**ÜSKÜDAR ÜNİVERSİTESİ**

**FEN BİLİMLERİ ENSTİTÜSÜ**

**YAPAY SİNİR AĞI-FİNAL Ödevi**

1. **Gerekli Kütüphanelerin Yüklenmesi**

* pandas, numpy (veri işleme)
* sklearn, TensorFlow/Keras (model oluşturma, veri ayırma, performans metrikleri)
* matplotlib ve seaborn (görselleştirme)

import os

import random

import shutil

import kagglehub

import numpy as np

import pandas as pd

import seaborn as sns

import tensorflow as tf

from tensorflow import keras

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from tensorflow.keras.models import Model

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.applications import DenseNet121, MobileNet

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.layers import Dense, Dropout, GlobalAveragePooling2D

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

1. **Veri Seti Yükleme**

Face Mask Detection Dataset’i yükleyiniz.

https://www.kaggle.com/datasets/omkargurav/face-mask-dataset

from google.colab import drive

drive.mount('/content/drive')

# Zip dosyasının doğru yolu

zip\_path = "/content/drive/MyDrive/data (1)/archive.zip"  # Google Drive'dan doğru yolu kullan

# Zip dosyasını doğru yere kopyala

!cp "$zip\_path" /content/  # Dosyayı /content/ dizinine kopyala

import zipfile

# Kopyalanan dosyayı çıkart

with zipfile.ZipFile("/content/archive.zip", 'r') as zip\_ref:  # Doğru dosya adı burada

    zip\_ref.extractall("/content/face\_mask\_dataset")  # Çıkartma işlemi

# Veri seti dizinini belirle

dataset\_dir = "/content/face\_mask\_dataset/data"

mask\_dir = os.path.join(dataset\_dir, "with\_mask")

no\_mask\_dir = os.path.join(dataset\_dir, "without\_mask")

# Maskeli ve masksiz görüntü sayısını kontrol et

print(f"Maskeli görüntü sayısı: {len(os.listdir(mask\_dir))}")

print(f"Masksiz görüntü sayısı: {len(os.listdir(no\_mask\_dir))}")

# Örnek bir görüntü gösterimi

import matplotlib.pyplot as plt

sample\_image = plt.imread(os.path.join(mask\_dir, os.listdir(mask\_dir)[0]))

plt.imshow(sample\_image)

plt.title("Maskeli Örnek Görüntü")

plt.axis('off')

plt.show()

metin, kişi, şahıs, insan yüzü, iç mekan içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. **Önceden Eğitilmiş (Pre-trained) Modellerin Hazırlanması**

Sınıflandırıcı olarak DenseNet121 ve MobileNetV1 (pre-trained) modellerinden en az biri kullanılacaktır. Base modeller indirildikten sonra 2 dense katmanı ve bir dropout katmanı ekleyiniz.

Örn: Kütüphane yükleme

from tensorflow.keras.applications import DenseNet121

from tensorflow.keras.applications import MobileNet

Örn: Model oluşturma

        base\_model = DenseNet121(weights='imagenet', include\_top=False, input\_shape=(image\_size[0], image\_size[1], 3))

        x = base\_model.output

        x = GlobalAveragePooling2D()(x)

        x = Dropout(0.5)(x)

        # x = Dense(128, activation='relu')(x)

        x = Dense(256, activation='relu')(x)

        x = Dense(256, activation='relu')(x)

        x = Dense(128, activation='relu')(x)

        x = Dense(64, activation='relu')(x)

        predictions = Dense(2, activation='softmax')(x)

        model = Model(inputs=base\_model.input, outputs=predictions)

def create\_pretrained\_model(base\_model\_name, input\_shape=(224, 224, 3), num\_classes=2, dropout\_rate=0.5):

    """

    Önceden eğitilmiş bir modelin üzerine dense ve dropout katmanları ekleyerek yeni bir model oluşturur.

