Makine Öğrenmesi Kodları

```
# --- Gerekli ek kütüphaneyi yükleme ---
!pip install mlxtend

# --- Kütüphaneleri Yükleme ---
import pandas as pd
import numpy as np
import os
import time
import pickle
import warnings

# Veri Hazırlama
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Modeller
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier

# Metrikler ve Değerlendirme
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_curve, auc, f1_score, precision_score,
recall_score
from mlxtend.evaluate import mcnemar_table, mcnemar

# Görselleştirme
import matplotlib.pyplot as plt
import seaborn as sns

warnings.filterwarnings('ignore')
print("Tüm kütüphaneler başarıyla yüklendi.")
print("="*80)
```

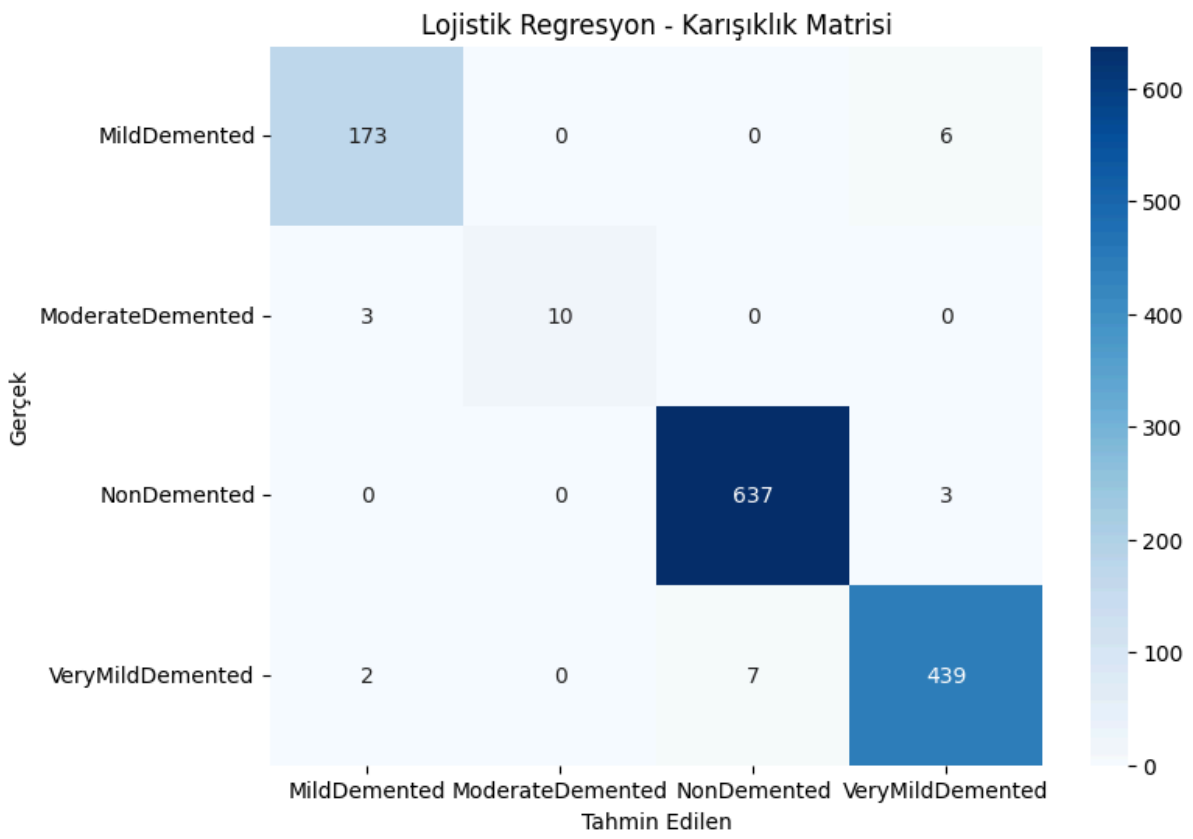
```
# --- Veri Seti Yükleme, Tanımlama ve Bölme Stratejisi ---
data_dir = '/content/Alzheimer_s Dataset/train'
filepaths, labels = [], []
for label in os.listdir(data_dir):
    class_path = os.path.join(data_dir, label)
    if os.path.isdir(class_path):
        for file in os.listdir(class_path):
            filepaths.append(os.path.join(class_path, file))
            labels.append(label)

image_df = pd.DataFrame({'filepath': filepaths, 'label': labels})

def generate_synthetic_data(label):
    if label == 'NonDemented':
        age, mmse, educ = np.random.randint(60, 75),
np.random.uniform(27, 30), np.random.randint(12, 20)
    elif label == 'VeryMildDemented':
        age, mmse, educ = np.random.randint(65, 80),
np.random.uniform(24, 27), np.random.randint(10, 18)
    elif label == 'MildDemented':
        age, mmse, educ = np.random.randint(70, 85),
np.random.uniform(20, 24), np.random.randint(8, 16)
    else:
        age, mmse, educ = np.random.randint(75, 90),
np.random.uniform(10, 20), np.random.randint(6, 14)
    gender = np.random.choice(['M', 'F'])
    return age, round(mmse, 2), educ, gender

synthetic_data = image_df['label'].apply(generate_synthetic_data)
synthetic_df = pd.DataFrame(synthetic_data.tolist(), columns=['Age',
'MMSE', 'EDUC', 'Gender'])
final_df = pd.concat([image_df, synthetic_df], axis=1)

print("1.1 Veri Seti Tanımı ve Bölme Stratejisi")
print(f"Toplam Örnek Sayısı: {len(final_df)}")
train_val_df, test_df = train_test_split(final_df, test_size=0.20,
random_state=42, stratify=final_df['label'])
print(f"Eğitim/Doğrulama Seti: {len(train_val_df)}, Kilitli Nihai Test
Seti: {len(test_df)}")
print("="*80)


# --- Veri Ön İşleme ---
```

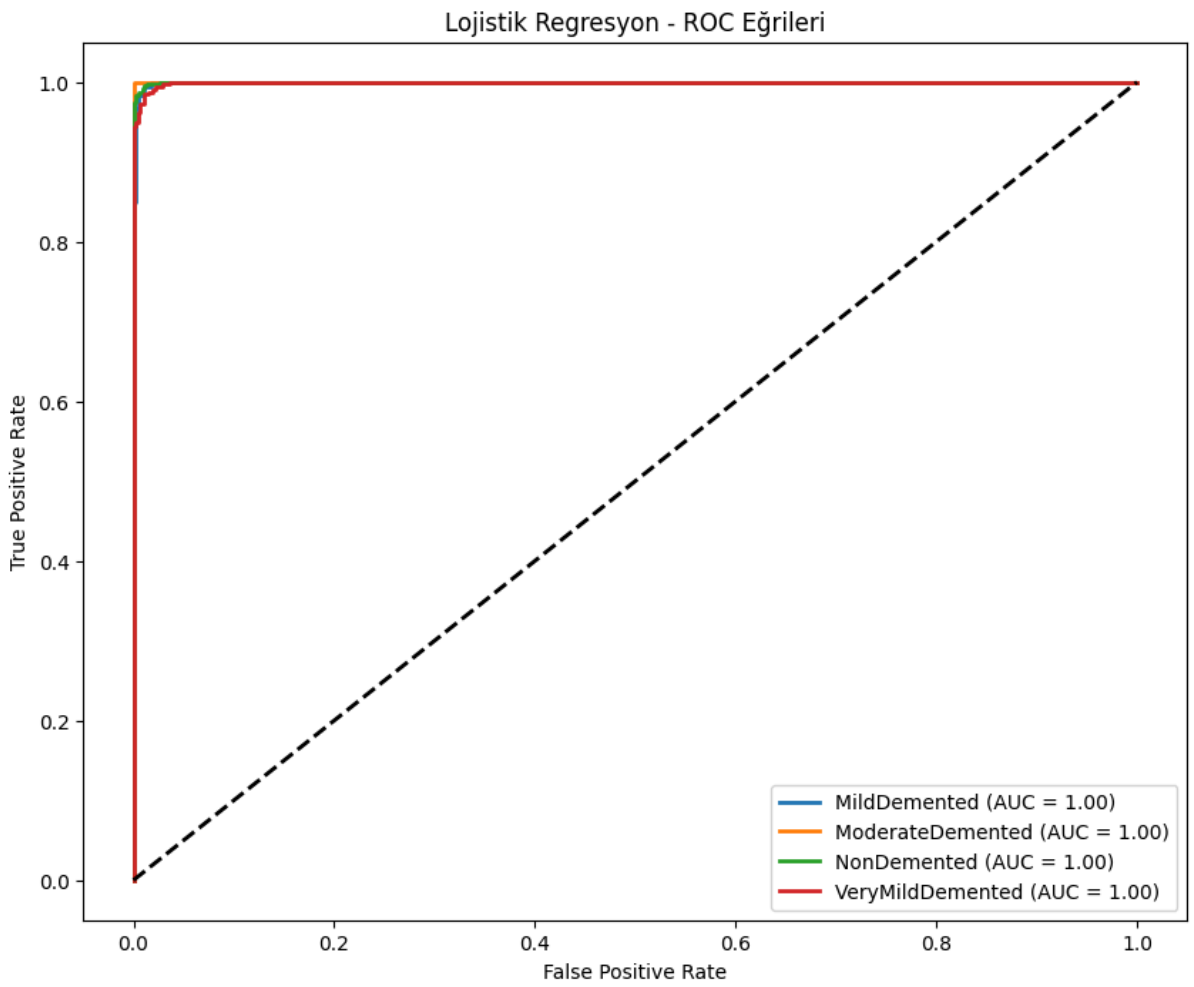
```
X = train_val_df[['Age', 'MMSE', 'EDUC', 'Gender']]
y = train_val_df['label']
X_final_test = test_df[['Age', 'MMSE', 'EDUC', 'Gender']]
y_final_test = test_df['label']

X = pd.get_dummies(X, columns=['Gender'], drop_first=True)
X_final_test = pd.get_dummies(X_final_test, columns=['Gender'],
drop_first=True)

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_final_test_encoded = label_encoder.transform(y_final_test)
class_names = label_encoder.classes_

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_final_test_scaled = scaler.transform(X_final_test)
print("1.3 Ön İşleme Tamamlandı.")
print("="*80)


# --- Modelleri Tanımlama ---
models = {
    "Lojistik Regresyon": LogisticRegression(random_state=42,
max_iter=1000),
    "K-En Yakın Komşu": KNeighborsClassifier(),
    "Destek Vektör Mak.": SVC(probability=True, random_state=42),
    "Rastgele Orman": RandomForestClassifier(random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42)
}

# --- K-Fold Çapraz Doğrulama Raporu ---
print("1.6.4 K-Fold Çapraz Doğrulama Raporu (k=5)")
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_results = []
for model_name, model in models.items():
    f1_scores_cv = [f1_score(y_encoded[val_index],
model.fit(X_scaled[train_index],
y_encoded[train_index]).predict(X_scaled[val_index]),
average='weighted') for train_index, val_index in kfold.split(X_scaled,
y_encoded)]
    cv_results.append({
        "Model": model_name,
```

