İçindekiler

[Yapay Sinir Ağları Ara Sınav Ödev Raporu 1](#_Toc183632245)

[Veri Setinin İndirilmeye Hazır Hale Getirilmesi Aşamaları 1](#_Toc183632246)

[Veri Setinin İndirilmesi ve Resimlerin İşlenmesi 2](#_Toc183632247)

[Veri Setinin İncelenmesi ve Veri Ön İşleme 4](#_Toc183632248)

[MLP Modelinin Oluşturulması – Temel Versiyon 8](#_Toc183632249)

[RBF Modelinin Oluşturulması – Temel Versiyon 14](#_Toc183632250)

[MLP İçin Hiperparametre Optimizasyonu 19](#_Toc183632251)

[RBF İçin Hiperparametre Optimizasyonu 24](#_Toc183632252)

# Yapay Sinir Ağları Ara Sınav Ödev Raporu

Yapay sinir ağları ödevinde, yapay sinir ağların temellerini öğrenmek amacıyla MLP (Multiple Layer Perceptron) ve RBF (Radial Basis Function) ağları ile “TSRB-German Traffic Sign Recognition Benchmark” veri seti kullanılarak bir sınıflandırma çalışması yapılmıştır. Bu çalışmanın ana başlıkları aşağıda belirtilmiştir;

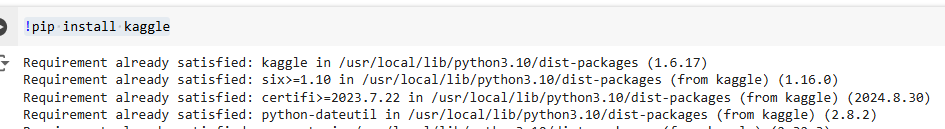
## Veri Setinin İndirilmeye Hazır Hale Getirilmesi Aşamaları

Yapay sinir ağları ile ilgili çalışma için Google Colab üzerinde mount etme işlemi yapılmıştır.

from google.colab import drive

drive.mount('/content/drive')

İlgili çalışma **“TSRB - German Traffic Sign Recognition Benchmark.”** veri seti üzerinden yapılmaktadır. İlgili veri setine erişim için, Google Colab üzerinden kaggle paketi yüklenerek ve ilgili API keyi ile bağlanılarak sağlanmıştır.



Daha sonrasında;

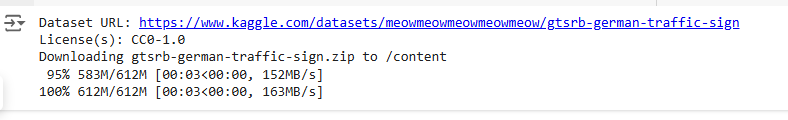
!mkdir -p ~/.kaggle

!echo '{"username":"username","key":"xfd15bda81dbea636d3690a3250d1679"}' > ~/.kaggle/kaggle.json

!chmod 600 ~/.kaggle/kaggle.json

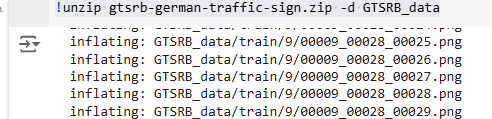
Şeklinde doğrulama dosyası oluşturulmuş ve veri seti indirilmek üzere hazır hale getirilmiştir. Bunun içinde aşağıdaki kod kullanılmıştır.

!kaggle datasets download -d meowmeowmeowmeowmeow/gtsrb-german-traffic-sign



İndirilen zip dosyasının açılması aşaması;

!unzip gtsrb-german-traffic-sign.zip -d GTSRB\_data



Dizinin içeriğini listelemek için aşağıdaki komutu kullanabiliriz;

!ls GTSRB\_data



* meta, Meta.csv: Veri setine dair ek bilgiler içeren meta veriler.
* test, Test, Test.csv: Test verileri veya ilgili dosyalar.
* train, Train, Train.csv: Eğitim verileri veya bu verilere ait dosyalar.

## Veri Setinin İndirilmesi ve Resimlerin İşlenmesi

GTSRB trafik işaretleri veri setinden resimleri indirip, her resmi 32x32 boyutlarına yeniden boyutlandırmak ve sınıf etiketleri birlikte numpy dizilerine dönüştürebiliriz.

Her sınıf için ayrı klasörlerdeki .ppm formatındaki resimleri okuyup uygun bir veri yapısına aktarmamız gerekiyor. Sonuçta, model eğitimi için kullanılacak resim verileri (x) ve bunların etiketleri (y) döndürülür.

Verileri numpy dizilerinde saklayacağız. Os kütüphanesi dosya ve dizin işlemleri için, Python Imaging Library kütüphanesi ile ise resimleri açma ve yeniden boyutlandırma gibi işlemleri yaparız.

Peki buna neden ihtiyacımız olur. Çünkü farklı büyüklükteki resimler, model girişine uygun bir formatta kullanılmaz ise hata oluşur. Bu nedenle tüm resimlerin boyutlarının aynı olması önemlidir. 32x32 veya 64x64 gibi boyutlar tercih edilebilir. Biz 32x32 yi tercih ediyoruz. Çok büyük boyutlu resimler çünkü daha fazla hesaplama gücü ve bellek ihtiyacı oluşturur. O nedenle bu boyut daha uygun görünmektedir. Aynı zamanda daha küçük boyutlara indirilen resimler temel özelliklerini koruyarak (örneğin, şekil, renk ve desenler) karmaşıklığı da azaltır.

import numpy as np

import os

from PIL import Image

def load\_data(data\_dir):

    images = []

    labels = []

    # Her bir sınıf için klasörleri dolaş

    for class\_id in range(43):

        class\_dir = os.path.join(data\_dir, str(class\_id))

        if not os.path.exists(class\_dir):

            print(f"Klasör bulunamadı: {class\_dir}")

            continue

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

        # Her bir resmi yükle

        for img\_name in class\_images:

            if img\_name.endswith('.ppm'):

                img\_path = os.path.join(class\_dir, img\_name)

                img = Image.open(img\_path)

                img = img.resize((32, 32))  # Resimleri yeniden boyutlandır

                images.append(np.array(img))

                labels.append(class\_id)

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

# Veriyi yükle

X, y = load\_data('GTSRB\_data/Train')

Kodda images=[] ve labels=[] isminde resimleri ve sınıf etiketlerini saklamak için boş liste değişkenleri oluşturduk. Aşağıdaki işlemler genel olarak kod içinde gerçekleştirilir;

* ***os.listdir(class\_dir):*** Klasördeki tüm dosyaları liste olarak alır.
* ***img\_name.endswith('.ppm'):***Sadece .ppm formatındaki dosyalar seçilir (PPM, Portable Pixmap resim formatıdır).
* ***Image.open(img\_path):*** Resim açılır.
* ***img.resize((32, 32)):*** Resim 32x32 piksel boyutlarına yeniden boyutlandırılır. Bu, model eğitimi için gerekli standart bir boyuttur.
* ***np.array(img):*** Resim numpy dizisine dönüştürülür ve listeye eklenir.
* ***labels.append(class\_id):*** Her resim için sınıf etiketi listeye eklenir.