    Args:

        base\_model\_name (str): Kullanılacak model adı ('DenseNet121' veya 'MobileNet').

        input\_shape (tuple): Giriş görüntü boyutu.

        num\_classes (int): Çıktı sınıf sayısı.

        dropout\_rate (float): Dropout oranı.

    Returns:

        model: Eğitim için hazır Keras model nesnesi.

    """

    if base\_model\_name == "DenseNet121":

        base\_model = DenseNet121(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    elif base\_model\_name == "MobileNet":

        base\_model = MobileNet(weights='imagenet', include\_top=False, input\_shape=input\_shape)

    else:

        raise ValueError("Desteklenmeyen model adı. 'DenseNet121' veya 'MobileNet' seçiniz.")

    # Base modelin katmanlarını dondurma

    base\_model.trainable = False

    # Yeni katmanlar ekleme

    x = base\_model.output

    x = GlobalAveragePooling2D()(x)

    x = Dropout(dropout\_rate)(x)

    x = Dense(256, activation='relu')(x)

    x = Dense(256, activation='relu')(x)

    x = Dropout(dropout\_rate)(x)

    x = Dense(128, activation='relu')(x)

    x = Dense(128, activation='relu')(x)

    x = Dropout(dropout\_rate)(x)

    x = Dense(64, activation='relu')(x)

    x = Dense(64, activation='relu')(x)

    predictions = Dense(num\_classes, activation='softmax')(x)

    # Yeni modeli oluşturma

    model = Model(inputs=base\_model.input, outputs=predictions)

    return model

# DenseNet121 Model

densenet\_model = create\_pretrained\_model(base\_model\_name="DenseNet121")

# MobileNet Model

mobilenet\_model = create\_pretrained\_model(base\_model\_name="MobileNet")

# Model özetlerini yazdırma

print("DenseNet121 Model Özeti:")

densenet\_model.summary()

print("\nMobileNet Model Özeti:")

mobilenet\_model.summary()

1. **Veri Setinin Eğitim/Validasyon ve Test Olarak Ayrılması:**

Toplam verinin **%20’si** **test set** olarak ayrılmalı, **%80’i** eğitim/validasyon için bırakılmalıdır.

import os

import shutil

from sklearn.model\_selection import train\_test\_split

# Dizinleri oluştur

os.makedirs(train\_mask\_dir, exist\_ok=True)

os.makedirs(train\_no\_mask\_dir, exist\_ok=True)

os.makedirs(val\_mask\_dir, exist\_ok=True)

os.makedirs(val\_no\_mask\_dir, exist\_ok=True)

os.makedirs(test\_mask\_dir, exist\_ok=True)

os.makedirs(test\_no\_mask\_dir, exist\_ok=True)

# Maskeli ve masksiz görselleri al

mask\_images = os.listdir(mask\_dir)

no\_mask\_images = os.listdir(no\_mask\_dir)

# Eğitim, validasyon ve test setlerini ayırma

train\_mask, temp\_mask = train\_test\_split(mask\_images, test\_size=0.4, random\_state=42)

train\_no\_mask, temp\_no\_mask = train\_test\_split(no\_mask\_images, test\_size=0.4, random\_state=42)

val\_mask, test\_mask = train\_test\_split(temp\_mask, test\_size=0.5, random\_state=42)

val\_no\_mask, test\_no\_mask = train\_test\_split(temp\_no\_mask, test\_size=0.5, random\_state=42)

# Eğitim ve validasyon setlerine kopyala

for image in train\_mask:

    shutil.copy(os.path.join(mask\_dir, image), os.path.join(train\_mask\_dir, image))

for image in val\_mask:

    shutil.copy(os.path.join(mask\_dir, image), os.path.join(val\_mask\_dir, image))

for image in test\_mask:

    shutil.copy(os.path.join(mask\_dir, image), os.path.join(test\_mask\_dir, image))

for image in train\_no\_mask:

    shutil.copy(os.path.join(no\_mask\_dir, image), os.path.join(train\_no\_mask\_dir, image))

for image in val\_no\_mask:

    shutil.copy(os.path.join(no\_mask\_dir, image), os.path.join(val\_no\_mask\_dir, image))

for image in test\_no\_mask:

    shutil.copy(os.path.join(no\_mask\_dir, image), os.path.join(test\_no\_mask\_dir, image))