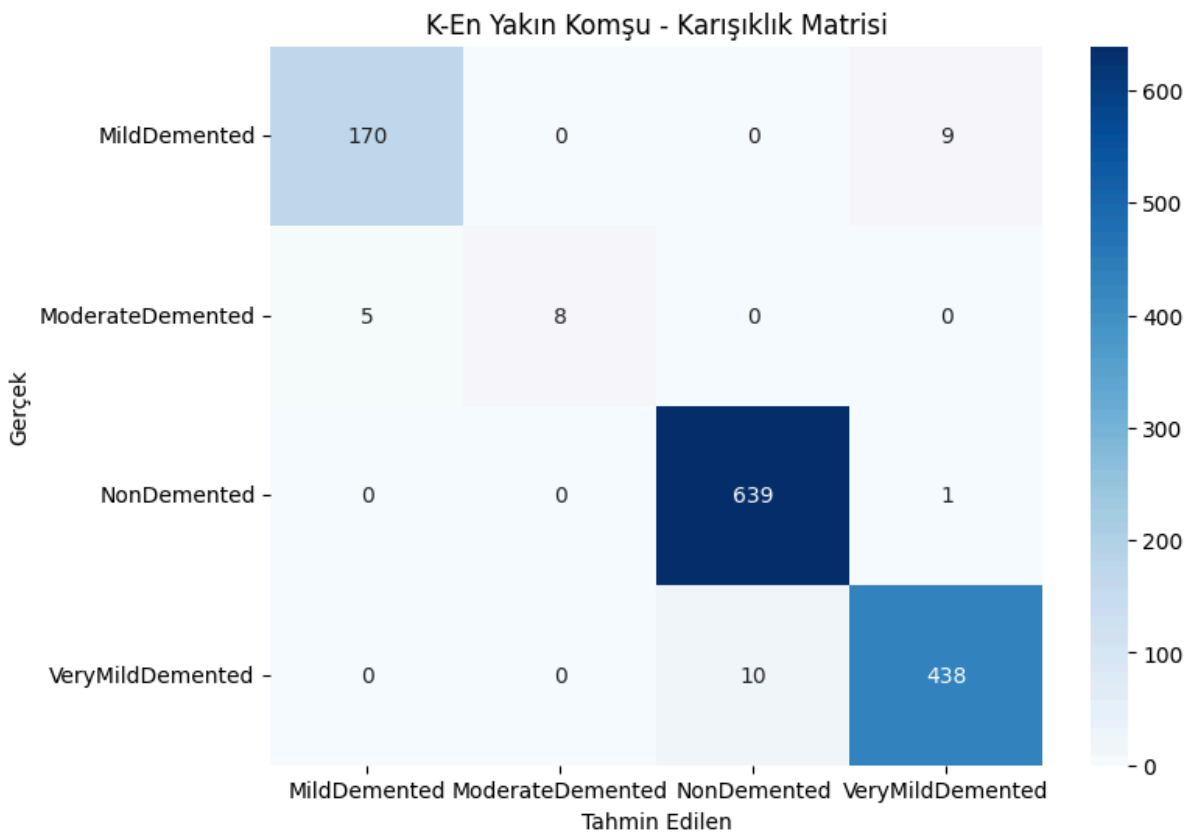
```
        "Ortalama F1-Skoru": np.mean(f1_scores_cv),
        "F1 Std. Sapma": np.std(f1_scores_cv)
    })
cv_results_df = pd.DataFrame(cv_results)
print("\n--- Çapraz Doğrulama Sonuç Özeti ---")
print(cv_results_df)
print("="*80)


# --- Nihai Test Seti Üzerinde Her Model İçin ---
print("1.5 & 1.6: Karşılaştırmalı ve Detaylı Sınıflandırma Analizleri
(Nihai Test Seti)")

final_results_summary = []
all_preds = {}

for model_name, model in models.items():
    print(f"\n{'=' * 30} {model_name.upper()} {'=' * 30}")

    model.fit(X_scaled, y_encoded)
    y_final_pred = model.predict(X_final_test_scaled)
    y_final_prob = model.predict_proba(X_final_test_scaled)
    all_preds[model_name] = y_final_pred

    print(f"\n--- {model_name} | Detaylı Sınıflandırma Raporu ---")
    report = classification_report(y_final_test_encoded, y_final_pred,
target_names=class_names, output_dict=True)
    print(pd.DataFrame(report).transpose())

    plt.figure(figsize=(8, 6))
    cm = confusion_matrix(y_final_test_encoded, y_final_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
    plt.title(f'{model_name} - Karışıklık Matrisi')
    plt.xlabel('Tahmin Edilen')
    plt.ylabel('Gerçek')
    plt.show()

    plt.figure(figsize=(10, 8))
    for i, class_name in enumerate(class_names):
        fpr, tpr, _ = roc_curve(y_final_test_encoded, y_final_prob[:,
i], pos_label=i)
        roc_auc = auc(fpr, tpr)
```

