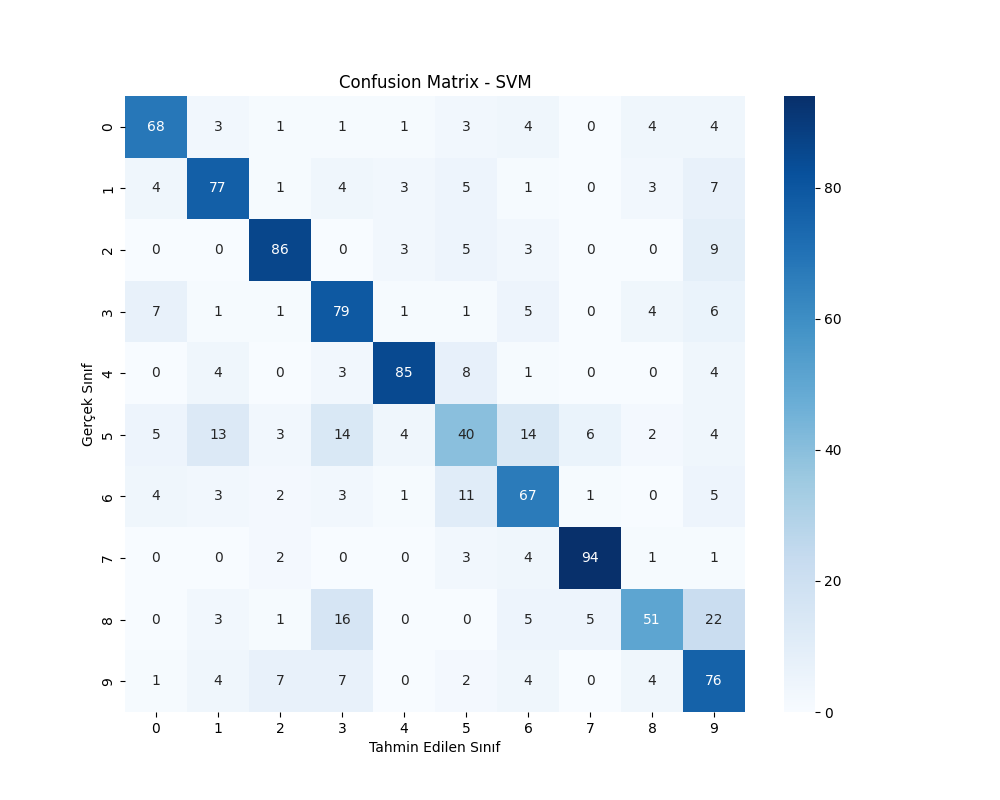
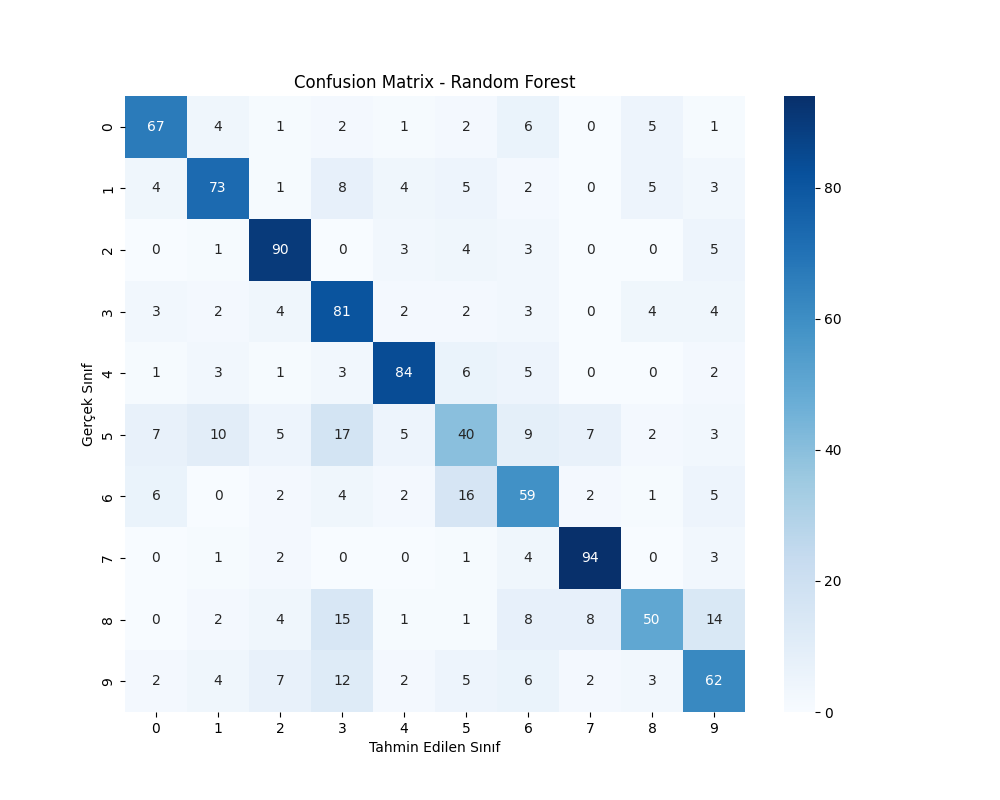
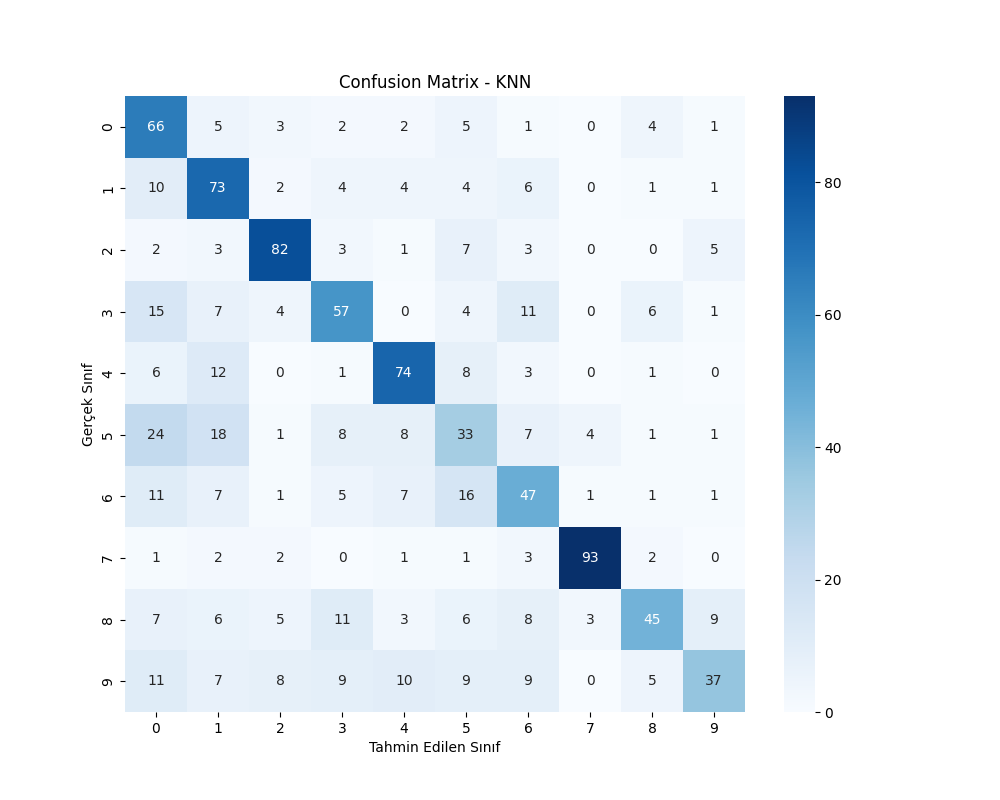
A graph of different colored lines

