







































import os

import cv2

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from tensorflow.keras.models import Sequential, Model, load\_model

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, GlobalAveragePooling2D, BatchNormalization

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.metrics import confusion\_matrix, precision\_score, recall\_score, f1\_score, accuracy\_score

from tensorflow.keras.applications import EfficientNetB0

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

from tensorflow.keras.optimizers import Adam

# Görüntü boyutu artırıldı

IMG\_SIZE = 224 # EfficientNet için optimize edilmiş boyut

NUM\_CLASSES = 10

# Geliştirilmiş veri artırma işlemleri

datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=30, # Artırıldı

width\_shift\_range=0.2,

height\_shift\_range=0.2,

horizontal\_flip=True,

vertical\_flip=True, # Eklendi

zoom\_range=0.2, # Artırıldı

shear\_range=0.15, # Eklendi

fill\_mode='nearest',

brightness\_range=[0.8, 1.2] # Eklendi

)

train\_generator = datagen.flow\_from\_directory(

train\_dir,

target\_size=(IMG\_SIZE, IMG\_SIZE),

batch\_size=16, # Batch size küçültüldü

class\_mode='categorical'

)

test\_datagen = ImageDataGenerator(rescale=1./255)

test\_generator = test\_datagen.flow\_from\_directory(

test\_dir,

target\_size=(IMG\_SIZE, IMG\_SIZE),

batch\_size=16,

class\_mode='categorical',

shuffle=False

)

# EfficientNetB0 kullanımı (ResNet50 yerine)

base\_model = EfficientNetB0(weights='imagenet', include\_top=False, input\_shape=(IMG\_SIZE, IMG\_SIZE, 3))

# İlk 100 katmanı dondur, son katmanları eğitilebilir bırak

for layer in base\_model.layers[:-30]:

layer.trainable = False

model = Sequential([

base\_model,

GlobalAveragePooling2D(),

BatchNormalization(),

Dense(1024, activation='relu'),

BatchNormalization(),

Dropout(0.5),

Dense(512, activation='relu'), # Ek katman

BatchNormalization(),

Dropout(0.3),

Dense(NUM\_CLASSES, activation='softmax')

])

# Learning rate ayarlanabilir optimizer

optimizer = Adam(learning\_rate=0.001)

# Model derleme

model.compile(

optimizer=optimizer,

loss='categorical\_crossentropy',

metrics=['accuracy']

)

# Callbacks eklendi

callbacks = [

EarlyStopping(

monitor='val\_loss',

patience=5,

restore\_best\_weights=True

),

ReduceLROnPlateau(

monitor='val\_loss',

factor=0.2,

patience=3,

min\_lr=1e-6

),

ModelCheckpoint(

'best\_model.h5',

monitor='val\_accuracy',

save\_best\_only=True

)

]

# Model eğitimi

history = model.fit(

train\_generator,

epochs=50, # Epoch sayısı artırıldı

validation\_data=test\_generator,

callbacks=callbacks

)

# Modeli .h5 formatında kaydetme

model.save("model\_fruit\_classifier\_resnet.h5")

# Eğitim ve doğrulama kayıp ve doğruluk grafikleri

plt.plot(history.history['accuracy'], label='Eğitim Doğruluğu')

plt.plot(history.history['val\_accuracy'], label='Doğrulama Doğruluğu')

plt.xlabel('Epoch')

plt.ylabel('Doğruluk')

plt.legend(loc='lower right')

plt.title('Model Doğruluğu')

plt.show()

plt.plot(history.history['loss'], label='Eğitim Kaybı')

plt.plot(history.history['val\_loss'], label='Doğrulama Kaybı')

plt.xlabel('Epoch')

plt.ylabel('Kayıp')

plt.legend(loc='upper right')

plt.title('Model Kaybı')

plt.show()

# Test verileri üzerinden tahminlerin yapılması

y\_pred = model.predict(test\_generator)

y\_pred\_classes = np.argmax(y\_pred, axis=1)

y\_true = test\_generator.classes

# Özellik çıkarımı (Feature Extraction)

feature\_extractor = Model(inputs=model.input, outputs=model.get\_layer("dense").output)

features = feature\_extractor.predict(test\_generator)

# Confusion matrix

conf\_matrix = confusion\_matrix(y\_true, y\_pred\_classes)

plt.figure(figsize=(10, 7))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Tahmin Edilen Etiket')

plt.ylabel('Gerçek Etiket')

plt.show()

# Metriklerin hesaplanması

precision = precision\_score(y\_true, y\_pred\_classes, average='macro')

recall = recall\_score(y\_true, y\_pred\_classes, average='macro')

f1 = f1\_score(y\_true, y\_pred\_classes, average='macro')

accuracy = accuracy\_score(y\_true, y\_pred\_classes)

# Specificity hesaplama

TN = conf\_matrix.sum() - (conf\_matrix.sum(axis=1) + conf\_matrix.sum(axis=0) - np.diag(conf\_matrix))

FP = conf\_matrix.sum(axis=0) - np.diag(conf\_matrix)

specificity = np.mean(TN / (TN + FP))

# Metriklerin yazdırılması

print(f'Accuracy: {accuracy:.4f}')

print(f'Precision: {precision:.4f}')

print(f'Recall: {recall:.4f}')

print(f'F1-Score: {f1:.4f}')

print(f'Specificity: {specificity:.4f}')