# Dosya sayısını yazdırma

print(f"Maskeli görüntü sayısı (train): {len(os.listdir(train\_mask\_dir))}")

print(f"Masksiz görüntü sayısı (train): {len(os.listdir(train\_no\_mask\_dir))}")

print(f"Maskeli görüntü sayısı (validation): {len(os.listdir(val\_mask\_dir))}")

print(f"Masksiz görüntü sayısı (validation): {len(os.listdir(val\_no\_mask\_dir))}")

print(f"Maskeli görüntü sayısı (test): {len(os.listdir(test\_mask\_dir))}")

print(f"Masksiz görüntü sayısı (test): {len(os.listdir(test\_no\_mask\_dir))}")

1. **Eğitim/Validasyon Setinin Bölünmesi ve Modelin Eğitilmesi**

Daha önce ayırdığınız **eğitim+validasyon** verisini **%80’i** eğitim **%20’si** validasyon olarak rastgele bölün. Eğitim verisine veri arttırımı uygulayın

1. datagen = ImageDataGenerator(
2. rescale=1.0/255,
3. rotation\_range=10,
4. width\_shift\_range=0.1,
5. height\_shift\_range=0.1,
6. shear\_range=0.1,
7. zoom\_range=0.1,
8. horizontal\_flip=True,
9. fill\_mode='nearest'
10. )

* Modellerinizi **100 epok** boyunca eğitin.
* **Early Stopping** kullanın: patience=25 olarak ayarlayın. 25 epok boyunca validasyon kaybında (validation loss) iyileşme olmazsa eğitim duracaktır.
* **Optimizer** olarak **ADAM** (Adaptive Moment Estimation) veya **SGD** (Stochastic Gradient Descent) kullanın.
* Öğrenme hızı lr başlangıç değerini 0.001 olarak başlatın.
* Momentum değeri isteğe bağlı ayarlanabilir (örn. 0.9).
* **ReduceLROnPlateau** mekanizması ile validasyon kaybı iyileşme göstermediğinde öğrenme hızını (lr) otomatik olarak azaltın. Örneğin: factor=0.1, patience=5 vs.
* **Eğitim/Validasyon Accuracy ve Loss Grafiklerini** rapora ekleyin.

Her epok sonunda eğitim kaybı (loss) ve doğruluğu (accuracy) ile validasyon kaybı ve doğruluğunu kaydedin ve eğriyi çizdirin.

# Veri arttırımı için ImageDataGenerator ayarları

train\_datagen = ImageDataGenerator(

    rescale=1.0/255,

    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'

)

# Validasyon ve test için sadece rescale işlemi

val\_test\_datagen = ImageDataGenerator(rescale=1.0/255)

# Eğitim verilerini al

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

    "/content/face\_mask\_dataset/train",

    target\_size=(224, 224),

    batch\_size=32,

    class\_mode='categorical'  # Çünkü iki sınıfımız var: Maskeli ve masksiz

)

# Validasyon verilerini al

val\_generator = val\_test\_datagen.flow\_from\_directory(

    "/content/face\_mask\_dataset/validation",

    target\_size=(224, 224),

    batch\_size=32,

    class\_mode='categorical'

)

# Test verilerini al

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

    "/content/face\_mask\_dataset/test",

    target\_size=(224, 224),

    batch\_size=32,

    class\_mode='categorical'

)

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.applications import DenseNet121, MobileNet

from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout, Dense

from tensorflow.keras.models import Model

import matplotlib.pyplot as plt

# Modeli derlemek

def compile\_and\_train(model, train\_generator, val\_generator, epochs=100, initial\_lr=0.001):