Description automatically generated with medium confidence

A screen shot of a computer screen

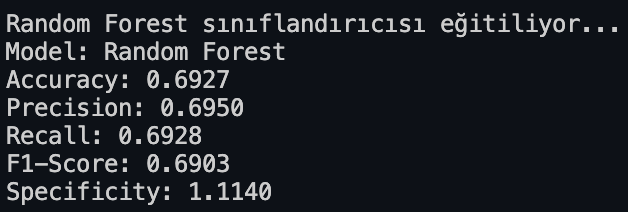
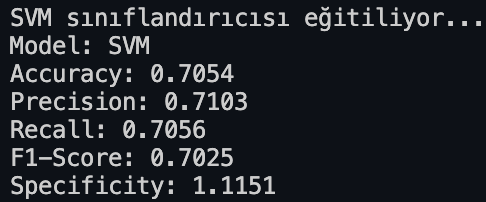
Description automatically generated



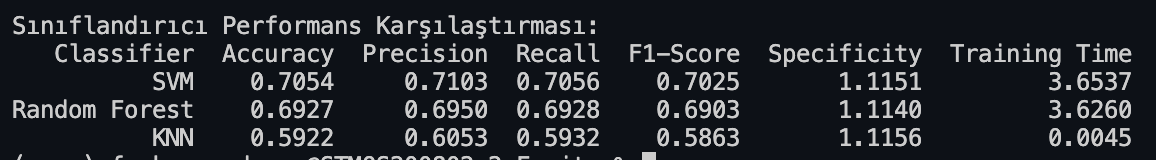


A graph showing different colored bars

Description automatically generated

  
A screenshot of a computer

Description automatically generated



import os

import numpy as np

from tensorflow.keras.models import Model

from tensorflow.keras.applications import VGG16

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix, classification\_report

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

# Eğitim ve test veri yolları

train\_data\_path = "/Users/furkan.erdogan/Projects/deep-learning/Fruits/fruits/train"

test\_data\_path = "/Users/furkan.erdogan/Projects/deep-learning/Fruits/fruits/test"

# Yol kontrolü

if not os.path.exists(train\_data\_path) or not os.path.exists(test\_data\_path):

raise FileNotFoundError("Eğitim veya test veri yolları bulunamadı. Yolları kontrol edin.")