En son fonksiyon aşağıdaki şekilde geri parametre döndürür;

 ***images:*** Tüm resimlerin numpy dizisi olarak döndürülmesi.

 ***labels:*** Her resmin sınıf etiketi (0-42 arasında) numpy dizisi olarak döndürülmesi.

Fonksiyon aşağıdaki şekilde çağrılarak; x:Resimler numpy dizisi(özellikler) ve y:Resimlerin sınıf etiketleri gönderilmektedir;

X, y = load\_data('GTSRB\_data/Train')

## Veri Setinin İncelenmesi ve Veri Ön İşleme

Verinin ilk bölümlerinin incelenerek bir fikir edinmek iyi olacaktır;

# CSV içeriğini kontrol et

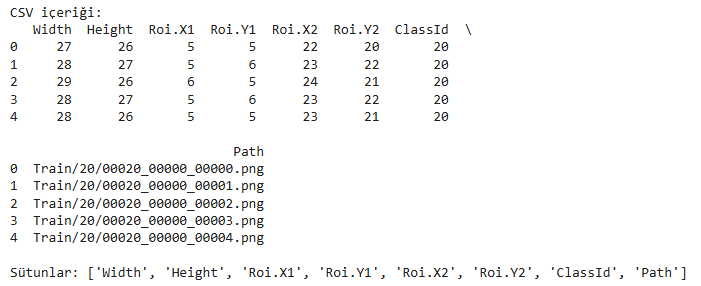
import pandas as pd

train\_df = pd.read\_csv('GTSRB\_data/Train.csv')

print("CSV içeriği:")

print(train\_df.head())

print("\nSütunlar:", train\_df.columns.tolist())



Veri seti üzerinde biraz açıklama yapmak gerekirse;

Öncelikle sütunları ifade edelim;

* Width ve Height:

Resmin genişlik ve yüksekliğini ifade eder.

Örneğin: İlk satırda resmin genişliği 27, yüksekliği 26 olarak belirtilmiş.

* Roi.X1, Roi.Y1, Roi.X2, Roi.Y2:

Resim üzerindeki ROI (Region of Interest) yani "ilgi alanı"nın koordinatlarını temsil eder.

Bu alan, trafik işaretinin bulunduğu dikdörtgen bölgeyi tanımlar.

Örneğin:

İlk satırda ROI sol üst köşesi (5, 5) koordinatında başlıyor.

Sağ alt köşesi (22, 20) koordinatında bitiyor.

* ClassId:

Resmin sınıf etiketini belirtir. Bu, resimdeki trafik işaretinin türünü ifade eden bir numaradır (örneğin, hız sınırı, dur işareti vb.).

Örneğin: İlk satırda ClassId değeri 20 olarak verilmiş.

* Path:

Resmin dosya yolunu içerir. Bu, resmin hangi klasörde bulunduğunu ve dosya adını ifade eder.

Örneğin: İlk satırda yol Train/20/00020\_00000\_00000.png olarak belirtilmiş. Bu, resmin Train/20 klasöründe bulunduğunu gösterir.

Satırları açıklamak gerekirse;

Her bir satır, bir trafik işareti resmine ait bilgileri içerir:

* İlk satırdaki trafik işareti resminin:
  + **Boyutu**: 27x26 piksel.
  + **ROI Alanı**: (5, 5) ile (22, 20) arasında.
  + **Sınıfı**: 20.
  + **Konumu**: Train/20/00020\_00000\_00000.png.

Aşağıdaki kodlarda ise veri ön işlem yapılmaktadır. Veri seti %80 eğitim ve %20 test verisi olarak ikiyi ayrılmaktadır.

Aynı zamanda verileri düzleştirilmesi gerekmektedir. Görüntüler makine öğrenimi algoritmalarına uygun hale getirmek için 1 D vektörlere dönüştürülür. (her 32x32x3 görüntü, 3072 boyutlu bir vektöre dönüştürülür)

Bunun yanında özellik ölçeklendirme ile görüntü verileri ölçeklendirilir. (ortalama 0, standart sapma 1 olacak şekilde normalize edilir.)

Matplotlib kütüphanesi ile verilerin bir kısmı görselleştirilir.

Ayrıca sınıf dağılımın dengeli olup olmadığını anlamak için inceleme yaparız.

**Özetle;**

***Veri önişleme*** ile eğitim ve test setlerine ayrılması, düzleştirilmesi, ölçeklemesi sağlanır.

***Görsellerin görselleştirmesi*** ile eğitim setinden birkaç görüntü sunulur.

***Sınıf dağılımının incelenmesi*** ile her sınıfın veri setindeki örnek sayısı hesaplanır.

#

import numpy as np

import os

from PIL import Image

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

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

import pandas as pd

# VERİ YÜKLEME

print("Veri yükleniyor...")

# Train.csv dosyasını oku

train\_df = pd.read\_csv('GTSRB\_data/Train.csv')

images = []

labels = []

print("Görüntüler yükleniyor...")

# Her bir görüntüyü yükle

for idx, row in train\_df.iterrows():

    try:

        img\_path = os.path.join('GTSRB\_data', row['Path'])

        if os.path.exists(img\_path):

            img = Image.open(img\_path)

            img = img.resize((32, 32))  # Resimleri yeniden boyutlandır

            img\_array = np.array(img)

            images.append(img\_array)

            labels.append(row['ClassId'])

            # Her 1000 görüntüde bir ilerleme göster

            if (idx + 1) % 1000 == 0:

                print(f"{idx + 1} görüntü yüklendi...")

        else:

            print(f"Dosya bulunamadı: {img\_path}")

    except Exception as e:

        print(f"Hata - {img\_path}: {str(e)}")

        continue

if len(images) == 0:

    raise ValueError("Hiç görüntü yüklenemedi!")

# Numpy dizilerine dönüştür

X = np.array(images)

y = np.array(labels)

print(f"\nToplam {len(X)} görüntü yüklendi.")

print(f"Görüntü boyutları: {X.shape}")

# VERİ ÖN İŞLEME

print("\nVeri önişleniyor...")

