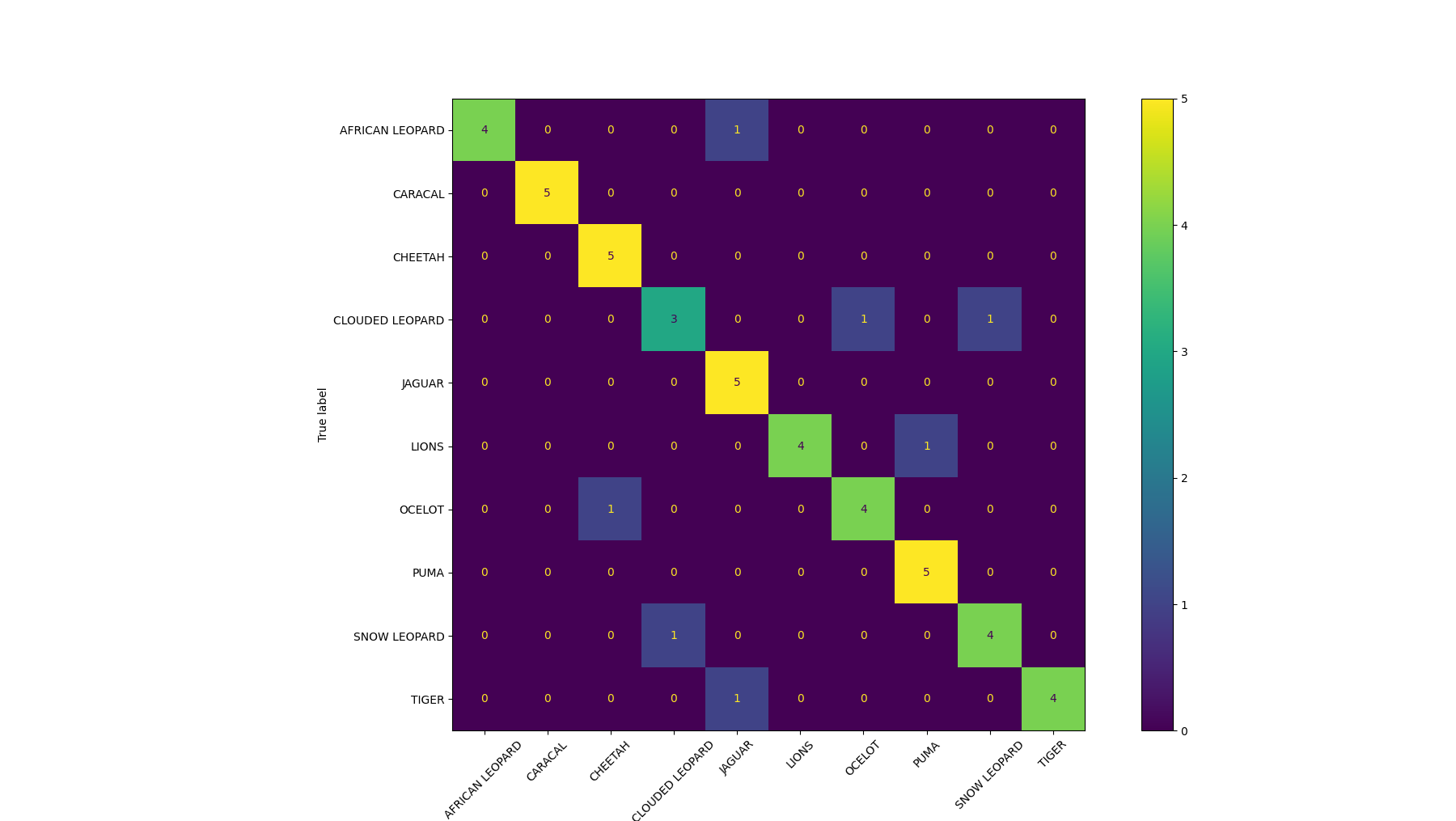
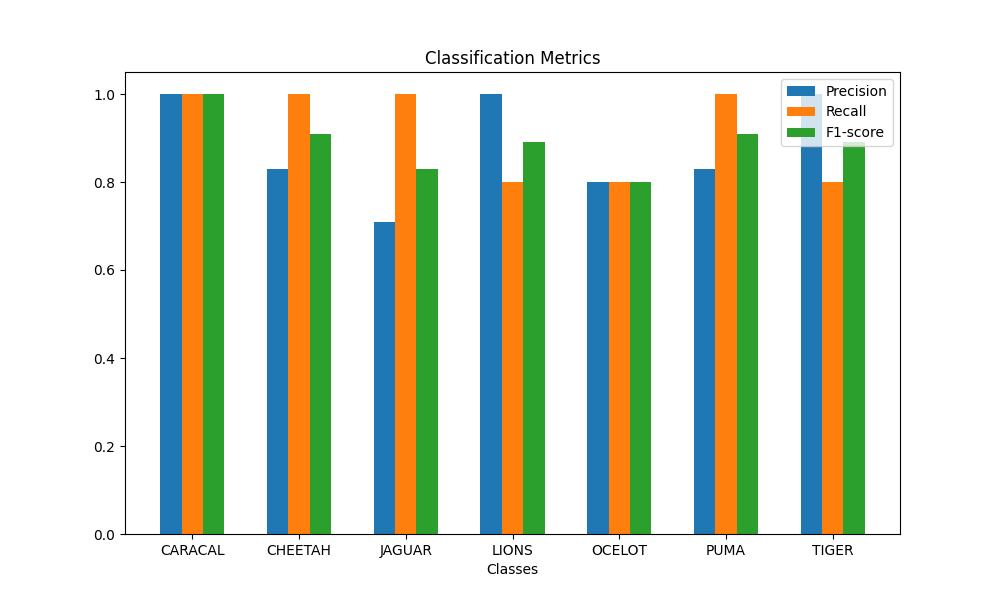
3.1 Xception ve Efficient