# Önceden eğitilmiş bir model kullanımı (örnek VGG16)

model = VGG16(weights='imagenet', include\_top=True)

# Özellik çıkarıcı model oluşturma

feature\_extractor = Model(inputs=model.input,

outputs=model.get\_layer('fc2').output) # VGG16'nın 'fc2' katmanı

# Veri artırıcıların tanımlanması

datagen = ImageDataGenerator(rescale=1.0 / 255)

# Eğitim ve test veri üreticileri

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

train\_data\_path, # Eğitim verileri klasörü

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical'

)

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

test\_data\_path, # Test verileri klasörü

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical'

)

# Eğitim ve test verilerinden özellik çıkarımı

def extract\_features(generator, feature\_extractor):

features = []

labels = []

batch\_count = 0

total\_batches = len(generator)

for \_ in range(total\_batches):

batch\_images, batch\_labels = next(generator)

batch\_features = feature\_extractor.predict(batch\_images)

features.extend(batch\_features)

labels.extend(np.argmax(batch\_labels, axis=1))

batch\_count += 1

print(f'\rÖzellik çıkarımı: {batch\_count}/{total\_batches} batch işlendi...', end='')

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

# Özellik çıkarımı

print("\nEğitim verilerinden özellik çıkarımı yapılıyor...")

X\_train, y\_train = extract\_features(train\_generator, feature\_extractor)

print("\nTest verilerinden özellik çıkarımı yapılıyor...")

X\_test, y\_test = extract\_features(test\_generator, feature\_extractor)

# Veri standardizasyonu

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Sınıflandırıcılar

classifiers = {

'SVM': SVC(kernel='rbf', random\_state=42),

'Random Forest': RandomForestClassifier(n\_estimators=100, random\_state=42),

'KNN': KNeighborsClassifier(n\_neighbors=5)

}

# Sonuçları saklamak için dictionary

results = {

'Classifier': [],

'Accuracy': [],

'Precision': [],

'Recall': [],

'F1-Score': [],

'Specificity': [],

'Training Time': []

}

# Her sınıflandırıcı için eğitim ve değerlendirme

for name, clf in classifiers.items():

print(f'\n{name} sınıflandırıcısı eğitiliyor...')

# Eğitim süresi ölçümü

import time

start\_time = time.time()

clf.fit(X\_train\_scaled, y\_train)

training\_time = time.time() - start\_time

# Tahminler

y\_pred = clf.predict(X\_test\_scaled)

# Metrikler

report = classification\_report(y\_test, y\_pred, output\_dict=True)

cm = confusion\_matrix(y\_test, y\_pred)

# Specificity hesaplama

tn = np.sum(cm) - (np.sum(cm, axis=0) + np.sum(cm, axis=1) - np.diag(cm)).sum()

fp = np.sum(cm, axis=0) - np.diag(cm)

specificity = np.mean(tn / (tn + fp))

# Sonuçları kaydet

results['Classifier'].append(name)

results['Accuracy'].append(report['accuracy'])

results['Precision'].append(report['macro avg']['precision'])

results['Recall'].append(report['macro avg']['recall'])

results['F1-Score'].append(report['macro avg']['f1-score'])

results['Specificity'].append(specificity)

results['Training Time'].append(training\_time)

# Model sonucu yazdırma

print(f"Model: {name}")

print(f"Accuracy: {report['accuracy']:.4f}")

print(f"Precision: {report['macro avg']['precision']:.4f}")

print(f"Recall: {report['macro avg']['recall']:.4f}")

print(f"F1-Score: {report['macro avg']['f1-score']:.4f}")

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

# Karmaşıklık matrisi görselleştirme

plt.figure(figsize=(10, 8))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title(f'Confusion Matrix - {name}')

plt.xlabel('Tahmin Edilen Sınıf')

plt.ylabel('Gerçek Sınıf')

plt.show()

# Sonuçları tablo halinde göster

results\_df = pd.DataFrame(results)

results\_df = results\_df.round(4)

print("\nSınıflandırıcı Performans Karşılaştırması:")

print(results\_df.to\_string(index=False))

# Performans metriklerini görselleştirme

metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score', 'Specificity']

plt.figure(figsize=(12, 6))

x = np.arange(len(classifiers))

width = 0.15

for i, metric in enumerate(metrics):

plt.bar(x + i\*width, results\_df[metric], width, label=metric)

plt.xlabel('Sınıflandırıcılar')

plt.ylabel('Skor')

plt.title('Sınıflandırıcı Performans Karşılaştırması')

plt.xticks(x + width\*2, results\_df['Classifier'])

plt.legend()

plt.tight\_layout()

plt.show()