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Veriyi düzleştir (MLP için)

n\_samples\_train = X\_train.shape[0]

n\_samples\_test = X\_test.shape[0]

n\_features = 32 \* 32 \* 3  # 32x32 RGB görüntüler

X\_train\_flat = X\_train.reshape((n\_samples\_train, n\_features))

X\_test\_flat = X\_test.reshape((n\_samples\_test, n\_features))

# Özellik ölçekleme

scaler = StandardScaler()

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

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

print("\nVeri önişleme tamamlandı.")

print(f"X\_train\_scaled: {X\_train\_scaled.shape}, X\_test\_scaled: {X\_test\_scaled.shape}")

# ÖRNEK GÖRÜNTÜLERİ GÖRSELLEŞTİR

plt.figure(figsize=(15, 3))

num\_samples = 5  # Gösterilecek örnek sayısı

for i in range(min(num\_samples, len(X\_train))):

    plt.subplot(1, num\_samples, i + 1)

    plt.imshow(X\_train[i])

    plt.title(f'Label: {y\_train[i]}')

    plt.axis('off')

plt.tight\_layout()

plt.show()

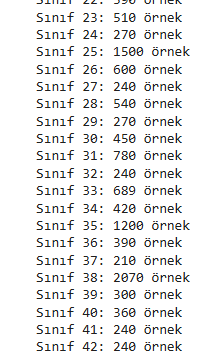
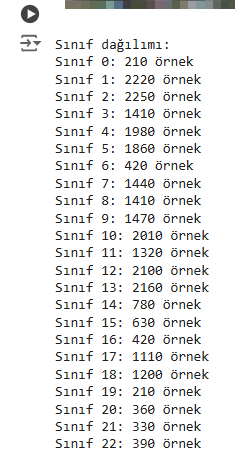
#SINIF DAĞILIMINI GÖSTER

print("\nSınıf dağılımı:")

for class\_id in np.unique(y):

    print(f"Sınıf {class\_id}: {np.sum(y == class\_id)} örnek")





Toplam 39209 görüntünün yüklendiği görülmektedir.

Görüntü boyutları: (39209, 32, 32, 3). Her bir görüntü 32x32 piksel çözünürlüğe ve 3 kanal (RGB) renk formatına sahiptir.

(39209, 32, 32, 3): İlk boyut toplam resim sayısını, sonraki boyutlar ise görüntülerin genişlik, yükseklik ve renk kanallarını temsil eder.

Görseller düşük çözünürlüklü ama trafik işaretlerinin temel özelliklerini öğrenmek için yeterlidir.

## MLP Modelinin Oluşturulması – Temel Versiyon

Çok katmanlı algılayıcı(MLP-Multi Layer Perceptron) ile GTSRB veri seti üzerinde sınıflamayı, önceki veri ön işleme sonrasında yapabiliriz;

**MLPClassifier**, **sklearn.neural\_network** modülünde yer alan bir sınıf olup, çok katmanlı ileri beslemeli sinir ağı oluşturmak ve eğitmek için kullanılır. Bizde bu sınıf model oluşturmak için kullanacağız.

mlp = MLPClassifier(hidden\_layer\_sizes=(100,), max\_iter=300, random\_state=42)

Burada tek gizli bir layer vardır. Bu bu katman 100 nörondan oluşuyor. Daha fazla katman eklemek veya her katman için farklı nöron sayıları belirlemek mümkündür. (örneğin, (100, 50, 30)).

Burada hiperparametreleri düzgün ayarlamalıyız. Çünkü fazla nöron aşırı öğrenme dediğimiz overfitting’e az nöron ise yetersiz öğrenme (underfitting) riskini artırabilir.

Aynı zamanda max\_iter= 300 ise; modelin eğitimi sırasında maksimum iterasyonu sayısını belirtir. İşte burada Gradient Descent gibi yöntemler ile modelin ağırlıklarını güncelleyerek optimuma ulaşmaya çalıştığımız süreçtir. Burada eğitim süresi uzun ve model optimuma ulaşmıyorsa artırılabilir. Çok küçük bir değer modelin yeterince öğrenememesine, çok büyük bir değer ise uzun eğitim sürelerine neden olabilir.

random\_state=42 ise rastgele işlemlerde kullanılan başlangıç durumunu belirler. Sonuçların tekrar edilebilir olması için kullanılır.

Model oluşturulurken ifade edilmeyen ama varsayılan olarak mevcut olan parametreler;

 **activation='relu'**: Aktivasyon fonksiyonu olarak ReLU (Rectified Linear Unit) kullanılır. Gizli katmanlarda nöronların doğrusal olmayan dönüşümünü gerçekleştirir.

 **solver='adam'**: Ağırlık optimizasyonu için Adam algoritması kullanılır. Hızlı ve etkili bir öğrenme algoritmasıdır.

 **alpha=0.0001**: L2 düzenleme terimi için katsayıdır. Overfitting'i önlemek için kullanılır.

 **learning\_rate='constant'**: Öğrenme oranı sabit olarak ayarlanır. Öğrenme oranını dinamik olarak değiştirmek istenirse (adaptive, invscaling) kullanılabilir.

# MLP Model - Temel Versiyon

from sklearn.neural\_network import MLPClassifier

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

import seaborn as sns

print("MLP modeli eğitiliyor...")

# Basit MLP modeli oluşturma

mlp = MLPClassifier(hidden\_layer\_sizes=(100,), max\_iter=300, random\_state=42)

# Modeli eğitme

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

# Tahminler

y\_pred\_mlp = mlp.predict(X\_test\_scaled)

# Performans değerlendirmesi

print("\nMLP Model Performansı:")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred\_mlp))

# Confusion Matrix

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

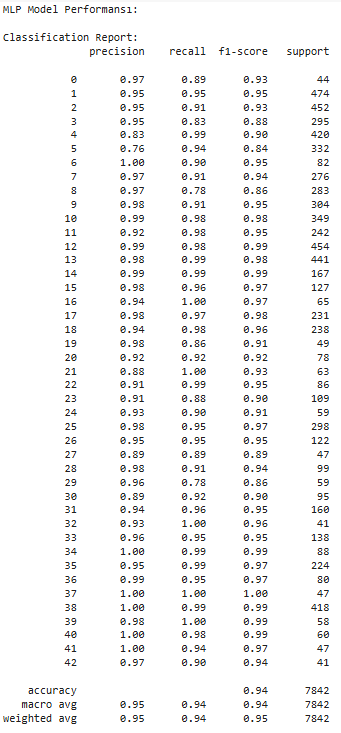
sns.heatmap(confusion\_matrix(y\_test, y\_pred\_mlp), annot=True, fmt='d', cmap='Blues')