ekran görüntüsü, metin, dikdörtgen, çizgi içeren bir resim

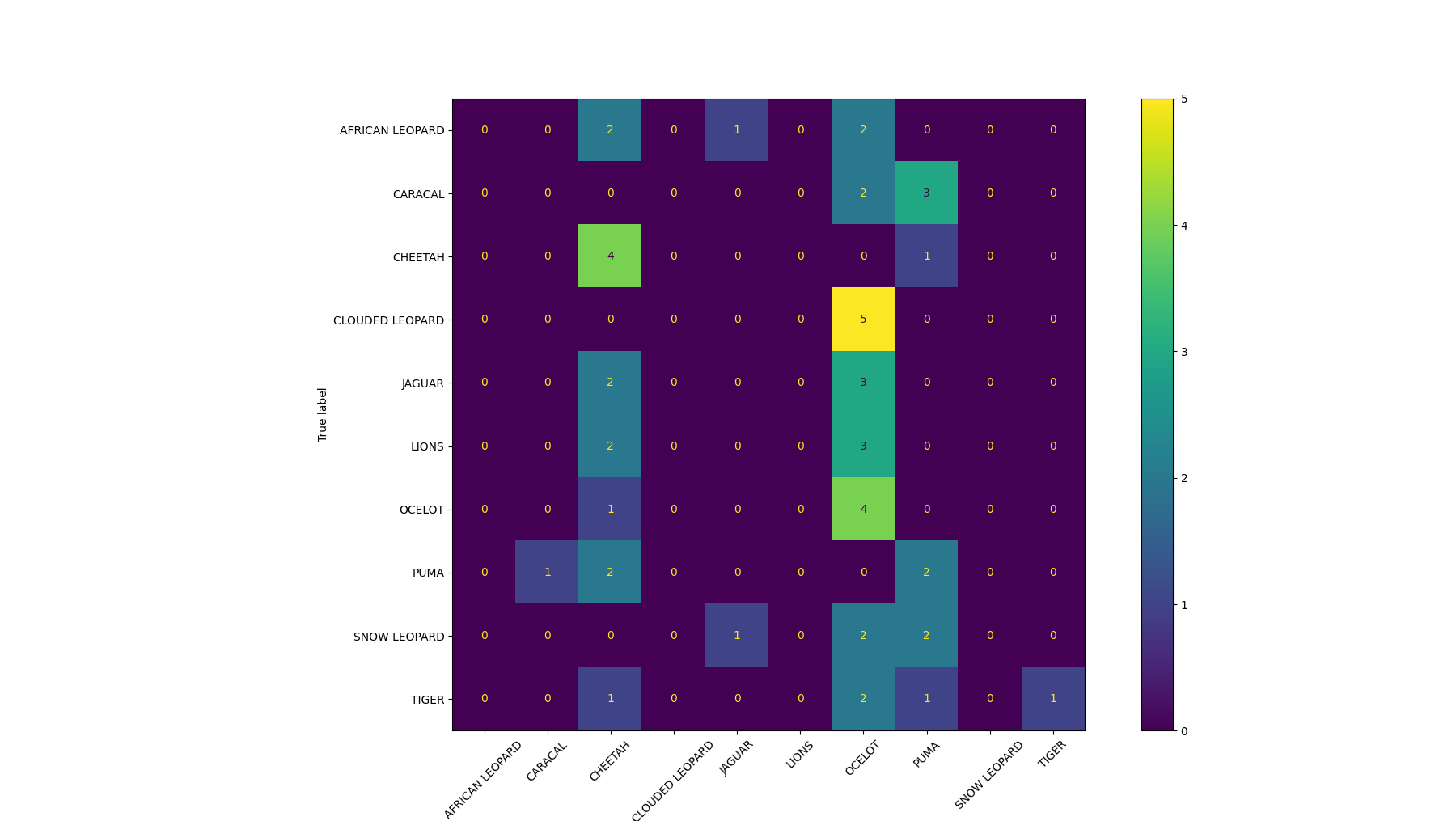
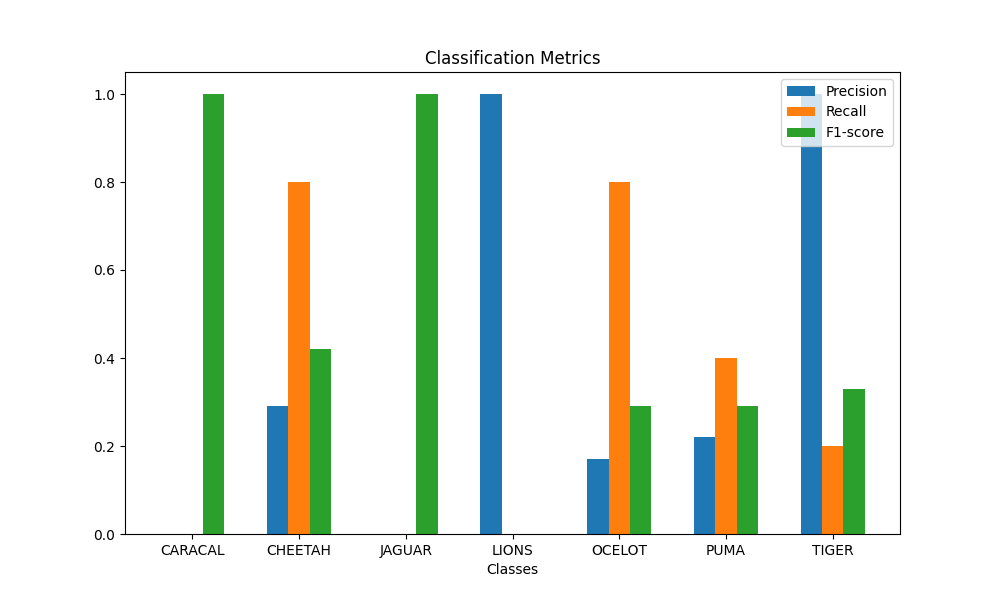
Açıklama otomatik olarak oluşturuldurenklilik, dikdörtgen, diyagram, piksel içeren bir resim

Açıklama otomatik olarak oluşturuldu

3.2 Resnet



3.3 Custom CNN



4.0 Veri seti ve modeller

**https://drive.google.com/file/d/1eD5I24a2UXXsx6VOum-5N9U3ChBdbdx1/view?usp=drive\_link**