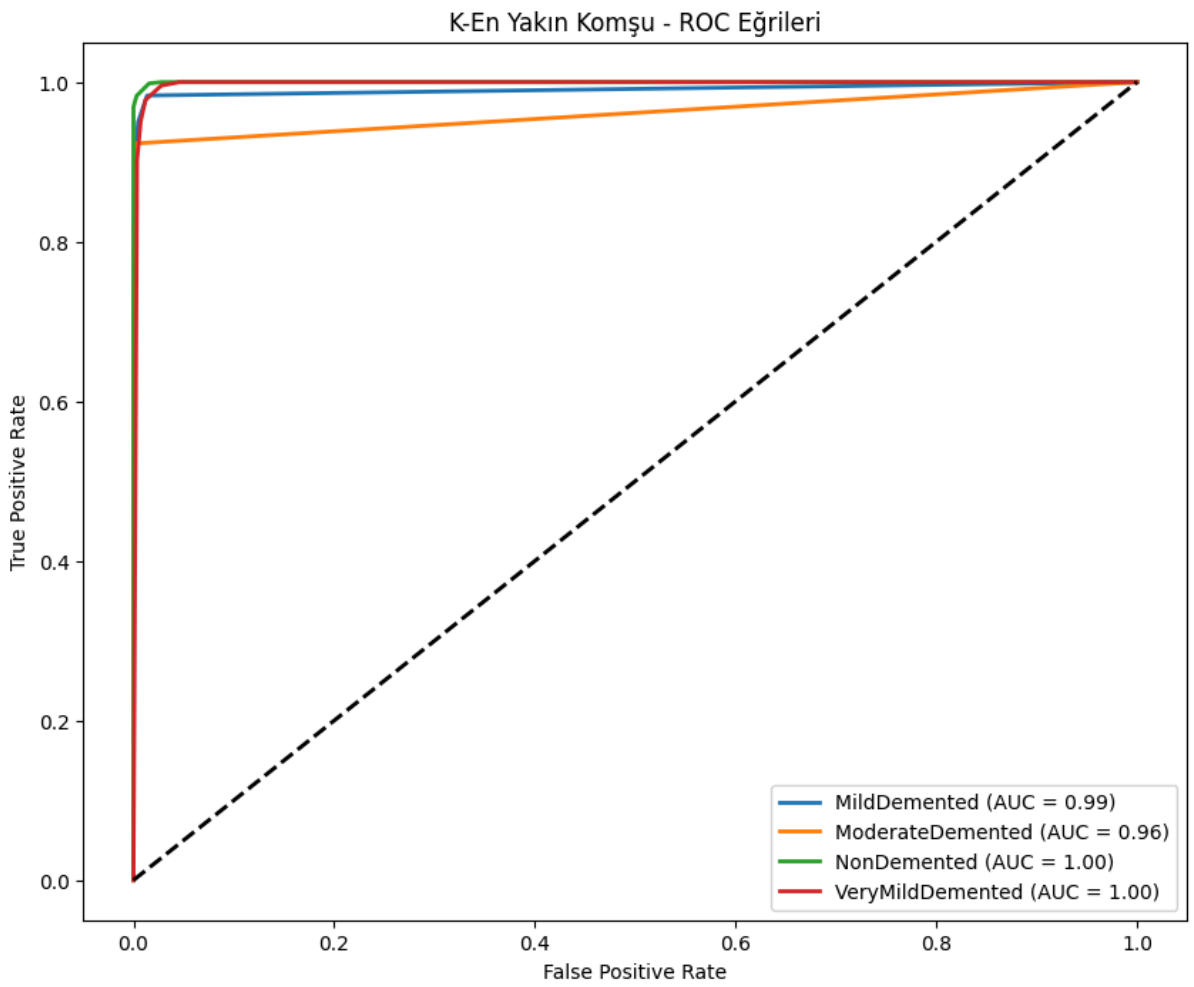
```
        plt.plot(fpr, tpr, lw=2, label=f'{class_name} (AUC =
{roc_auc:.2f})')

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'{model_name} - ROC Eğrileri')
    plt.legend(loc="lower right")
    plt.show()

    final_results_summary.append({
        "Model": model_name,
        "Doğruluk (ACC)": accuracy_score(y_final_test_encoded,
y_final_pred),
        "Hassasiyet (Makro)": precision_score(y_final_test_encoded,
y_final_pred, average='macro', zero_division=0),
        "Duyarlılık (Makro)": recall_score(y_final_test_encoded,
y_final_pred, average='macro', zero_division=0),
        "F1-Skoru (Makro)": f1_score(y_final_test_encoded,
y_final_pred, average='macro', zero_division=0),
        "F1-Skoru (Ağırlıklı)": f1_score(y_final_test_encoded,
y_final_pred, average='weighted', zero_division=0)
    })


# --- MODELLERİN SONUÇLARINI ÖZETLEME VE KARŞILAŞTIRMA ---
print("\n" + "="*80)
print("--- 1.5.1 KAPSAMLI PERFORMANS KARŞILAŞTIRMA ÖZET TABLOSU (NİHAİ
TEST SETİ) ---")

final_results_df = pd.DataFrame(final_results_summary)
# Ondalıkları daha okunaklı yapmak için formatlayalım
pd.options.display.float_format = '{:.4f}'.format
print(final_results_df)

# Görsel Sunum (Çubuk Grafik)
final_results_df.set_index('Model').plot(kind='bar', figsize=(12, 7),
rot=45)
plt.title('Tüm Modellerin Nihai Test Seti Performansı')
plt.ylabel('Skor')
plt.grid(axis='y', linestyle='--')
plt.legend(title='Metrikler', bbox_to_anchor=(1.05, 1), loc='upper
left')
```

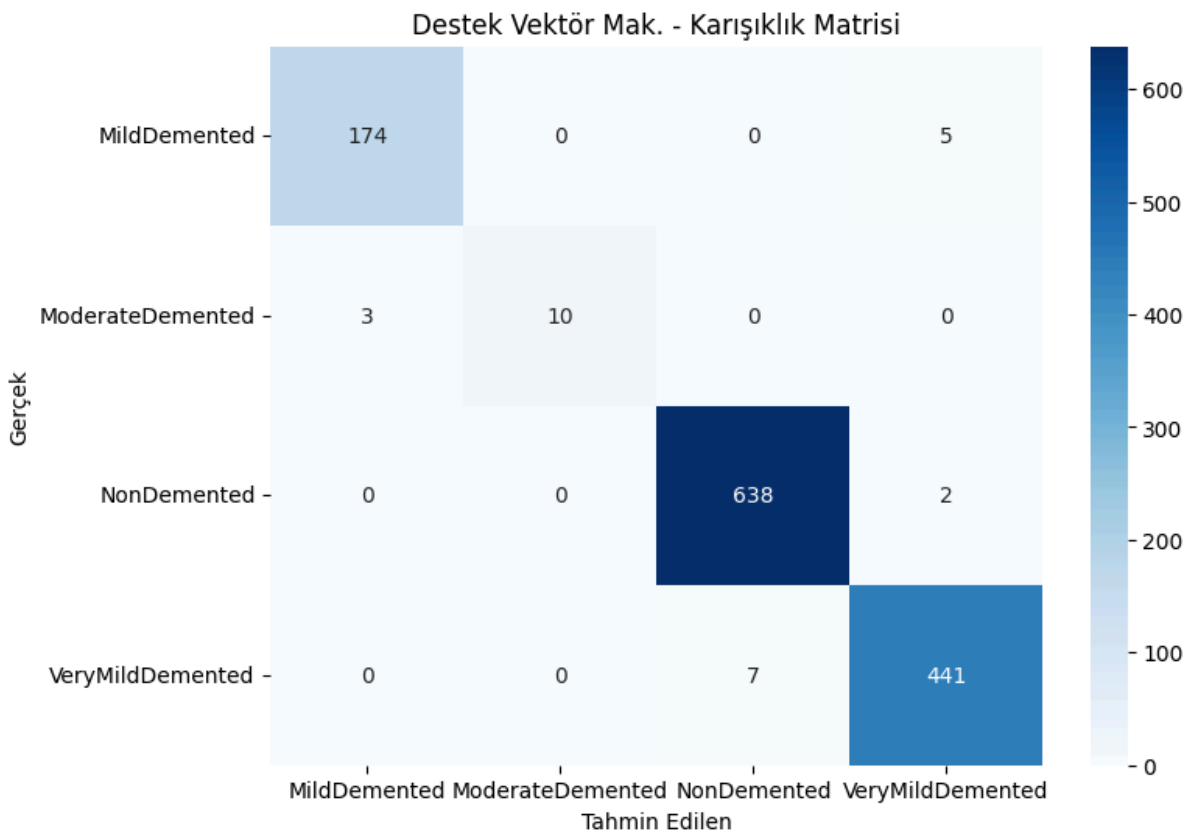
```
plt.tight_layout()
plt.show()


# İstatistiksel Anlamlılık Testi
print("\n--- 1.5.2 İstatistiksel Anlamlılık Testi ---")
sorted_results = final_results_df.sort_values(by='F1-Skoru
(Ağırlıklı)', ascending=False)
if len(sorted_results) > 1:
    best_model_name = sorted_results.iloc[0]['Model']
    second_best_model_name = sorted_results.iloc[1]['Model']

    best_model_preds = all_preds[best_model_name]
    second_best_model_preds = all_preds[second_best_model_name]

    tb = mcnemar_table(y_target=y_final_test_encoded,
y_model1=best_model_preds, y_model2=second_best_model_preds)

    print(f"En İyi Model ({best_model_name}) ile İkinci En İyi Model
({second_best_model_name}) karşılaştırılıyor.")
    chi2, p_value = mcnemar(ary=tb, corrected=True)

    print(f"\nKi-kare istatistiği: {chi2:.4f}, p-değeri:
{p_value:.6f}")
    if p_value < 0.05:
        print("Sonuç: İki modelin performansı arasındaki fark
istatistiksel olarak ANLAMLIDIR.")
    else:
        print("Sonuç: İki modelin performansı arasındaki fark
istatistiksel olarak ANLAMLI DEĞİLDİR.")
else:
    print("Karşılaştırma için yeterli model sonucu bulunamadı.")


print("\n\n#################################################################
####################")
print("# MAKİNE ÖĞRENMESİ AŞAMASI (MAKSİMUM DETAY) TAMAMLANDI #")
print("#################################################################
#################")
```

