**ÜSKÜDAR ÜNİVERSİTESİ**

**FEN BİLİMLERİ ENSTİTÜSÜ**

**YAPAY SİNİR AĞI-Ara sınav Ödevi**

**Furkan Sukan**

**234329008**

[**https://colab.research.google.com/drive/1yQvZ2WJm0qlZDuRaOe7CTm1Tn49ShRDt?usp=sharing**](https://colab.research.google.com/drive/1yQvZ2WJm0qlZDuRaOe7CTm1Tn49ShRDt?usp=sharing)

**Ödev Yönergeleri**

**1. Gerekli Kütüphanelerin Yüklenmesi**

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

pip install scikit-learn keras torch torchvision requests scikeras

pip install scikeras

import torch

import joblib

import numpy as np

import pandas as pd

import seaborn as sns

from keras.layers import Dense

from collections import Counter

import matplotlib.pyplot as plt

from keras.models import Sequential

from torchvision import datasets, transforms

from sklearn.preprocessing import StandardScaler

from sklearn.neural\_network import MLPClassifier

from torch.utils.data import DataLoader, TensorDataset

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

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

**2. Veri Seti Yükleme ve Ön İşleme**

1. **Veri Yükleme**:

transform = transforms.Compose([

transforms.Resize((32, 32)), # Resize images to 32x32

transforms.ToTensor(),

])

train\_data = datasets.GTSRB(root='path/to/gtsrb\_root/', split='train', download=True, transform=transform)

test\_data = datasets.GTSRB(root='path/to/gtsrb\_root/', split='test', download=True, transform=transform)

train\_loader = torch.utils.data.DataLoader(train\_data, batch\_size=4, shuffle=True, num\_workers=2)

test\_loader = torch.utils.data.DataLoader(test\_data, batch\_size=4, shuffle=True, num\_workers=2)

1. **Eksik Değerlerin İşlenmesi**: Eğer veri setinde eksik değerler varsa, uygun yöntemlerle doldurulmalıdır.

def show\_images(loader):

data\_iter = iter(loader)

images, labels = next(data\_iter)

fig, axes = plt.subplots(1, len(images), figsize=(12, 3))

for i, img in enumerate(images):

axes[i].imshow(img.permute(1, 2, 0)) # Permute to change tensor shape for matplotlib

axes[i].set\_title(f"Label: {labels[i].item()}")

axes[i].axis('off')

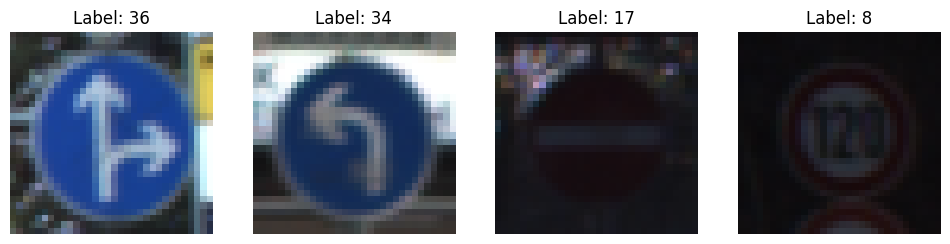
plt.show()

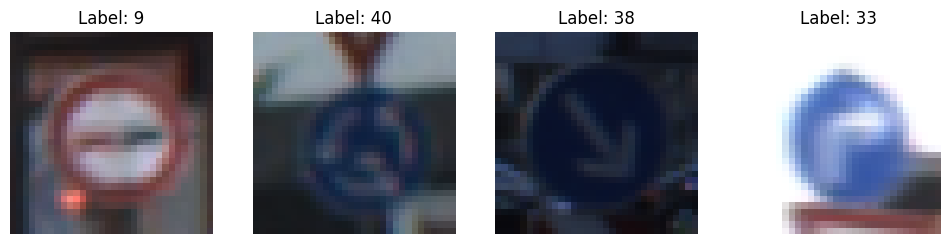
print("Eğitim verilerinden örnek görüntüler:")

show\_images(train\_loader)

print("Test verilerinden örnek görüntüler:")

show\_images(test\_loader)





# Her bir veri kümesindeki sınıf dağılımlarını sayma

def count\_labels(dataset):

labels = [label for \_, label in dataset]

label\_counts = Counter(labels)

return label\_counts

# Eğitim veri seti için sınıf frekanslarını hesaplama

train\_labels = [label for \_, label in train\_data]

train\_label\_counts = Counter(train\_labels)