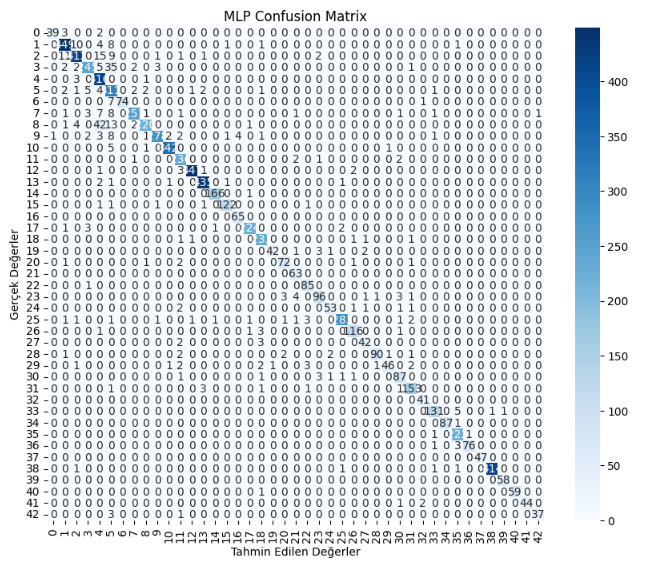
plt.title('MLP Confusion Matrix')

plt.ylabel('Gerçek Değerler')

plt.xlabel('Tahmin Edilen Değerler')

plt.show()





Metrikler;

1. **Precision**:
   * Bir sınıf için yapılan tahminlerin doğruluk oranıdır.
   * Örneğin: **Sınıf 0** için precision %97, yani modelin sınıf 0 olarak tahmin ettiği verilerin %97’si doğru.
2. **Recall**:
   * Bir sınıfın tüm gerçek örneklerinden kaç tanesinin doğru tahmin edildiğini gösterir.
   * Örneğin: **Sınıf 2** için recall %91, yani sınıf 2’ye ait gerçek örneklerin %91’i doğru tahmin edilmiş.
3. **F1-Score**:
   * Precision ve recall'un harmonik ortalamasıdır. Dengeli bir başarı ölçütüdür.
   * Örneğin: **Sınıf 36** için F1-score %97, bu da hem precision hem recall’un yüksek olduğunu gösterir.
4. **Support**:
   * Her sınıfa ait toplam örnek sayısıdır.
   * Örneğin: **Sınıf 1** için veri setinde 474 örnek bulunuyor.

Genel Metrikler;

* **Accuracy**:
  + Tüm sınıflar için toplam doğruluk oranıdır.
  + **Sonuç**: %94 doğruluk oranı ile model genel olarak iyi performans göstermektedir.
* **Macro Avg**:
  + Tüm sınıfların precision, recall ve F1-score’unun basit ortalamasıdır. Veri dengesizliği olan durumlarda önemlidir.
  + **Sonuç**: %95 ile sınıflar arasında ortalama performans oldukça iyidir.
* **Weighted Avg**:
  + Sınıflar arasındaki destek sayısına göre ağırlıklı ortalamadır. Veri dengesizliği olan veri setleri için daha gerçekçidir.
  + **Sonuç**: %94 ile model ağırlıklı olarak başarılıdır.

Dikkat Çekici Noktalar;

* **Sınıf 1, 3, 13 ve 36** gibi sınıflarda precision ve recall oldukça yüksek.
* **Sınıf 2 ve 8** gibi sınıflarda precision veya recall biraz düşüktür, bu da modelin bu sınıflarda daha fazla yanlış tahmin yaptığını gösterir.

Confusion Matrix;

Bu matriste, modelin her bir sınıf için doğru ve yanlış tahminleri görselleştirilmiştir:

1. **Diagonal Elemanlar**:
   * Gerçek ve tahmin edilen sınıfların eşleştiği doğru tahminlerdir.
   * Örneğin: Sınıf 1 için 410 doğru tahmin yapılmış.
2. **Yanlış Tahminler**:
   * Diagonal dışındaki elemanlar, yanlış tahmin edilen örneklerdir.
   * Örneğin: **Sınıf 2**, sınıf 1 ve 5 ile karıştırılmış (birkaç yanlış tahmin mevcut).

Genel Analiz;

* Çoğu sınıfta doğru tahmin oranı oldukça yüksektir (diagonal elemanlar baskındır).
* **Yanlış tahminler**, genelde benzer görsellerin olduğu sınıflar arasında ortaya çıkmış olabilir (örneğin, hız limitleri veya benzer şekilli işaretler).
* Matriste belirli sınıflarda hata yoğunluğu oldukça düşüktür, bu da modelin iyi genelleme yaptığını gösterir.

Sonuç ve Öneriler;

* **Model Performansı**: Model genel olarak iyi performans göstermekte, doğruluk %94 civarındadır.
* **İyileştirme Potansiyeli**:
  + **Sınıf 2 ve 8** gibi recall veya precision düşük olan sınıflar üzerinde çalışılabilir. Bu sınıflar için veri artırma (data augmentation) veya sınıf ağırlıklarını dengeleme yöntemleri kullanılabilir.
  + Model hiperparametreleri (hidden\_layer\_sizes, max\_iter) optimize edilerek performans artırılabilir.
  + Yanlış tahmin edilen örneklerin analizi ile modelin daha fazla öğrenmesi sağlanabilir.
* **Genel Yorum**: MLP modeli, dengesiz sınıflara rağmen geniş veri setinde oldukça başarılı sonuçlar elde etmiştir. Confusion matrix üzerinden daha detaylı bir hata analizi yapılabilir.

## RBF Modelinin Oluşturulması – Temel Versiyon

Bu bölümde ise RBF yani (Radial Basis Function) modeli tarzında bir sınıflandırıcı oluşturmayı amaçlayan modeli kurgulamaya çalışacağız.

Geleneksel RBF ağlarında, gizli katmanlar Gaussian tabanlı aktivasyon fonksiyonları kullanır. Bu kodda ise, gizli katmanlarda ReLU aktivasyonu kullanarak daha yaygın bir ANN mimarisi tercih edilmiştir.

Gizli katmanlar, giriş özelliklerinden önemli temsiller öğrenir.