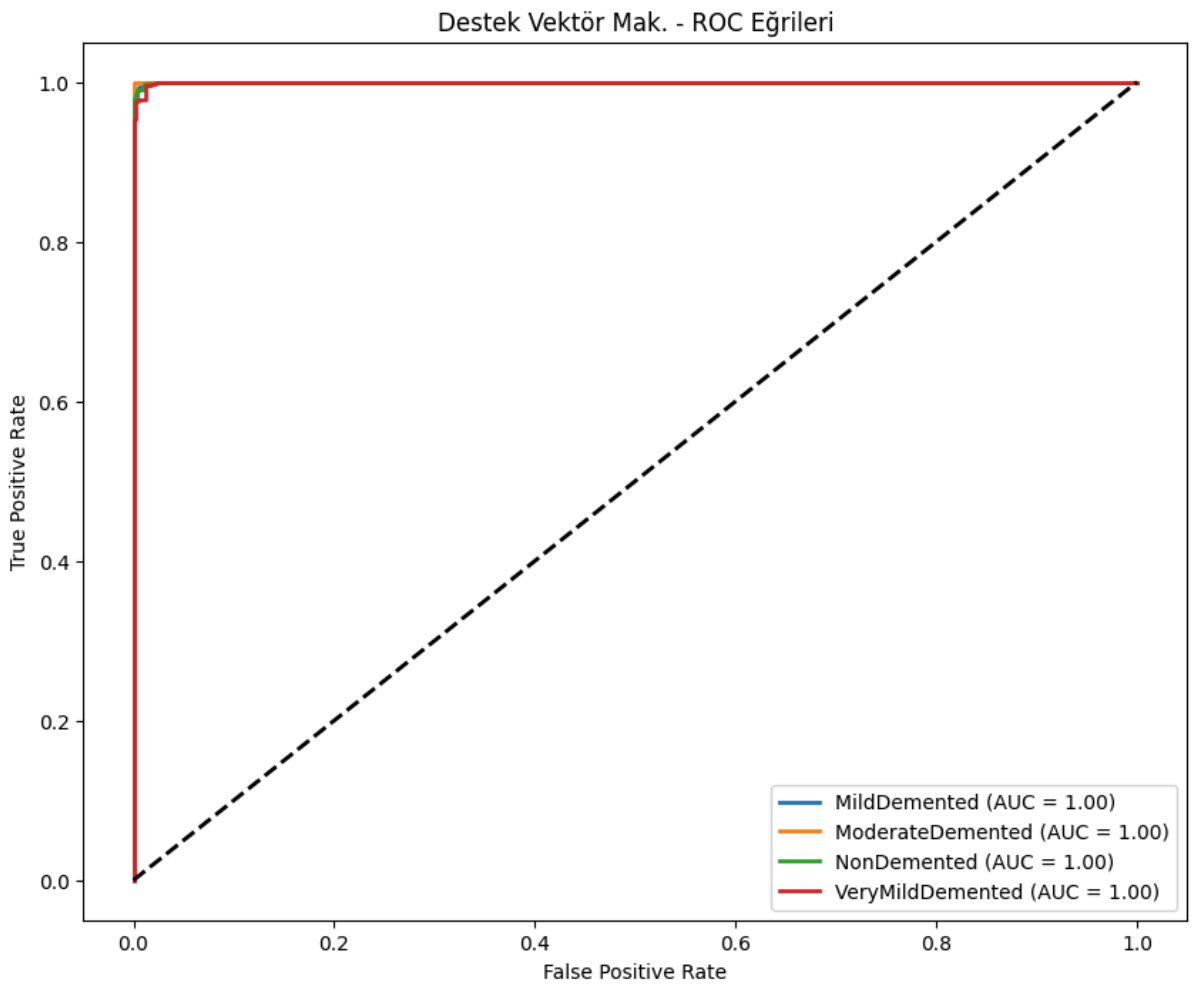

Lojistik Regresyon - Karışıklık Matrisi
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Gerçek
173
0
0
6
3
10
0
0
0
0
637
3
2
0
7
439
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Tahmin Edilen
600
500
400
300
200
100
0

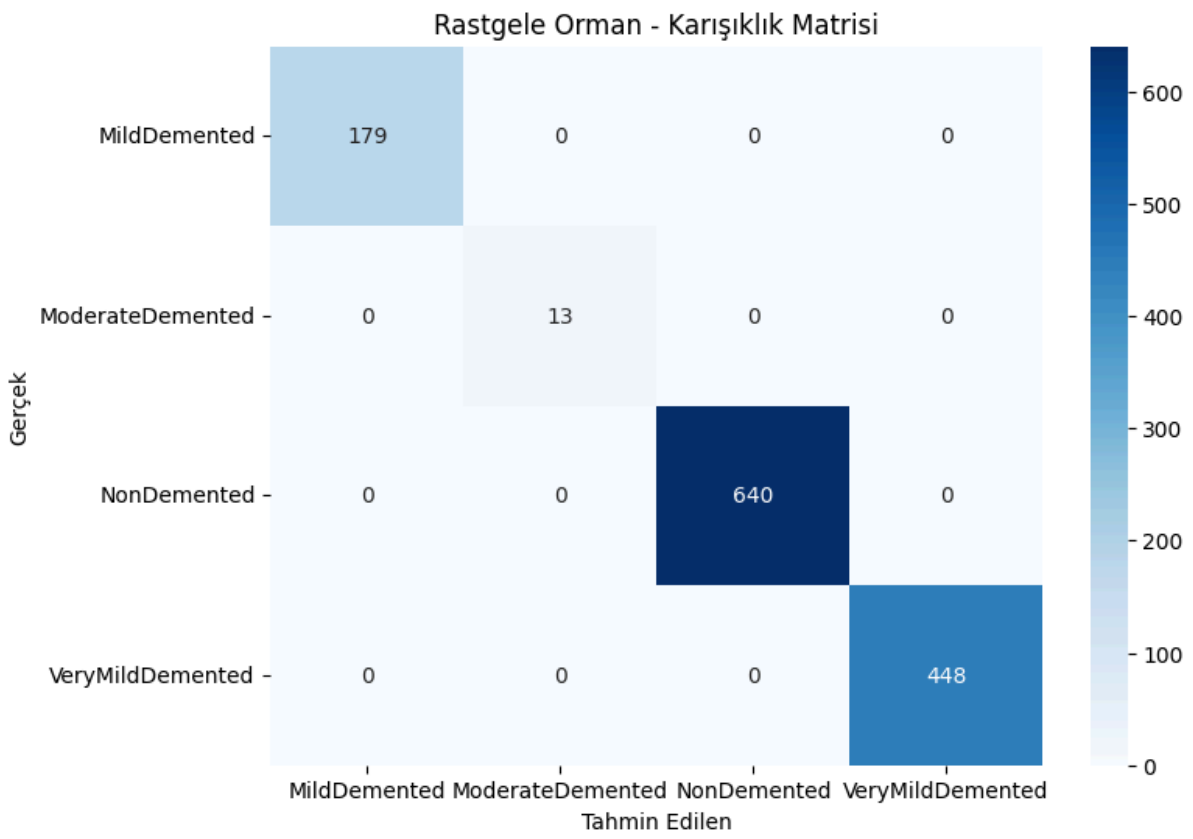

Lojistik Regresyon - ROC Eğrileri
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
MildDemented (AUC = 1.00)
ModerateDemented (AUC = 1.00)
NonDemented (AUC = 1.00)
VeryMildDemented (AUC = 1.00)