# Test veri seti için sınıf frekanslarını hesaplama

test\_labels = [label for \_, label in test\_data]

test\_label\_counts = Counter(test\_labels)

# Histogram çizme fonksiyonu

def plot\_histogram(label\_counts, title):

labels, counts = zip(\*sorted(label\_counts.items()))

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

bars = plt.bar(labels, counts, color='skyblue')

plt.title(title)

plt.xlabel("Sınıf Etiketi")

plt.ylabel("Frekans")

# Çubukların üstüne frekans değerlerini ekleme

for bar, count in zip(bars, counts):

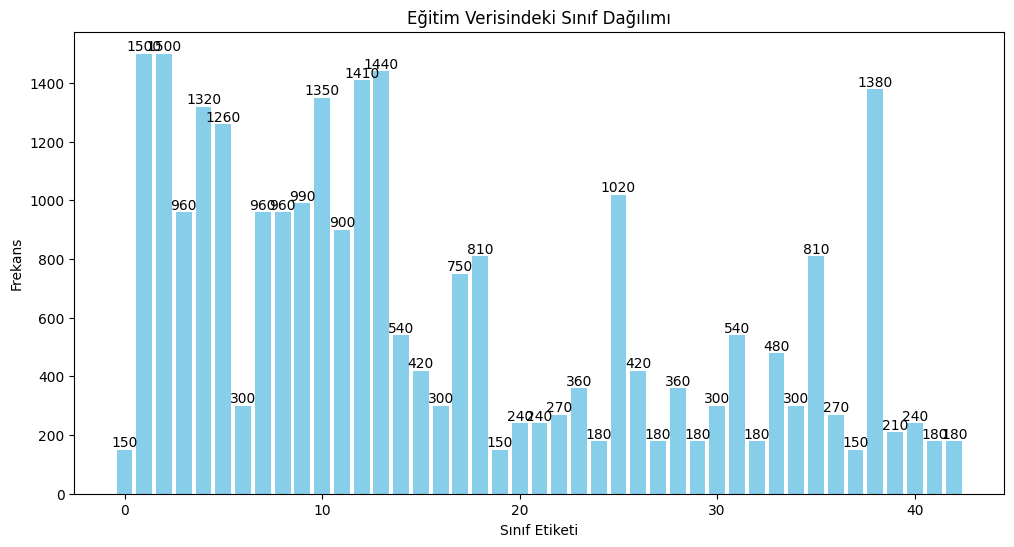
plt.text(bar.get\_x() + bar.get\_width() / 2, bar.get\_height(), str(count), ha='center', va='bottom')

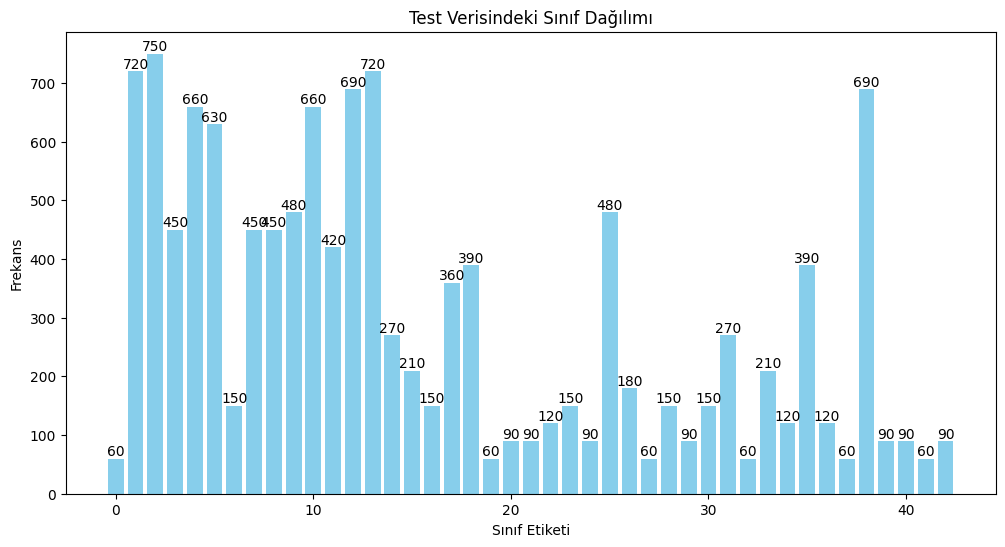
plt.show()