Çıkış katmanı her sınıfa ait olasılık değerlerini üretir ve en yüksek olasılığa sahip sınıf modelin tahmini olarak seçilir.

Öğrenme sürecinde ise ağırlıklar Adam optimizasyon algoritması ile her iterasyonda güncellenir. Kayıp fonksiyonu (crossentropy) modelin doğru tahmin yapması için hata değerlerini minimize eder.

# RBF Model - Temel Versiyon

from tensorflow import keras

from tensorflow.keras import layers

def create\_rbf\_model(input\_shape, num\_classes):

    model = keras.Sequential([

        layers.Dense(100, input\_shape=(input\_shape,), activation='relu'),

        layers.Dense(50, activation='relu'),

        layers.Dense(num\_classes, activation='softmax')

    ])

    model.compile(

        optimizer='adam',

        loss='sparse\_categorical\_crossentropy',

        metrics=['accuracy']

    )

    return model

# RBF modeli oluşturma ve eğitme

print("RBF modeli eğitiliyor...")

input\_shape = X\_train\_scaled.shape[1]

num\_classes = len(np.unique(y\_train))

rbf\_model = create\_rbf\_model(input\_shape, num\_classes)

history = rbf\_model.fit(

    X\_train\_scaled, y\_train,

    epochs=50,

    batch\_size=32,

    validation\_split=0.2,

    verbose=1

)

# RBF model performans değerlendirmesi

y\_pred\_rbf = np.argmax(rbf\_model.predict(X\_test\_scaled), axis=1)

print("\nRBF Model Performansı:")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred\_rbf))

# Confusion Matrix

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

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_rbf), annot=True, fmt='d', cmap='Blues')

plt.title('RBF Confusion Matrix')

plt.ylabel('Gerçek Değerler')

plt.xlabel('Tahmin Edilen Değerler')

plt.show()

# Eğitim geçmişini görselleştirme

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

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

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

plt.title('RBF Model Accuracy')

plt.legend()

plt.subplot(1, 2, 2)

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

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

plt.title('RBF Model Loss')

plt.legend()

plt.show()

Hiperparametreler ve Özelliker;

Katman Boyutları (hidden\_layer\_sizes):

* İlk gizli katmanda 100, ikincide 50 nöron kullanılmış. Daha fazla nöron, modelin kapasitesini artırırken aşırı öğrenme riskini de artırabilir.

Aktivasyon Fonksiyonları (activation):

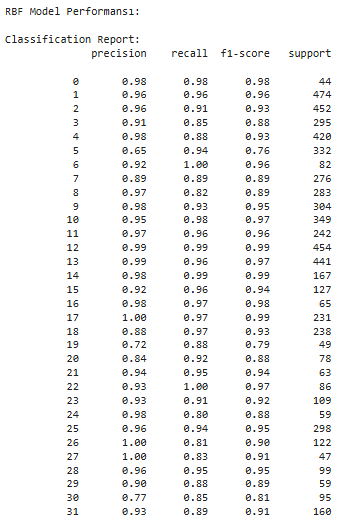
* ReLU: Hızlı öğrenme sağlar ve gradyan problemini hafifletir.
* Softmax: Çok sınıflı sınıflandırma için kullanılır.

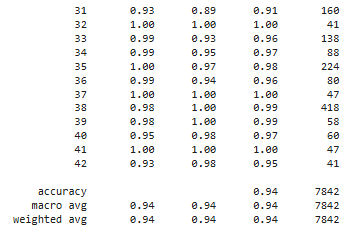
Optimizasyon Algoritması (optimizer):

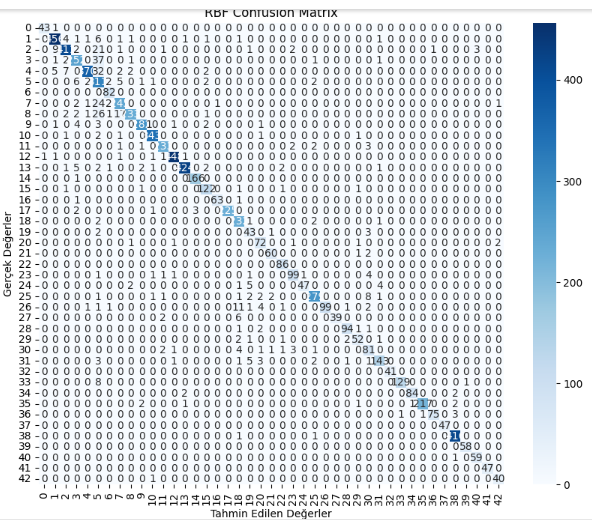
* Adam: Uyarlanabilir öğrenme oranıyla hızlı ve etkili bir optimizasyon sağlar.

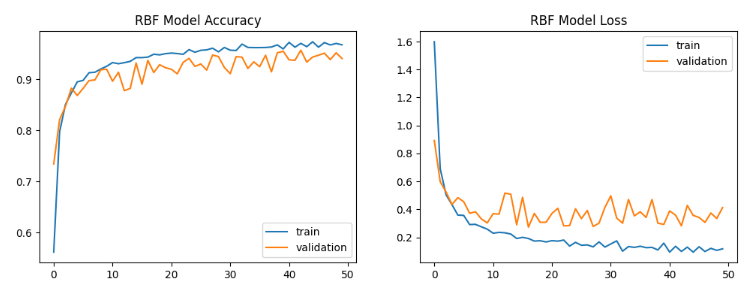
Eğitim Parametreleri:

* epochs: Modelin ne kadar sürede eğitileceğini belirler. 50 epoch genelde yeterli, ancak bu değer artırılabilir.
* batch\_size: Veri seti küçük gruplar halinde eğitilir. Daha küçük bir batch size daha hassas bir güncelleme sağlar ancak eğitim süresi artabilir.









Sonuç ve Çıktıların Yorumu;

* **Accuracy**: Modelin genel doğruluğu %94'tür.
* **Macro Average**: Tüm sınıfların eşit ağırlıkla ortalaması %94. Bu, modelin sınıflar arasında dengeli olduğunu gösterir.
* **Weighted Average**: Sınıf örnek sayısına göre ağırlıklı ortalama %94. Veri dengesizliği varsa bile modelin iyi bir performans sergilediğini belirtir.

Sınıf Bazlı Analiz;

* Çoğu sınıfta precision, recall ve F1-score değerleri oldukça yüksektir (%90-%100 arası).
* **Düşük Performanslı Sınıflar**:
  + **Sınıf 30**: Precision %77, recall %85, F1-score %81. Bu, modelin bu sınıfta bazı karışıklıklar yaşadığını gösterir.
  + **Sınıf 28 ve 29**: Precision ve recall'da diğer sınıflara göre biraz daha düşük performans gözlemlenmiştir.