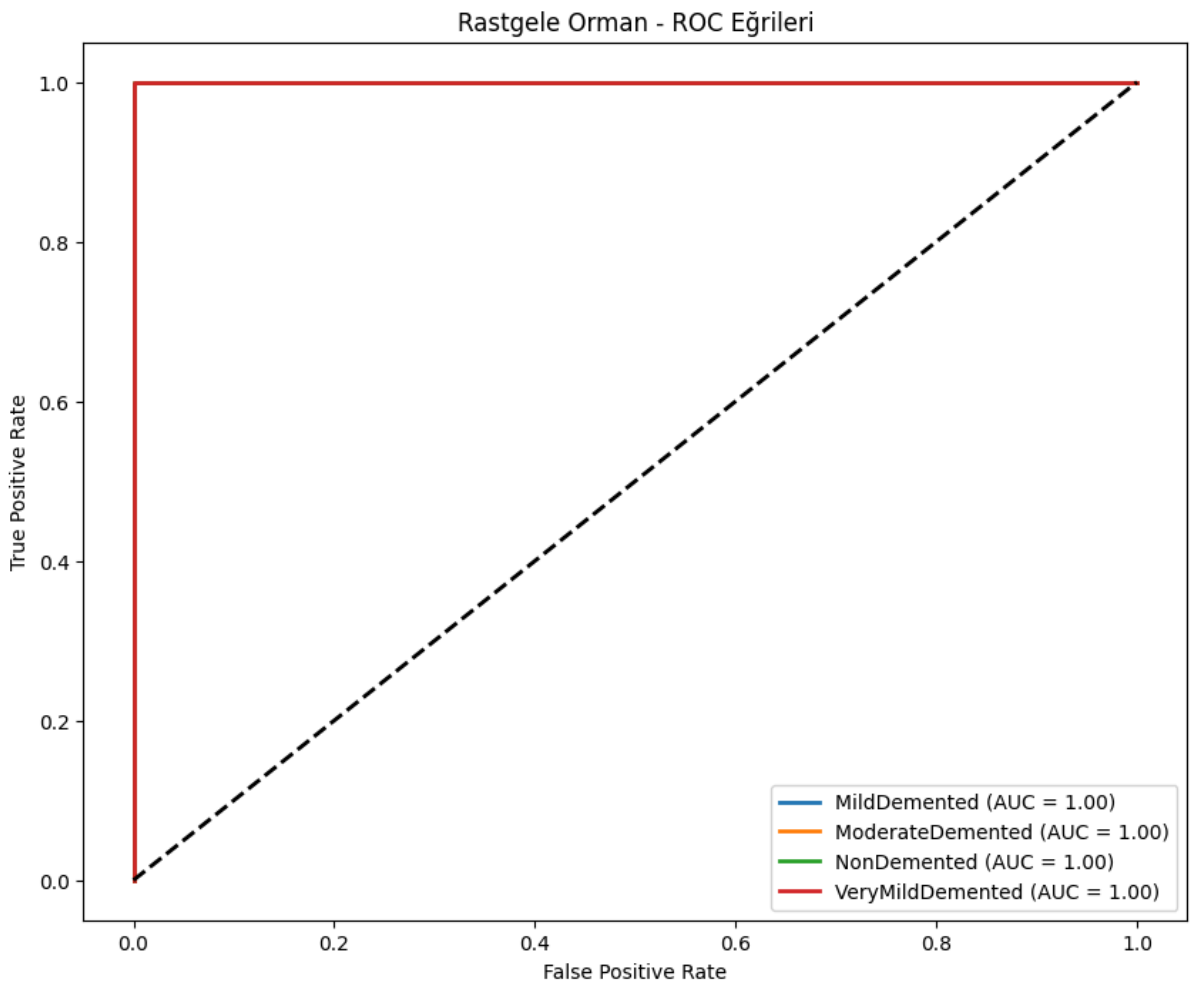

K-En Yakın Komşu - Karışıklık Matrisi
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Gerçek
170
0
0
9
5
8
0
0
0
0
639
1
0
0
10
438
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Tahmin Edilen
600
500
400
300
200
100
0

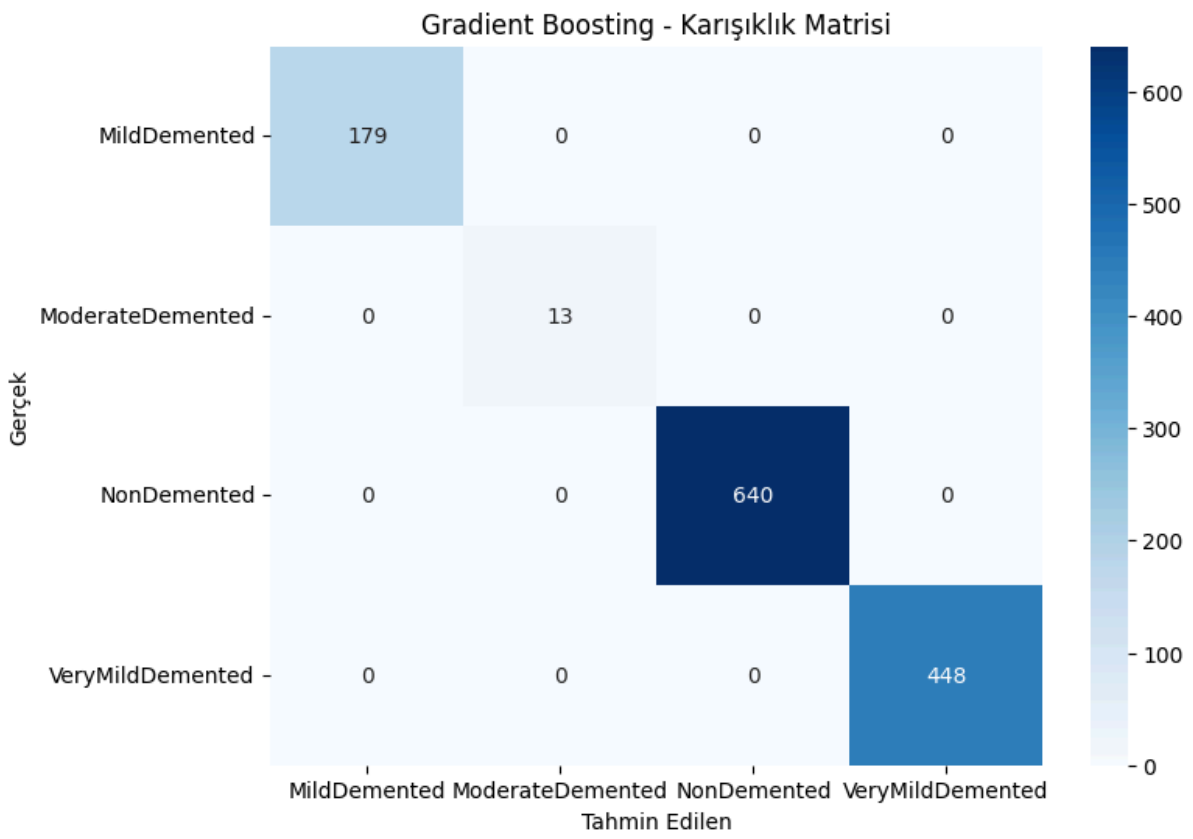

K-En Yakın Komşu - ROC Eğrileri
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
MildDemented (AUC = 0.99)
ModerateDemented (AUC = 0.96)
NonDemented (AUC = 1.00)
VeryMildDemented (AUC = 1.00)

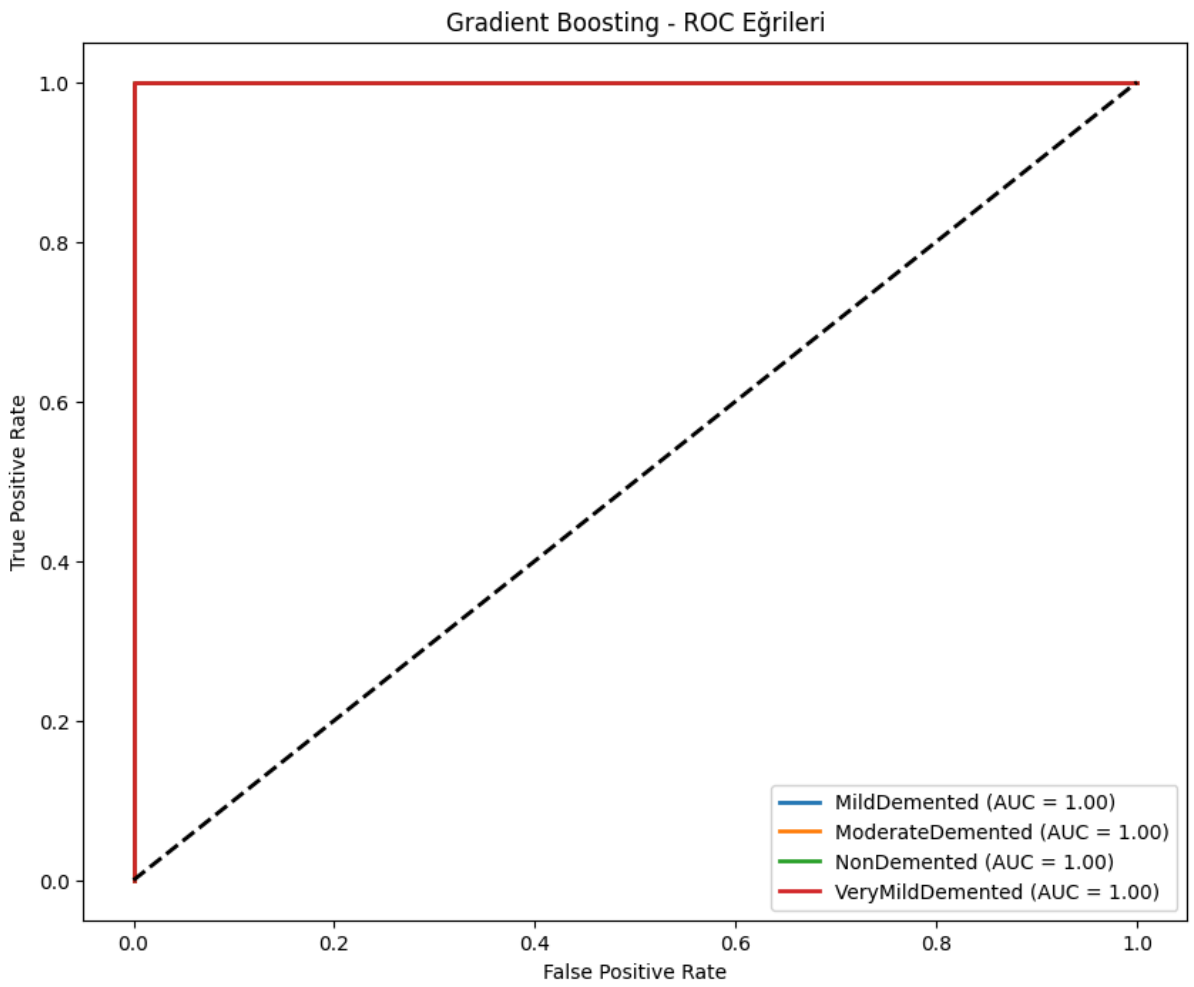

Destek Vektör Mak. - Karışıklık Matrisi
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Gerçek
174
0
0
5
3
10
0
0
0
0
638
2
0
0
7
441
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Tahmin Edilen
600
500
400
300
200
100
0