# Eğitim ve test veri setleri için histogramları çizdirme

plot\_histogram(train\_label\_counts, "Eğitim Verisindeki Sınıf Dağılımı")

plot\_histogram(test\_label\_counts, "Test Verisindeki Sınıf Dağılımı")





**Eğitim ve test verisindeki dağılımlar grafikte gösterilmiştir. Sınıflar arası dağılım dengeli değildir. Bu dengesiz dağılım overfitting’e götürebilir. Dengeli dağılım için fotoğrafların boyutları özellikleri ile feature engineering yapılabilir, scale, rotation gibi özellikler kullanılabilir.**

1. **Veri Setini Eğitim ve Test Olarak Ayırma**:
   * Veriyi %80 eğitim, %20 test olarak ikiye ayırın.
   * sklearn.model\_selection.train\_test\_split fonksiyonunu kullanın.

# Eğitim verilerini ve etiketlerini ayırma

images = torch.stack([item[0] for item in train\_data]) # Tüm görüntüleri alıyoruz

labels = torch.tensor([item[1] for item in train\_data]) # Tüm etiketleri alıyoruz

# Veriyi 0.8 - 0.2 oranında ayırma

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

# Yeni veri setlerini TensorDataset'e dönüştürme

train\_dataset = TensorDataset(X\_train, y\_train)

val\_dataset = TensorDataset(X\_val, y\_val)

# DataLoader'ları oluşturma

train\_loader = DataLoader(train\_dataset, batch\_size=4, shuffle=True, num\_workers=2)

val\_loader = DataLoader(val\_dataset, batch\_size=4, shuffle=True, num\_workers=2)

1. **Özellik Ölçekleme (Feature Scaling)**:
   * Verin (örn. StandartScaler veya MinMaxScaler kullanarleri MLP ve RBF için uygun hale getirmek adına standartlaştırıak).

# StandartScaler ile veriyi ölçeklendirme

scaler = StandardScaler()

# X\_train'in boyutunu (num\_samples, features) olarak değiştirme

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

num\_features = X\_train.shape[1] \* X\_train.shape[2] \* X\_train.shape[3]

X\_train\_reshaped = X\_train.reshape(num\_samples, num\_features)

# Eğitim verilerini ölçeklendir

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

# X\_val'in boyutunu da aynı şekilde değiştirme

num\_samples\_val = X\_val.shape[0]

X\_val\_reshaped = X\_val.reshape(num\_samples\_val, num\_features)

# Doğrulama verilerini aynı scaler ile ölçeklendir

X\_val\_scaled = scaler.transform(X\_val\_reshaped)

# Ölçeklendirilmiş verileri orijinal şekline geri döndürme

X\_train\_scaled = torch.tensor(X\_train\_scaled.reshape(X\_train.shape), dtype=torch.float32)

X\_val\_scaled = torch.tensor(X\_val\_scaled.reshape(X\_val.shape), dtype=torch.float32)

# Yeni veri setlerini TensorDataset'e dönüştürme

train\_dataset = TensorDataset(X\_train\_scaled, y\_train)

val\_dataset = TensorDataset(X\_val\_scaled, y\_val)

# DataLoader'ları oluşturma

train\_loader = DataLoader(train\_dataset, batch\_size=4, shuffle=True, num\_workers=2)

val\_loader = DataLoader(val\_dataset, batch\_size=4, shuffle=True, num\_workers=2)

**3. Model Eğitimi ve Hiperparametre Denemeleri**

1. **MLP Modeli**:
   * sklearn.neural\_network.MLPClassifier kullanarak bir MLP modeli oluşturun.
   * Hiperparametre denemeleri için farklı katman sayıları, nöron sayıları ve aktivasyon fonksiyonlarını test edin (örn. hidden\_layer\_sizes, activation ve learning\_rate gibi parametreler).

from sklearn.neural\_network import MLPClassifier

from sklearn.model\_selection import GridSearchCV

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

import matplotlib.pyplot as plt

import seaborn as sns

import joblib

import numpy as np

# 1. Eğitim ve doğrulama verilerini numpy dizilerine çevirme

X\_train\_np = X\_train\_scaled.numpy()

y\_train\_np = y\_train.numpy()

X\_val\_np = X\_val\_scaled.numpy()

y\_val\_np = y\_val.numpy()