Confusion Matrix;

 **Diagonal Elemanlar**: Doğru sınıflandırmaları temsil eder. Matriste diagonal yoğunluğu oldukça yüksektir, bu da modelin büyük ölçüde doğru tahmin yaptığını gösterir.

 **Yanlış Tahminler**:

* Sınıf 30 ile diğer bazı sınıflar arasında karışıklık gözlenmiştir. Örneğin, Sınıf 30 bazen Sınıf 29 veya Sınıf 28 olarak yanlış tahmin edilmiştir.
* Hatalar genellikle şekil veya görsel benzerliklerden kaynaklanabilir (örneğin, hız limitleri veya benzer uyarı işaretleri).

Model Accuracy(Doğruluk)

* Eğitim doğruluğu, epoch sayısı arttıkça sürekli artmıştır.
* Doğrulama doğruluğu (validation accuracy) da genel olarak yükselmiştir ve eğitim doğruluğuna oldukça yakındır.
* **Overfitting (Aşırı Öğrenme)**: Doğrulama doğruluğu ve eğitim doğruluğu arasında büyük bir fark olmadığından, modelin overfitting yapmadığı söylenebilir.

Model Loss(Kayıp)

* Eğitim kaybı epoch sayısı arttıkça düzenli olarak azalmıştır.
* Doğrulama kaybı (validation loss) da azalmış, ancak bazı epoch'larda ufak dalgalanmalar gözlenmiştir. Bu durum, modelin doğrulama setinde küçük bir genelleme problemi yaşayabileceğini gösterebilir.

Sonuç ve Öneriler;

1. **Model Performansı**:
   * Genel doğruluk %94 ile oldukça başarılıdır.
   * Sınıf dengesizliğine rağmen çoğu sınıfta precision, recall ve F1-score değerleri oldukça yüksektir.
2. **İyileştirme Alanları**:
   * **Sınıf 30, 28 ve 29** gibi düşük performans gösteren sınıflar için veri artırma (data augmentation) yöntemleri uygulanabilir.
   * Yanlış sınıflandırılan örneklerin analizi yapılarak, bu sınıflar arasındaki karışıklığın nedenleri belirlenebilir.
   * Modelin hiperparametreleri (örneğin, katman sayısı, nöron sayısı, öğrenme oranı) daha optimize edilerek daha iyi sonuçlar elde edilebilir.
3. **Genel Yorum**:
   * RBF modeli, veri setindeki çoğu sınıfta yüksek doğruluk sağlamış ve overfitting yapmamıştır.
   * Eğitim süreci ve doğrulama süreci dengeli ilerlemiş, dolayısıyla modelin genelleme kapasitesi oldukça iyidir.
   * Model, trafik işareti sınıflandırma gibi görevler için iyi bir başlangıç noktası sunar.

## MLP İçin Hiperparametre Optimizasyonu

Hiperparametre için GridSearch kütüphanesini kullanalım;

# MLP için Hiperparametre Optimizasyonu

from sklearn.model\_selection import GridSearchCV

# Hiperparametre arama alanını tanımlama

mlp\_params = {

    'hidden\_layer\_sizes': [(50,), (100,), (50,50), (100,50)],

    'activation': ['relu', 'tanh'],

    'learning\_rate\_init': [0.001, 0.01],

    'max\_iter': [300],

    'solver': ['adam', 'sgd']

}

# GridSearchCV ile en iyi parametreleri bulma

print("MLP için hiperparametre optimizasyonu başlıyor...")

mlp\_grid = GridSearchCV(MLPClassifier(random\_state=42),

                       mlp\_params,

                       cv=3,

                       n\_jobs=-1,

                       verbose=1)

mlp\_grid.fit(X\_train\_scaled, y\_train)

print("\nEn iyi MLP parametreleri:", mlp\_grid.best\_params\_)

print("En iyi cross-validation skoru:", mlp\_grid.best\_score\_)

# En iyi model ile tahminler

y\_pred\_mlp\_best = mlp\_grid.predict(X\_test\_scaled)

print("\nEn İyi MLP Modeli Performansı:")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred\_mlp\_best))

# Confusion Matrix

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

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_mlp\_best), annot=True, fmt='d', cmap='Blues')

plt.title('En İyi MLP Modeli - Confusion Matrix')

plt.ylabel('Gerçek Değerler')

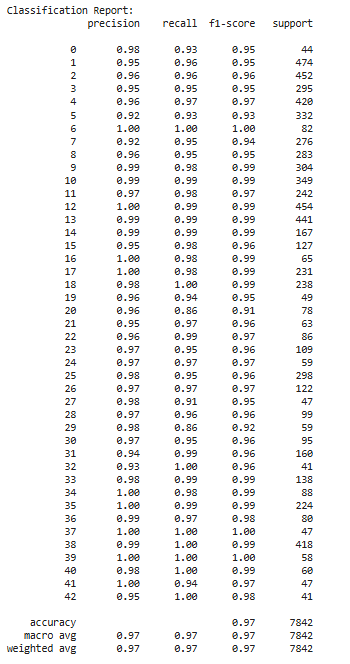
plt.xlabel('Tahmin Edilen Değerler')

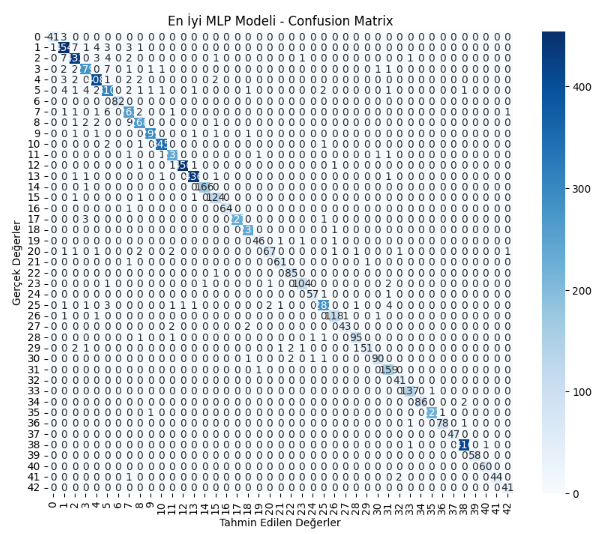
plt.show()

**Hiperparametreler**:

* **hidden\_layer\_sizes**: Gizli katmanlardaki nöron sayıları ve katman yapısı.
  + Örneğin, (50,) tek bir gizli katmanda 50 nöron, (50,50) iki gizli katmanda 50 nöron içerir.
* **activation**: Aktivasyon fonksiyonu.
  + relu: Lineer olmayan hızlı bir öğrenme fonksiyonu.
  + tanh: Hyperbolik tanjant fonksiyonu.
* **learning\_rate\_init**: Öğrenme oranı. Modelin ağırlıkları ne kadar hızlı güncellediğini belirler.
  + Küçük değerler daha hassas güncellemeler, büyük değerler daha hızlı yakınsama sağlar.
* **max\_iter**: Eğitim sırasında maksimum iterasyon sayısı. Sabit tutulmuş: 300.
* **solver**: Optimizasyon algoritması.
  + adam: Uyarlanabilir öğrenme oranına sahip popüler bir algoritma.
  + sgd: Stokastik gradyan inişi.