    # EarlyStopping callback

    early\_stopping = EarlyStopping(monitor='val\_loss', patience=25, restore\_best\_weights=True)

    # ReduceLROnPlateau callback

    reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.1, patience=5, min\_lr=1e-6)

    # Adam optimizer

    optimizer = Adam(learning\_rate=initial\_lr)

    # Modeli derle

    model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])

    # Eğitim sürecini başlat

    history = model.fit(

        train\_generator,

        epochs=epochs,

        validation\_data=val\_generator,

        callbacks=[early\_stopping, reduce\_lr],

        verbose=1

    )

    return history

# Model özetlerini yazdırma

densenet\_model = create\_pretrained\_model(base\_model\_name="DenseNet121")

mobilenet\_model = create\_pretrained\_model(base\_model\_name="MobileNet")

# DenseNet121 Modeli Eğit

print("DenseNet121 Model Eğitiliyor...")

densenet\_history = compile\_and\_train(densenet\_model, train\_generator, val\_generator)

# MobileNet Modeli Eğit

print("\nMobileNet Model Eğitiliyor...")

mobilenet\_history = compile\_and\_train(mobilenet\_model, train\_generator, val\_generator)

# Eğitim ve Validasyon Kaybı ve Doğruluğunu Çizdirmek

def plot\_history(history, model\_name):

    # Eğitim ve Validasyon Kayıplarını çiz

    plt.figure(figsize=(12, 6))

    # Kaybı çiz

    plt.subplot(1, 2, 1)

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

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

    plt.title(f'{model\_name} Kaybı')

    plt.xlabel('Epochs')

    plt.ylabel('Loss')

    plt.legend()

    # Doğruluğu çiz

    plt.subplot(1, 2, 2)

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

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

    plt.title(f'{model\_name} Doğruluğu')

    plt.xlabel('Epochs')

    plt.ylabel('Accuracy')

    plt.legend()

    plt.tight\_layout()

    plt.show()

# Her modelin eğitim ve doğruluk grafiğini çiz

plot\_history(densenet\_history, "DenseNet121")

plot\_history(mobilenet\_history, "MobileNet")

metin, diyagram, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, diyagram, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Yorum**

**DenseNet**

Eğitim ilk epoch'larda hızlı şekilde doğruluk oranı olarak yüzde 97'ye kadar yükseliyor. Epoch sayısı arttıkça loss grafiğinde de düşüş gözlemleniyor.

ReduceLROnPlateau callback fonksiyonu ise accuracy'nin fazla olduğu yerde öğrenme oranını düşürüyor ve optimal noktayı bulmaya çalışıyor. Model en son epochlarda öğrenme oranını çok düşük seviyey indirdi.

Modelin doğruluk seviyesinde önemli artış yaşanmadığı için early stopping fonksiyonu ile daha fazla eğitilmedi. Gereksiz yere eğitmemiş olduk. 30. epoch'tan sonra model eğitimi bıraktı.

Accuracy: 98.31

Validation Accuracy: 99.61

Accuracy değerlerine bakarsak modelimiz iyi sınıflandırıyor. Overfitting gözükmüyor, validasyon loss değeri düşük. Validasyon performansı eğitim performansından iyi olduğu için genelleme yeteneği iyi.

**MobileNet**

Model ilk epoch'ta %81 accuracy ile başlayıp ikinci epoch'ta %97 accuracy'e sıçramış ve hızla öğrenmiştir.

Loss değeri 0.02 seviyelerine kadar düşmüştür.

Başlangıçta learning rate 0.001 iken 20. epoch'ta 0.0001, 25. epochta 0.00001, 30. epoch'ta 0.000001'e kadar düştü. Yani adoğruluk arttıkça öğrenme katsayımızı düşürdük ReduceLRonPlateu ile.

Validasyon ve eğitim eğrileri birbirini yakınlar overfitting olmadığını görebiliyoruz. Genelleme performansımız çok iyi.