# Reshape the input data to 2D before fitting the model

# Flatten the image data: (num\_samples, channels, height, width) -> (num\_samples, channels \* height \* width)

X\_train\_np = X\_train\_np.reshape(X\_train\_np.shape[0], -1)

X\_val\_np = X\_val\_np.reshape(X\_val\_np.shape[0], -1)

# 2. MLPClassifier ile model oluşturma

mlp = MLPClassifier(max\_iter=1000)

# --- Modelin ilk halinin değerlendirilmesi ---

# Modeli eğitiyoruz

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

# İlk modelin tahminlerini yapma

y\_pred\_initial = mlp.predict(X\_val\_np)

# İlk modelin performans metriklerini hesaplama

print("Initial Model - Classification Report:")

print(classification\_report(y\_val\_np, y\_pred\_initial))

# İlk modelin Confusion Matrix grafiğini çizme

conf\_matrix\_initial = confusion\_matrix(y\_val\_np, y\_pred\_initial)

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

sns.heatmap(conf\_matrix\_initial, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y\_val\_np), yticklabels=np.unique(y\_val\_np))

plt.title('Initial Model - Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# --- Hiperparametre optimizasyonu ve GridSearchCV ile en iyi modelin eğitilmesi ---

param\_grid = {

'hidden\_layer\_sizes': [(50,), (100,), (50, 50), (100, 50), (100, 100)], # Farklı gizli katman yapıları ve boyutları

'activation': ['tanh', 'relu'],

'solver': ['lbfgs', 'sgd', 'adam'],

'alpha': [0.001, 0.01, 0.1],

'learning\_rate': ['adaptive'],

'learning\_rate\_init': [0.001, 0.01, 0.1],

'max\_iter': [200, 300, 500]

}

# GridSearchCV ile çapraz doğrulama ve hiperparametre optimizasyonu

grid\_search = GridSearchCV(mlp, param\_grid, cv=5, n\_jobs=-1, verbose=1)

grid\_search.fit(X\_train\_np, y\_train\_np)

# En iyi parametreleri yazdırma

print(f"En iyi parametreler: {grid\_search.best\_params\_}")

# --- Hiperparametre optimizasyonu sonrası modelin değerlendirilmesi ---

# En iyi modelin tahminlerini yapma

best\_model = grid\_search.best\_estimator\_

y\_pred\_best = best\_model.predict(X\_val\_np)

# En iyi modelin performans metriklerini hesaplama

print("Optimized Model - Classification Report:")

print(classification\_report(y\_val\_np, y\_pred\_best))

# En iyi modelin Confusion Matrix grafiğini çizme

conf\_matrix\_best = confusion\_matrix(y\_val\_np, y\_pred\_best)

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

sns.heatmap(conf\_matrix\_best, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y\_val\_np), yticklabels=np.unique(y\_val\_np))

plt.title('Optimized Model - Confusion Matrix')

plt.xlabel('Predicted')

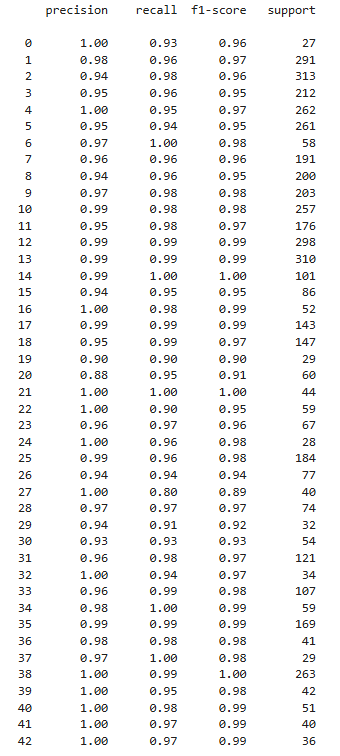
plt.ylabel('True')

plt.show()