Kodun çıktıları;





Genel Metrikler;

* **Accuracy**: Modelin genel doğruluğu %97'dir. Bu, test setindeki tüm örneklerin %97’sinin doğru tahmin edildiğini gösterir.
* **Macro Average**: Precision, recall ve F1-score'un eşit ağırlıklı ortalaması %97. Bu, modelin tüm sınıflar arasında dengeli bir performans sergilediğini gösterir.
* **Weighted Average**: Sınıf örnek sayısına göre ağırlıklı ortalama %97. Veri dengesizliği olsa bile, modelin performansı yüksektir.

Sınıf Bazlı Analiz

* Çoğu sınıfta precision, recall ve F1-score değerleri %95-%100 arasında olup oldukça yüksektir.
* Dikkat Çeken Sınıflar: Sınıf 1, 7, 13, 37 ve 41:

F1-score değerleri %99-100 arasında. Bu sınıflar, modelin en iyi performans gösterdiği sınıflardır.

* Sınıf 28, 29 ve 30: Performansı diğer sınıflara göre daha düşüktür. Precision, recall ve F1-score değerleri %90 civarındadır.

Destek(Support)

* Destek sütunu, her sınıfta test setinde kaç örnek olduğunu gösterir.
  + Örneğin:
    - **Sınıf 1**: 474 örnek (model bu sınıfta oldukça başarılı).
    - **Sınıf 30**: 59 örnek (performans diğer sınıflara kıyasla daha düşük).

Confusion Matrix

* **Diagonal Elemanlar**: Modelin doğru tahminlerini gösterir. Çoğu sınıfta doğru tahminler çok yüksektir.
  + Örneğin:
    - Sınıf 13: 441 doğru tahmin yapılmış.
    - Sınıf 30: 51 doğru tahmin yapılmış, ancak bu sınıf için diğer sınıflara göre karışıklık oranı daha yüksektir.
* **Yanlış Tahminler**:
  + Diagonal dışındaki elemanlar, yanlış tahmin edilen örnekleri temsil eder.
  + Örneğin:
    - Sınıf 30, sıklıkla Sınıf 29 ve Sınıf 28 ile karıştırılmıştır. Bu, görsel benzerliklerden kaynaklanabilir (örneğin, hız limitleri veya uyarı işaretleri).
    - Sınıf 28, bazen Sınıf 27 ile karıştırılmıştır.

**Genel Analiz**

* Model, çoğu sınıfta yüksek doğruluk sağlamıştır.
* Yanlış tahminler, özellikle görsel benzerliği yüksek olan sınıflar arasında yoğunlaşmıştır (örneğin, hız limitleri ve uyarı işaretleri).

Sonuç ve Öneriler

1. **Performans Değerlendirmesi**:
   * Optimize edilmiş MLP modeli, %97 doğruluk oranıyla oldukça başarılıdır.
   * Precision, recall ve F1-score değerleri genel olarak yüksektir, bu da modelin sınıflar arasında dengeli bir performans sergilediğini gösterir.
2. **İyileştirme Alanları**:
   * **Sınıf 28, 29 ve 30** gibi düşük performans gösteren sınıflar için veri artırma (data augmentation) veya daha fazla örnek ekleme gibi yöntemler kullanılabilir.
   * Yanlış tahmin edilen örnekler analiz edilerek modelin sınıflar arasındaki farkları daha iyi öğrenmesi sağlanabilir.
3. **Genel Yorum**:
   * GridSearchCV ile optimize edilen hiperparametreler, modelin performansını artırmıştır.
   * Çoğu sınıfta yüksek doğruluk sağlayarak, trafik işareti sınıflandırma gibi çok sınıflı problemler için güçlü bir çözüm sunmuştur.
   * Yanlış tahminlerin yoğunlaştığı sınıflar üzerine odaklanarak model daha da iyileştirilebilir.

## RBF İçin Hiperparametre Optimizasyonu

* neurons1\_options: İlk gizli katmandaki nöron sayıları ([64, 128, 256]).
* neurons2\_options: İkinci gizli katmandaki nöron sayıları ([32, 64, 128]).
* learning\_rates: Öğrenme oranları ([0.001, 0.01]).

for n1 in neurons1\_options:

for n2 in neurons2\_options:

for lr in learning\_rates:

...

Üçlü döngü ile tüm olası hiperparametre kombinasyonları test edilir.

Her bir kombinasyonda:

* Model oluşturulur ve eğitim yapılır.
* Test seti üzerinde performans (test\_accuracy) değerlendirilir.
* Performans sonuçları bir listeye kaydedilir.

Aşağıdaki kod ile de Early Stopping yapılır; Eğer doğrulama kaybı (val\_loss) 5 epoch boyunca iyileşmezse eğitim durdurulur ve en iyi model ağırlıkları geri yüklenir.

early\_stopping = EarlyStopping(

monitor='val\_loss',

patience=5,

restore\_best\_weights=True

)

# RBF Hiperparametrezisyon için Gerekli kütüphaneleri import edelim

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.callbacks import EarlyStopping

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

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

def create\_rbf\_model\_with\_params(neurons1, neurons2, learning\_rate, input\_dim, num\_classes):

    """

    RBF modelini tanımlayan ve döndüren fonksiyon.