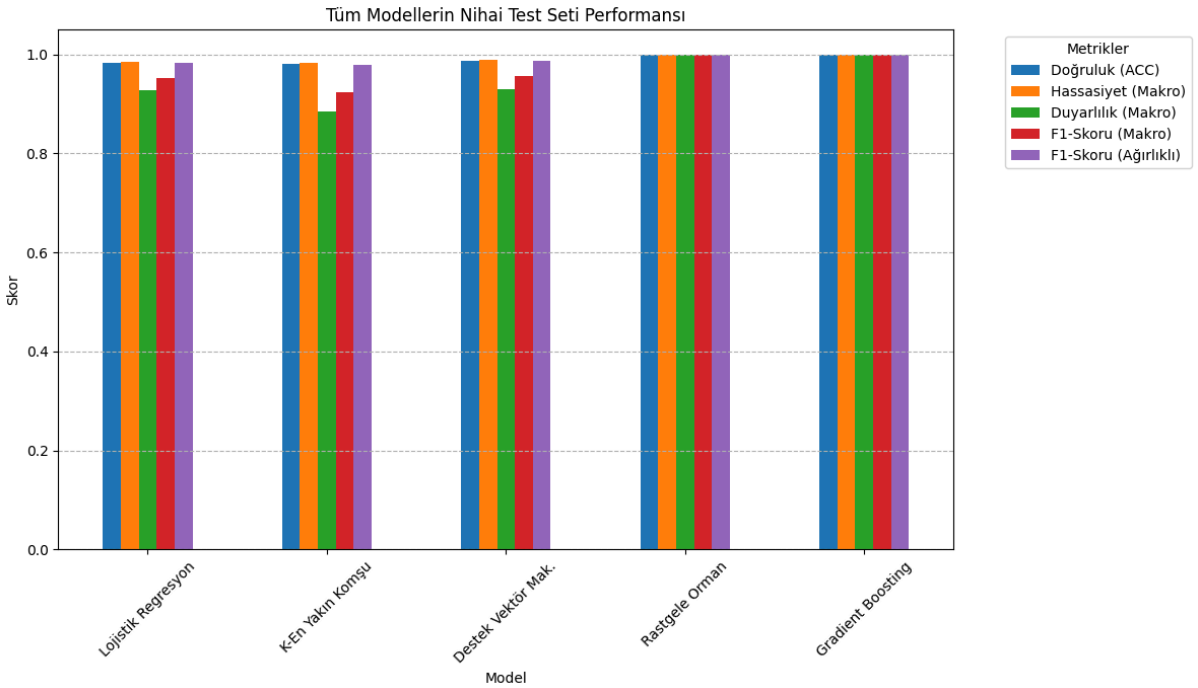

Destek Vektör Mak. - ROC Eğrileri
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
MildDemented (AUC = 1.00)
ModerateDemented (AUC = 1.00)
NonDemented (AUC = 1.00)
VeryMildDemented (AUC = 1.00)


Rastgele Orman - Karışıklık Matrisi
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Gerçek
179
0
0
0
0
13
0
0
0
0
640
0
0
0
0
448
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Tahmin Edilen
600
500
400
300
200
100
0


Rastgele Orman - ROC Eğrileri
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
MildDemented (AUC = 1.00)
ModerateDemented (AUC = 1.00)
NonDemented (AUC = 1.00)
VeryMildDemented (AUC = 1.00)


Gradient Boosting - Karışıklık Matrisi
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Gerçek
179
0
0
0
0
13
0
0
0
0
640
0
0
0
0
448
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Tahmin Edilen
600
500
400
300
200
100
0


Gradient Boosting - ROC Eğrileri
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
MildDemented (AUC = 1.00)
ModerateDemented (AUC = 1.00)
NonDemented (AUC = 1.00)
VeryMildDemented (AUC = 1.00)


Tüm Modellerin Nihai Test Seti Performansı
Skor
Model
Lojistik Regresyon
K-En Yakın Komşu
Destek Vektör Mak.
Rastgele Orman
Gradient Boosting
Metrikler
Doğruluk (ACC)
Hassasiyet (Makro)
Duyarlılık (Makro)
F1-Skoru (Makro)
F1-Skoru (Ağırlıklı)

Derin Öğrenme 3 model

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import (Input, Dense, Conv2D,
MaxPooling2D, Flatten, Dropout,
                                     BatchNormalization,
GlobalAveragePooling2D, Layer)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
# MLP ve ViT için ek katmanlar
from tensorflow.keras.layers import Reshape, LayerNormalization,
MultiHeadAttention, Add, Embedding

print(f"TensorFlow Sürümü: {tf.__version__}")
print("="*80)

# Parametreler
IMG_SIZE = (128, 128)
BATCH_SIZE = 32
N_CLASSES = len(class_names)


train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.15,
    height_shift_range=0.15,
    shear_range=0.15,
    zoom_range=0.15,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)


print("2.2 Derin Öğrenme Veri Üreteçleri (ImageDataGenerator)
tanımlandı.")
print("="*80)
```

```python
# --- Temel CNN ---
def build_baseline_cnn(input_shape, n_classes):
    model = Sequential([
        Input(shape=input_shape),
        Conv2D(32, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(n_classes, activation='softmax')
    ], name="Temel_CNN")
    return model

# --- Derin CNN (BN + Dropout) ---
def build_deep_cnn(input_shape, n_classes):
    model = Sequential([
        Input(shape=input_shape),
        Conv2D(32, (3, 3), padding='same'),
        BatchNormalization(),
        tf.keras.layers.Activation('relu'),
        MaxPooling2D((2, 2)),

        Conv2D(64, (3, 3), padding='same'),
        BatchNormalization(),
        tf.keras.layers.Activation('relu'),
        MaxPooling2D((2, 2)),

        Conv2D(128, (3, 3), padding='same'),
        BatchNormalization(),
        tf.keras.layers.Activation('relu'),
        MaxPooling2D((2, 2)),

        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(n_classes, activation='softmax')
    ], name="Derin_CNN_BN_Dropout")
    return model