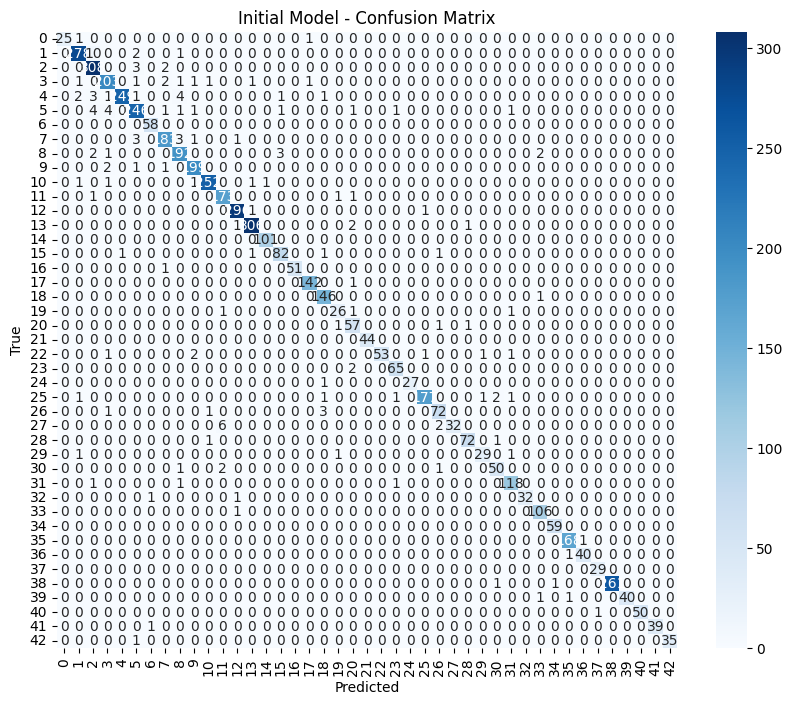
joblib.dump(best\_model, 'best\_mlp\_model.pkl')

**MLP Base Model**

ACC : 0.97

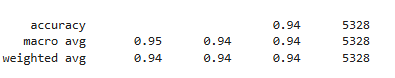
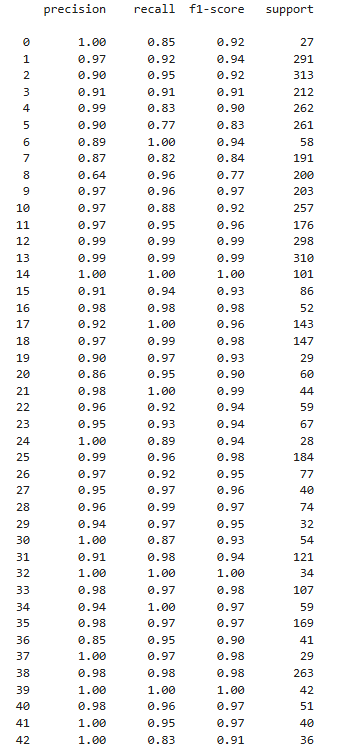


**Base Model Confusion Matrix**

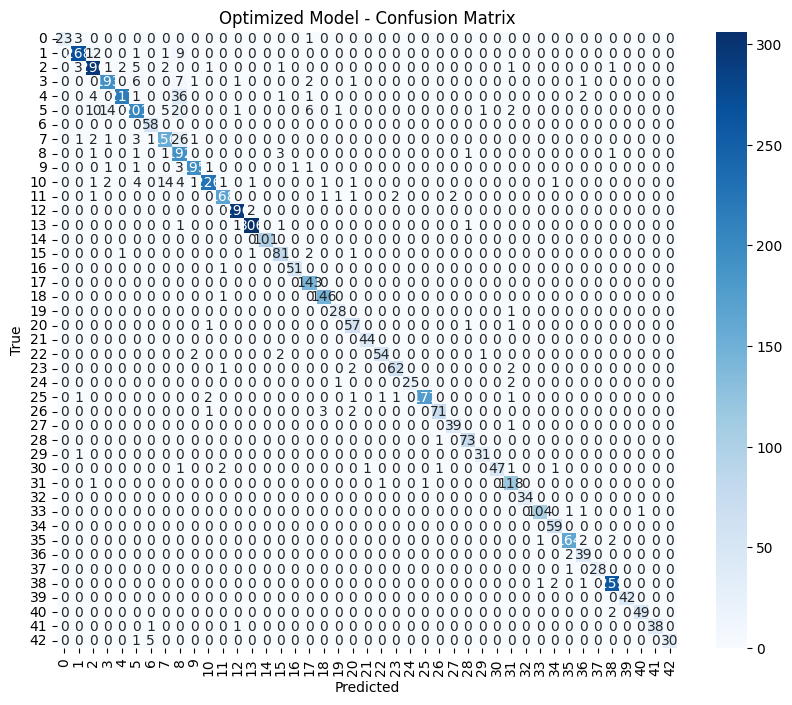


**Optimized MLP Model**

ACC: 0.94



**Optimized Model Confusion Matrix**



Hiperparametreler’de hidden layer, activation fonksiyonu, learning\_rate gibi hiperparametreler ile optimizasyon yapılmış en fazla sonucu veren değerler ile model kurulmuştur. Overfitting olduğu için optimizasyon ile acc düşmüş eskiye göre daha tutarlı sonuç vermiştir.