    """

    model = keras.Sequential([

        layers.Dense(neurons1, input\_shape=(input\_dim,), activation='relu'),

        layers.Dense(neurons2, activation='relu'),

        layers.Dropout(0.2),

        layers.Dense(num\_classes, activation='softmax')

    ])

    optimizer = keras.optimizers.Adam(learning\_rate=learning\_rate)

    model.compile(

        optimizer=optimizer,

        loss='sparse\_categorical\_crossentropy',

        metrics=['accuracy']

    )

    return model

# Hiperparametreler

neurons1\_options = [64, 128, 256]

neurons2\_options = [32, 64, 128]

learning\_rates = [0.001, 0.01]

# Veri girişi boyutu ve sınıf sayısı

input\_dim = X\_train\_scaled.shape[1]  # Özellik sayısı

num\_classes = len(np.unique(y\_train))  # Sınıf sayısı

# Sonuçları saklamak için

results = []

# Early stopping

early\_stopping = EarlyStopping(

    monitor='val\_loss',

    patience=5,

    restore\_best\_weights=True

)

print("RBF için hiperparametre optimizasyonu başlıyor...")

# Grid search

for n1 in neurons1\_options:

    for n2 in neurons2\_options:

        for lr in learning\_rates:

            print(f"\nTest ediliyor: neurons1={n1}, neurons2={n2}, learning\_rate={lr}")

            # Model oluştur

            model = create\_rbf\_model\_with\_params(n1, n2, lr, input\_dim, num\_classes)

            # Modeli eğit

            history = model.fit(

                X\_train\_scaled, y\_train,

                epochs=50,

                batch\_size=32,

                validation\_split=0.2,

                callbacks=[early\_stopping],

                verbose=0

            )

            # Test seti üzerinde değerlendirme

            test\_loss, test\_acc = model.evaluate(X\_test\_scaled, y\_test, verbose=0)

            results.append({

                'neurons1': n1,

                'neurons2': n2,

                'learning\_rate': lr,

                'test\_accuracy': test\_acc

            })

# En iyi sonucu bulma

best\_result = max(results, key=lambda x: x['test\_accuracy'])

print("\nEn iyi RBF parametreleri:", best\_result)

# En iyi model ile son bir eğitim

best\_model = create\_rbf\_model\_with\_params(

    best\_result['neurons1'],

    best\_result['neurons2'],

    best\_result['learning\_rate'],

    input\_dim,

    num\_classes

)

history = best\_model.fit(

    X\_train\_scaled, y\_train,

    epochs=50,

    batch\_size=32,

    validation\_split=0.2,

    callbacks=[early\_stopping],

    verbose=1

)

# En iyi model performans değerlendirmesi

y\_pred\_rbf\_best = np.argmax(best\_model.predict(X\_test\_scaled), axis=1)

print("\nEn İyi RBF Modeli Performansı:")

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred\_rbf\_best))

# Confusion Matrix

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

sns.heatmap(confusion\_matrix(y\_test, y\_pred\_rbf\_best), annot=True, fmt='d', cmap='Blues')

plt.title('En İyi RBF Modeli - Confusion Matrix')

plt.ylabel('Gerçek Değerler')

plt.xlabel('Tahmin Edilen Değerler')

plt.show()

# Eğitim geçmişini görselleştirme

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

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

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

plt.title('En İyi RBF Model - Accuracy')

plt.legend()

plt.subplot(1, 2, 2)

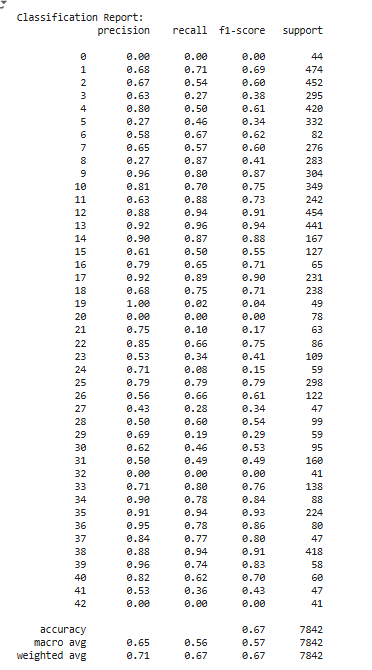
plt.plot(history.history['loss'], label='train')

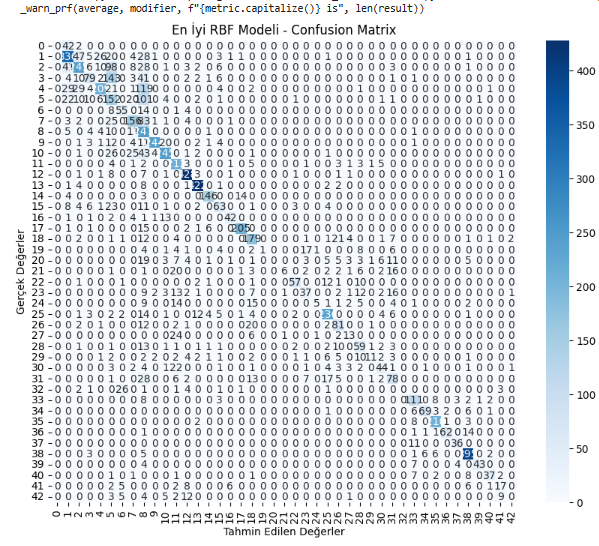
plt.plot(history.history['val\_loss'], label='validation')

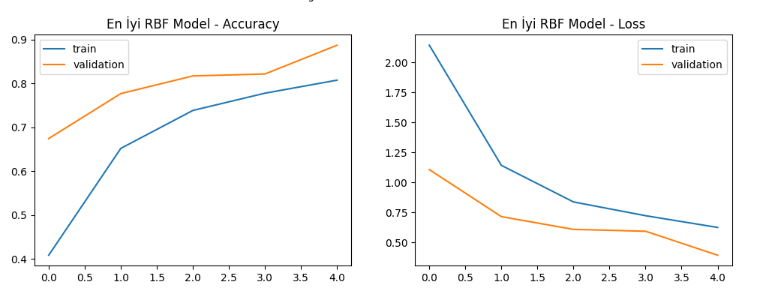
plt.title('En İyi RBF Model - Loss')

plt.legend()

plt.show()







Bu sonuçlar, RBF modeli için yapılan **hiperparametre optimizasyonu** sonucunda elde edilen "en iyi" modelin performansını göstermektedir. Ancak sonuçlar, beklenenden daha düşük bir performansı işaret etmektedir. Temel versiyonla karşılaştırıldığında bu modelin performansı daha düşüktür.

Hiperparametre optimizasyonunun sonuçları genelde daha iyi performans sunarken, bu örnekte performans düşüşü gözlemlenmiştir. Bu, seçilen arama alanının model için uygun olmadığını veya eğitimin yetersiz olduğunu gösterebilir.

Aytekin Özel-Üsküdar Üniversitesi