EarlyStopping parametresi 39. epoch'ta durdu daha fazla eğitime gerek yok, modelimiz optimum noktaya ulaştı.

**4. Test Verisi İle Tahmin (Prediction) ve Sonuç Analizi**

* Eğitilen model(ler)in en iyi kayıtlı ağırlıklarını (weights) yükleyip test set üzerinde tahmin (inference) yapın.

Çıktı olarak aşağıdaki metrikleri rapora ekleyin:

* ROC Eğrisi (Receiver Operating Characteristic Curve) **ve AUC (Area Under Curve)**
* Confusion Matrix **(Karışıklık Matrisi)**
* Accuracy **(Doğruluk)**
* Precision **(Duyarlılık)**
* Recall **(Hassasiyet / Duyarlılık)**
* Specificity **(Özgüllük)**
* F1-score

# Modeli kaydetmek için

densenet\_model.save('densenet\_model.h5')  # DenseNet121 modelini kaydediyoruz

mobilenet\_model.save('mobilenet\_model.h5')  # MobileNet modelini kaydediyoruz

import tensorflow as tf

from tensorflow.keras.models import load\_model

from sklearn.metrics import confusion\_matrix, classification\_report, roc\_curve, auc

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# Kaydedilen modelin yüklenmesi

densenet\_model = load\_model('densenet\_model.h5')

mobilenet\_model = load\_model('mobilenet\_model.h5')

# Test verisini hazırlama (Örnek olarak test\_generator kullanılıyor)

# test\_generator = ...

# Test seti üzerinde tahmin yapma

y\_pred\_densenet = densenet\_model.predict(test\_generator, verbose=1)

y\_pred\_mobilenet = mobilenet\_model.predict(test\_generator, verbose=1)

# Gerçek etiketler

y\_true = test\_generator.classes

# En yüksek olasılık değerine göre tahmin yapma (y\_pred'in argmax'ını alacağız)

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

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

# ROC Eğrisi ve AUC (DenseNet ve MobileNet için ayrı ayrı)

def plot\_roc\_curve(y\_true, y\_pred, model\_name):

    fpr, tpr, \_ = roc\_curve(y\_true, y\_pred)

    roc\_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 8))

    plt.plot(fpr, tpr, color='blue', lw=2, label=f'{model\_name} (AUC = {roc\_auc:.2f})')

    plt.plot([0, 1], [0, 1], color='gray', linestyle='--')

    plt.xlim([0.0, 1.0])

    plt.ylim([0.0, 1.05])

    plt.xlabel('False Positive Rate')

    plt.ylabel('True Positive Rate')

    plt.title(f'ROC Curve - {model\_name}')

    plt.legend(loc="lower right")

    plt.show()

plot\_roc\_curve(y\_true, y\_pred\_densenet, 'DenseNet121')

plot\_roc\_curve(y\_true, y\_pred\_mobilenet, 'MobileNet')

# Confusion Matrix (Karışıklık Matrisi)

def plot\_confusion\_matrix(y\_true, y\_pred, model\_name):

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

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Mask', 'With Mask'], yticklabels=['No Mask', 'With Mask'])

    plt.title(f'Confusion Matrix - {model\_name}')

    plt.xlabel('Predicted')

    plt.ylabel('True')

    plt.show()

plot\_confusion\_matrix(y\_true, y\_pred\_densenet, 'DenseNet121')

plot\_confusion\_matrix(y\_true, y\_pred\_mobilenet, 'MobileNet')

# Classification Report (Precision, Recall, F1-score)

report\_densenet = classification\_report(y\_true, y\_pred\_densenet, target\_names=['No Mask', 'With Mask'])

report\_mobilenet = classification\_report(y\_true, y\_pred\_mobilenet, target\_names=['No Mask', 'With Mask'])

print("DenseNet121 Classification Report:")

print(report\_densenet)

print("\nMobileNet Classification Report:")

print(report\_mobilenet)