1. **RBF Modeli**:
   * sklearn kütüphanesinde doğrudan RBF modeli bulunmadığından, KEras kütüphanesi kullanın
   * Alternatif olarak, rbfnetwork gibi özel kütüphaneler veya Python’da yazılmış RBF fonksiyonları kullanılabilir.

import torch

from torch import nn

import torch.optim as optim

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

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import GridSearchCV

from sklearn.preprocessing import StandardScaler

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

import numpy as np

from torch.utils.data import DataLoader, TensorDataset

from keras import layers, models

# Veri setine göre sınıf sayısını otomatik olarak alalım

n\_classes = len(np.unique(y\_train))  # y\_train, eğitim etiketleri

# Görüntü verisinin boyutunu 2D'ye dönüştürelim

X\_train\_scaled = X\_train\_scaled.reshape(X\_train\_scaled.shape[0], -1)  # [batch\_size, width\*height\*channels]

X\_val\_scaled = X\_val\_scaled.reshape(X\_val\_scaled.shape[0], -1)  # [batch\_size, width\*height\*channels]

# Modeli tanımlama

model = models.Sequential()

# Girdi katmanını ve RBF katmanını ekleyelim

model.add(layers.InputLayer(input\_shape=(X\_train\_scaled.shape[1],)))

# RBF katmanı (gizli katman)

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(32, activation='relu'))

model.add(layers.Dense(16, activation='relu'))

# Çıkış katmanını sınıf sayısına göre otomatik belirleyelim

model.add(layers.Dense(n\_classes, activation='softmax'))  # Sınıf sayısına göre çıkış

# Modeli derleyelim

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

# Modeli eğitme

history = model.fit(X\_train\_scaled, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_val\_scaled, y\_val))

# Eğitim ve doğrulama performansını grafikte gösterelim

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

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

plt.xlabel('Epochs')

plt.ylabel('Doğruluk')

plt.legend()

plt.title('Model Doğruluğu')

plt.show()

# Modelin test verisi üzerindeki sonuçlarını değerlendirelim

y\_pred = model.predict(X\_val\_scaled)

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

# Performans metriklerini hesaplayalım

acc = accuracy\_score(y\_val, y\_pred)

f1 = f1\_score(y\_val, y\_pred, average='weighted')

prec = precision\_score(y\_val, y\_pred, average='weighted')

sens = recall\_score(y\_val, y\_pred, average='weighted')

# Sonuçları yazdıralım

print("Accuracy: ", acc)

print("F1 Score: ", f1)

print("Precision: ", prec)

print("Sensitivity (Recall): ", sens)

# Confusion Matrix çizelim

conf\_matrix = confusion\_matrix(y\_val, y\_pred)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y\_val), yticklabels=np.unique(y\_val))

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

# GridSearchCV ile hiperparametre optimizasyonu

from sklearn.model\_selection import GridSearchCV

from sklearn.neural\_network import MLPClassifier

# Modeli GridSearch ile optimize edelim

param\_grid = {

    'hidden\_layer\_sizes': [(64,), (128,), (64, 32)],

    'activation': ['relu', 'tanh'],

    'solver': ['adam', 'sgd'],

    'alpha': [0.0001, 0.001]

}

# MLPClassifier kullanarak hiperparametre optimizasyonu

grid\_search = GridSearchCV(MLPClassifier(max\_iter=1000), param\_grid, cv=3, scoring='accuracy', verbose=2)

grid\_search.fit(X\_train\_scaled, y\_train)

# En iyi parametreleri ve sonuçları yazdıralım

print("Best parameters found: ", grid\_search.best\_params\_)

best\_model = grid\_search.best\_estimator\_

# En iyi model ile test sonucu

y\_pred\_best = best\_model.predict(X\_val\_scaled)

acc\_best = accuracy\_score(y\_val, y\_pred\_best)

f1\_best = f1\_score(y\_val, y\_pred\_best, average='weighted')

prec\_best = precision\_score(y\_val, y\_pred\_best, average='weighted')

sens\_best = recall\_score(y\_val, y\_pred\_best, average='weighted')