# --- Çok Katmanlı Algılayıcı (MLP) ---
def build_mlp(input_shape, n_classes):
    model = Sequential([
```

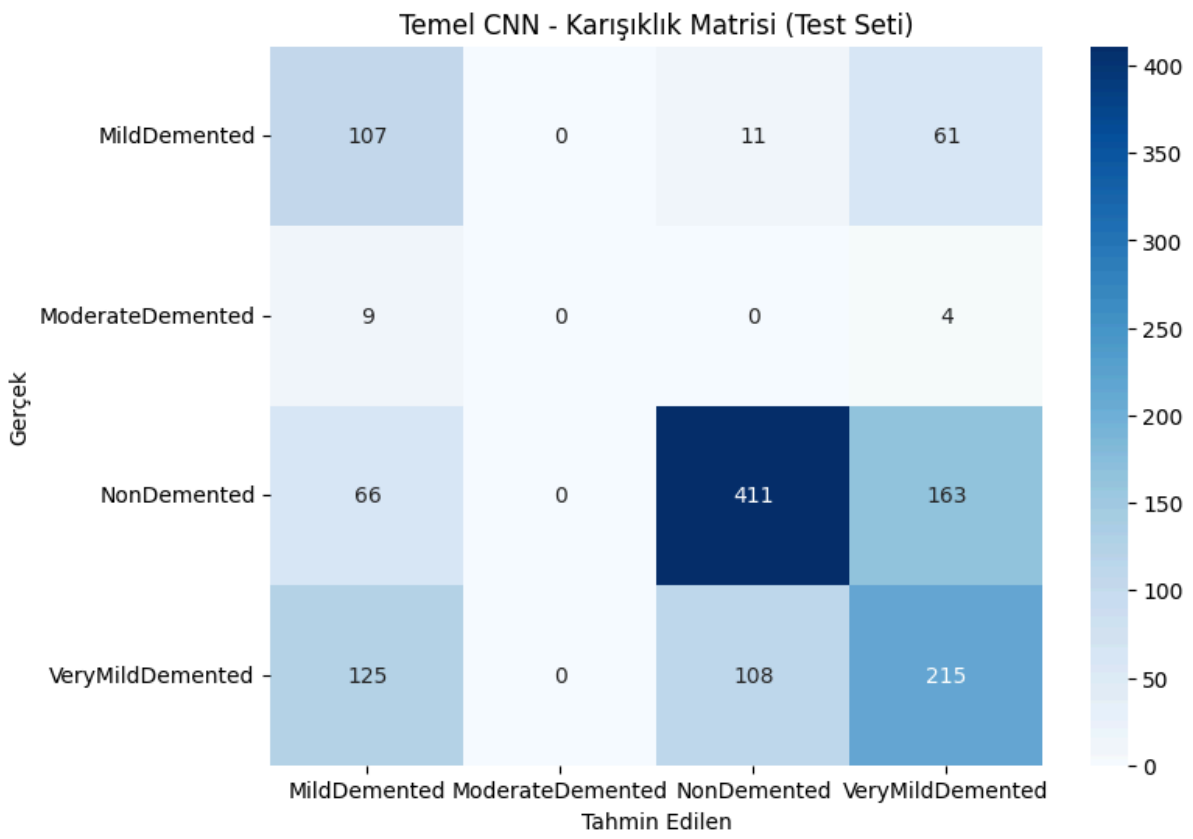
```
        Input(shape=input_shape),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.3),
        Dense(256, activation='relu'),
        Dropout(0.3),
        Dense(128, activation='relu'),
        Dense(n_classes, activation='softmax')
    ], name="MLP")
    return model


dl_models = {
    "Temel CNN": build_baseline_cnn,
    "Derin CNN (BN+Dropout)": build_deep_cnn,
    "MLP (Çok Katmanlı Algılayıcı)": build_mlp
}

print("2.3 Üç adet özgün Derin Öğrenme modeli tanımlandı.")
print(list(dl_models.keys()))
print("="*80)


kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
dl_cv_results = []
dl_final_results = {}

# Her model için döngü
for model_name, model_builder in dl_models.items():
    print(f"\n{'#' * 30}\n# Model: {model_name.upper()}\n{'#' * 30}")

    fold_no = 1
    fold_scores = []

    # Her fold için döngü
    for train_index, val_index in kfold.split(train_val_df,
train_val_df['label']):
        print(f"\n--- {model_name} | Fold
{fold_no}/{kfold.get_n_splits()} ---")

        # Fold verilerini ayır
        train_fold_df = train_val_df.iloc[train_index]
        val_fold_df = train_val_df.iloc[val_index]
```

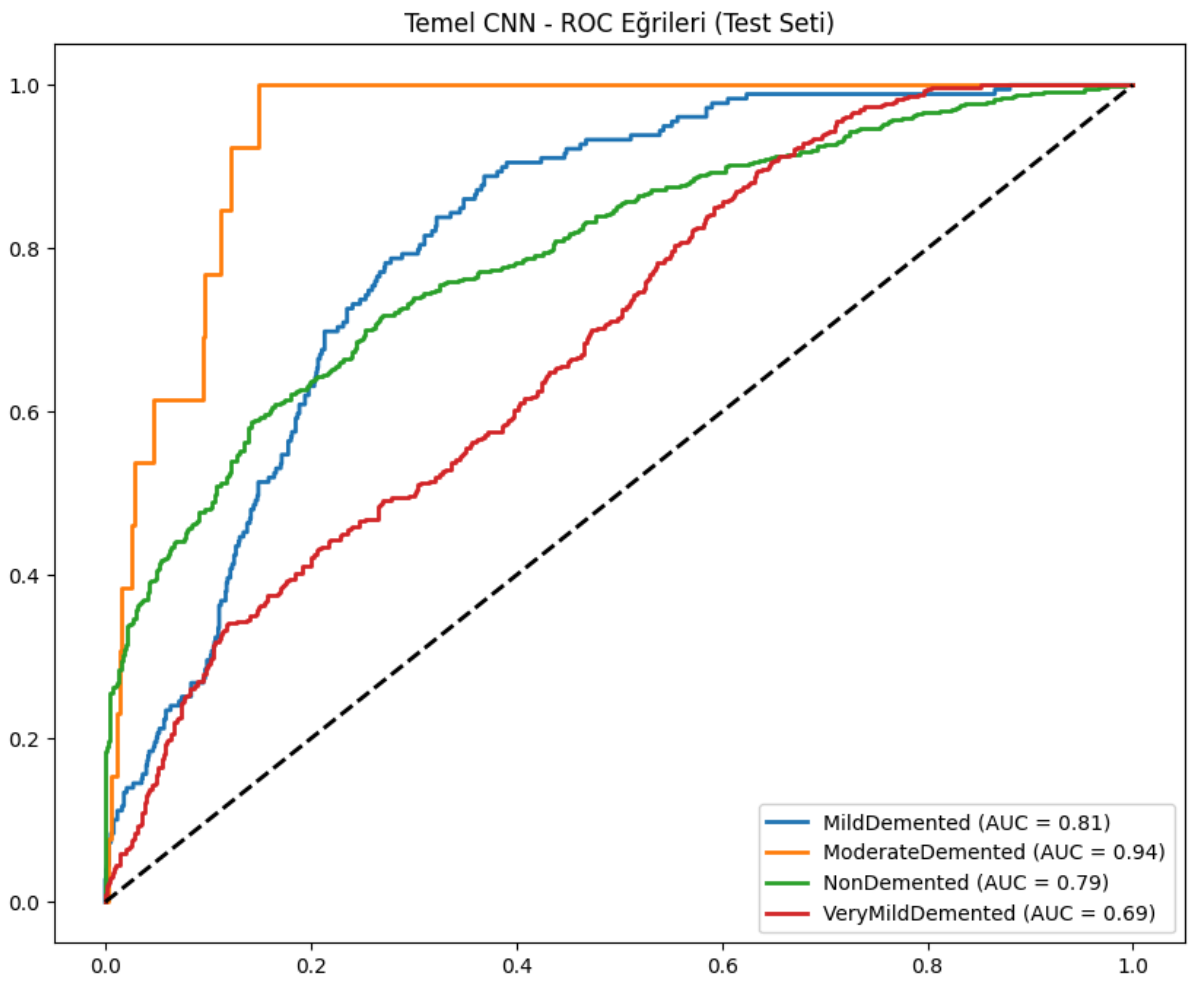
```
        # Veri üreteçlerini oluştur
        train_generator = train_datagen.flow_from_dataframe(
            dataframe=train_fold_df,
            x_col='filepath',
            y_col='label',
            target_size=IMG_SIZE,
            batch_size=BATCH_SIZE,
            class_mode='categorical',
            shuffle=True
        )
        validation_generator = test_datagen.flow_from_dataframe(
            dataframe=val_fold_df,
            x_col='filepath',
            y_col='label',
            target_size=IMG_SIZE,
            batch_size=BATCH_SIZE,
            class_mode='categorical',
            shuffle=False
        )

        # Modeli oluştur ve derle
        model = model_builder(input_shape=(IMG_SIZE[0], IMG_SIZE[1],
3), n_classes=N_CLASSES)
        model.compile(optimizer=Adam(learning_rate=0.001),
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

        # Erken durdurma ve öğrenme oranını düşürme callback'leri
        early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
        reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
patience=5, min_lr=1e-6)