# Accuracy, Precision, Recall, Specificity, F1-Score (Kendi hesaplamalarımız)

def calculate\_metrics(cm):

    accuracy = np.trace(cm) / float(np.sum(cm))

    precision = cm[1, 1] / float(cm[0, 1] + cm[1, 1])

    recall = cm[1, 1] / float(cm[1, 0] + cm[1, 1])

    specificity = cm[0, 0] / float(cm[0, 0] + cm[0, 1])

    f1\_score = 2 \* (precision \* recall) / (precision + recall)

    return accuracy, precision, recall, specificity, f1\_score

cm\_densenet = confusion\_matrix(y\_true, y\_pred\_densenet)

cm\_mobilenet = confusion\_matrix(y\_true, y\_pred\_mobilenet)

metrics\_densenet = calculate\_metrics(cm\_densenet)

metrics\_mobilenet = calculate\_metrics(cm\_mobilenet)

print("\nDenseNet121 Metrics:")

print(f"Accuracy: {metrics\_densenet[0]:.2f}")

print(f"Precision: {metrics\_densenet[1]:.2f}")

print(f"Recall: {metrics\_densenet[2]:.2f}")

print(f"Specificity: {metrics\_densenet[3]:.2f}")

print(f"F1-Score: {metrics\_densenet[4]:.2f}")

print("\nMobileNet Metrics:")

print(f"Accuracy: {metrics\_mobilenet[0]:.2f}")

print(f"Precision: {metrics\_mobilenet[1]:.2f}")

print(f"Recall: {metrics\_mobilenet[2]:.2f}")

print(f"Specificity: {metrics\_mobilenet[3]:.2f}")

metin, çizgi, öykü gelişim çizgisi; kumpas; grafiğini çıkarma, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, çizgi, ekran görüntüsü, öykü gelişim çizgisi; kumpas; grafiğini çıkarma içeren bir resim

Açıklama otomatik olarak oluşturulduprint(f"F1-Score: {metrics\_mobilenet[4]:.2f}")

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

Açıklama otomatik olarak oluşturuldu**metin, ekran görüntüsü, diyagram, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, diyagram, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**DenseNet121 Metrics:**

Accuracy: 0.50

Precision: 0.50

Recall: 0.50

Specificity: 0.49

F1-Score: 0.50

**MobileNet Metrics:**

Accuracy: 0.51

Precision: 0.52

Recall: 0.51

Specificity: 0.51

F1-Score: 0.51

**Yorum**

**Recall**: İki modeli kıyasladığımızda pozitifleri daha iyi yakalama konusunda MobileNet önde.

**F1-Score:** Dengeli performans metriğine bakarsak 0.01 fark ile MobileNet önde.

MobileNet modeli DenseNet modelini burun farkıyla geçiyor. Test veri setinde modellerimiz iyi başarı göstermiyor. Veri seti arttırılabilir, data augmentation teknikleri uygulanabilir.

Modelin hiperparametreleri için optimizasyon yapılabilir.

Eğitim sırasında %99 doğruluk elde edilmesine rağmen test setinde %50-51 performans, aşırı öğrenme (overfitting) olduğunu gösterir bize.

Eğitim ve test veri seti seti arasında dağılım farkı olabilir. Veri seti dengesiz dağılmış olabilir.

Model iyileştirmesi için dropout ekleyebiliriz.

**Teslim Edilecekler**

1. **Kod**: Tüm adımları içeren ve çalıştırılabilir bir Jupyter Notebook dosyası.
2. **Sonuçlar ve Yorumlar**: Confusion matrix, performans metrikleri ve hiperparametre denemelerine göre elde edilen sonuçlarla birlikte MLP ve RBF modellerinin karşılaştırılması.
3. **Yorumlar**: Model performansını etkileyen hiperparametreler, model seçimleri ve sonuçlarla ilgili kısa açıklamalar.
4. Github linki
5. Word dosyası