# Sonuçları yazdıralım

print("Best Accuracy: ", acc\_best)

print("Best F1 Score: ", f1\_best)

print("Best Precision: ", prec\_best)

print("Best Sensitivity (Recall): ", sens\_best)

# Confusion matrix for best model

conf\_matrix\_best = confusion\_matrix(y\_val, y\_pred\_best)

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix\_best, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y\_val), yticklabels=np.unique(y\_val))

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix for Best Model')

plt.show()

# Modeli kaydedelim

model.save('traffic\_sign\_model.h5')

**RBF Base Model**

Accuracy: 0.9069069069069069

F1 Score: 0.9070136159845725

Precision: 0.9119140625448604

Sensitivity (Recall): 0.9069069069069069

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

Açıklama otomatik olarak oluşturuldu

**Base Model Confusion Matrix**

metin, ekran görüntüsü, çizgi, dikiş içeren bir resim

Açıklama otomatik olarak oluşturuldu

**RBF Optimized Model**

Best parameters found: {'activation': 'tanh', 'alpha': 0.0001, 'hidden\_layer\_sizes': (128,), 'solver': 'sgd'}

Best Accuracy: 0.973536036036036

Best F1 Score: 0.9734836053536855

Best Precision: 0.9738675952652837

Best Sensitivity (Recall): 0.973536036036036

**Optimized Confusion Matrix**

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

Açıklama otomatik olarak oluşturuldu

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Metrics** | **MLP Base Model** | **MLP Optimized Model** | **RBF Base Model** | **RBF Optimized Model** |
| **Accuracy** | 0.97 | 0.94 | 0.90 | 0.97 |
| **Precision** | 0.97 | 0.95 | 0.91 | 0.97 |
| **Recall** | 0.96 | 0.94 | 0.90 | 0.97 |
| **F1-Score** | 0.97 | 0.94 | 0.90 | 0.97 |

**Yorum**

**RBF Base Model**, yalnızca %90 doğruluk sağlamış ve diğer modellere kıyasla en düşük performansı göstermiştir. **MLP Base Model** ve **RBF Optimized Model** yine en yüksek duyarlılığı (%96 ve %97) sağlamıştır.

**Optimizasyon**, MLP modelinin performansını düşürmüş. Özellikle Accuracy (%97'den %94'e) ve Recall gibi metriklerde azalma gözlemlenmiştir. Bunun nedenini, optimizasyon sırasında yanlış hiperparametre seçimleri veya overfitting/underfitting gibi problemlerle açıklayabiliriz.

Optimizasyon, RBF modelinde performansı ciddi şekilde artırmıştır. **Accuracy, Precision, Recall ve F1-Score** değerleri %90'dan %97'ye çıkmıştır. Bu, doğru hiperparametre seçiminin performans üzerinde ne kadar etkili olduğunu gösterir.

Eğer optimizasyon sonrası elde edilen başarı önemliyse, **RBF Optimized Model** tercih edilir. Ancak modelin optimizasyon gerektirmeden yüksek performans göstermesi isteniyorsa, MLP Base Model daha uygun olabilir.

Optimizasyonun her zaman performansı artırmayacağı (örneğin MLP için) dikkate alınmalıdır.

En düşük performansa sahip model ise RBF Base Model olarak görülmektedir.

MLP Optimizasyonlu Modeli, optimize edilmesine rağmen temel haline kıyasla bazı metriklerde düşüş yaşamıştır; bu durum, aşırı öğrenme (overfitting) ya da yanlış hiperparametre seçimleri gibi sorunların bir işareti olabilir. RBF Optimizasyonlu Modeli ise, optimize edildiğinde tüm metriklerde tutarlı bir şekilde iyileşme göstermiştir, bu da modelin uygun bir şekilde genel performansını artırdığını ortaya koymaktadır. Genel olarak, model seçimi ve optimizasyon süreçleri performans üzerinde büyük bir etkiye sahiptir, ancak yanlış optimize edilmiş modeller performansı düşürebilir ve modelin genel genelleme kabiliyetini sınırlandırabilir. Bu nedenle, model geliştirme sürecinde denge, dikkatli hiperparametre seçimi ve aşırı öğrenmeden kaçınma kritik önem taşır.

**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