        # Modeli eğit
        history = model.fit(
            train_generator,
            epochs=50,
            validation_data=validation_generator,
            callbacks=[early_stopping, reduce_lr],
            verbose=1
        )
```

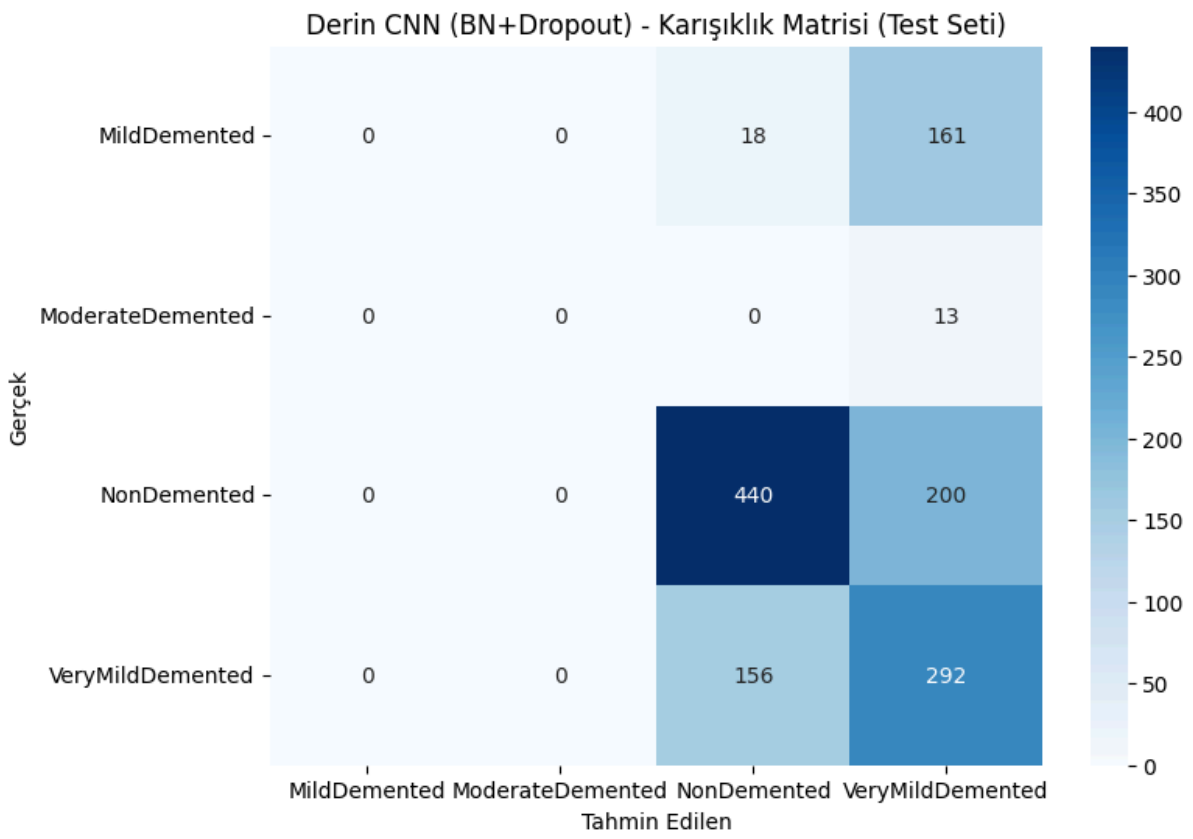
```
        # Fold performansını değerlendir
        loss, accuracy = model.evaluate(validation_generator)
        print(f"Fold {fold_no} Sonucu: Accuracy = {accuracy:.4f}")
        fold_scores.append(accuracy)
        fold_no += 1

    # Modelin K-Fold sonuçlarını kaydet
    dl_cv_results.append({
        "Model": model_name,
        "Ortalama Doğruluk": np.mean(fold_scores),
        "Doğruluk Std. Sapma": np.std(fold_scores)
    })

    # --- Nihai Test Seti Değerlendirmesi (Her DL Modeli İçin) ---
    print(f"\n--- {model_name} | Nihai Test Seti Değerlendirmesi ---")

    # Modeli tüm train_val verisiyle yeniden eğit
    full_train_generator = train_datagen.flow_from_dataframe(
        dataframe=train_val_df,
        x_col='filepath',
        y_col='label',
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        shuffle=True
    )
    test_generator = test_datagen.flow_from_dataframe(
        dataframe=test_df,
        x_col='filepath',
        y_col='label',
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='categorical',
        shuffle=False
    )

    final_model = model_builder(input_shape=(IMG_SIZE[0], IMG_SIZE[1],
3), n_classes=N_CLASSES)
    final_model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    # En iyi epoch sayısını bulmak için EarlyStopping ile tekrar eğit
```

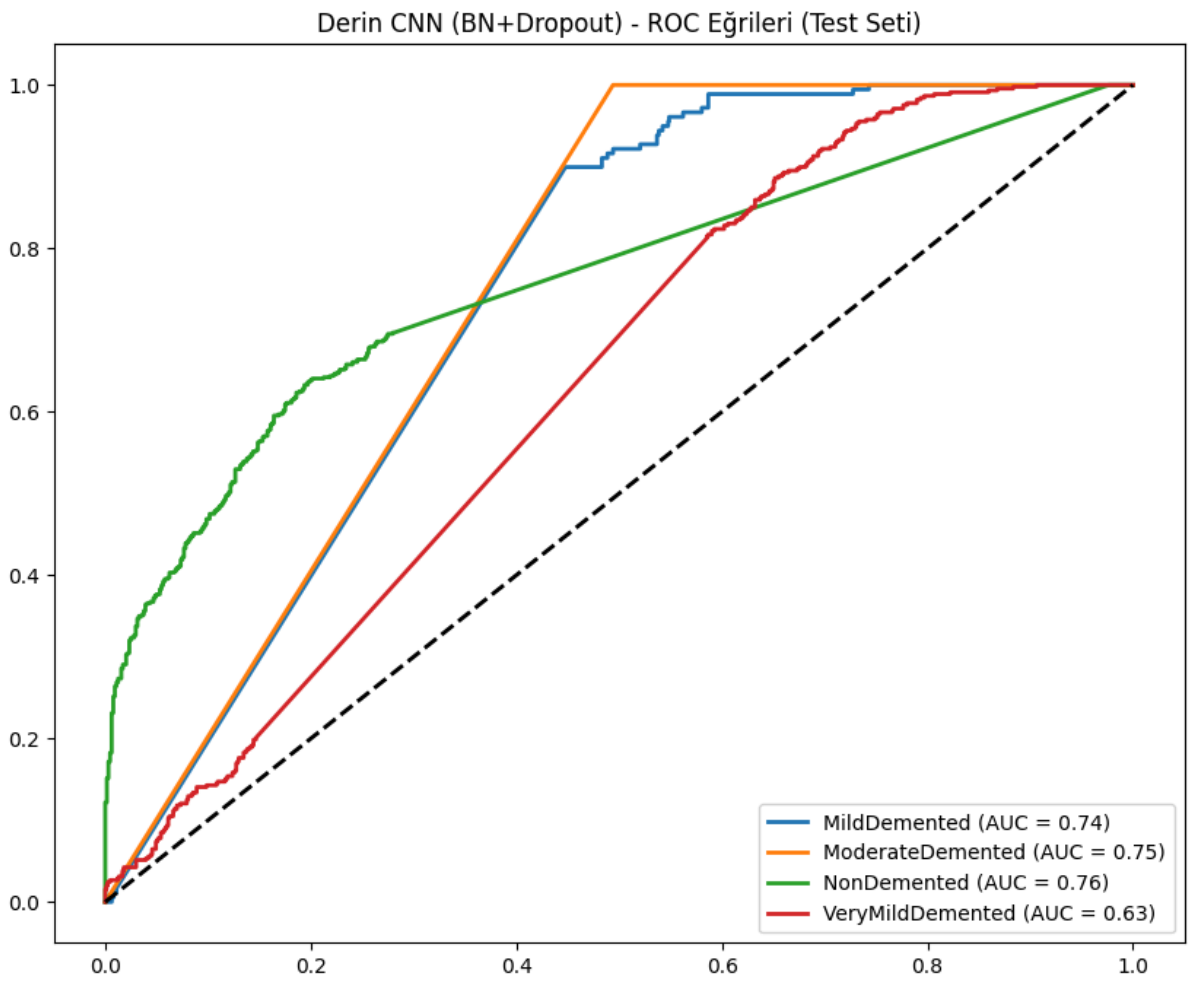
```
    final_model.fit(
        full_train_generator,
        epochs=50,
        validation_data=test_generator, # Test setini burada doğrulama
için kullanabiliriz
        callbacks=[EarlyStopping(monitor='val_loss', patience=10),
ReduceLROnPlateau(monitor='val_loss', patience=5)],
        verbose=1
    )

    # Test setinde tahmin yap
    y_pred_probs = final_model.predict(test_generator)
    y_pred_classes = np.argmax(y_pred_probs, axis=1)
    y_true_classes = test_generator.classes

    # Detaylı Rapor
    print(f"\n--- {model_name} | Detaylı Sınıflandırma Raporu (Test
Seti) ---")
    report = classification_report(y_true_classes, y_pred_classes,
target_names=class_names, output_dict=True)
    print(pd.DataFrame(report).transpose())

    # Karışıklık Matrisi
    plt.figure(figsize=(8, 6))
    cm = confusion_matrix(y_true_classes, y_pred_classes)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
    plt.title(f'{model_name} - Karışıklık Matrisi (Test Seti)')
    plt.xlabel('Tahmin Edilen')
    plt.ylabel('Gerçek')
    plt.show()

    # ROC Eğrileri
    plt.figure(figsize=(10, 8))
    for i, class_name in enumerate(class_names):
        fpr, tpr, _ = roc_curve(y_true_classes, y_pred_probs[:, i],
pos_label=i)
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, lw=2, label=f'{class_name} (AUC =
{roc_auc:.2f})')

    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.title(f'{model_name} - ROC Eğrileri (Test Seti)')
```

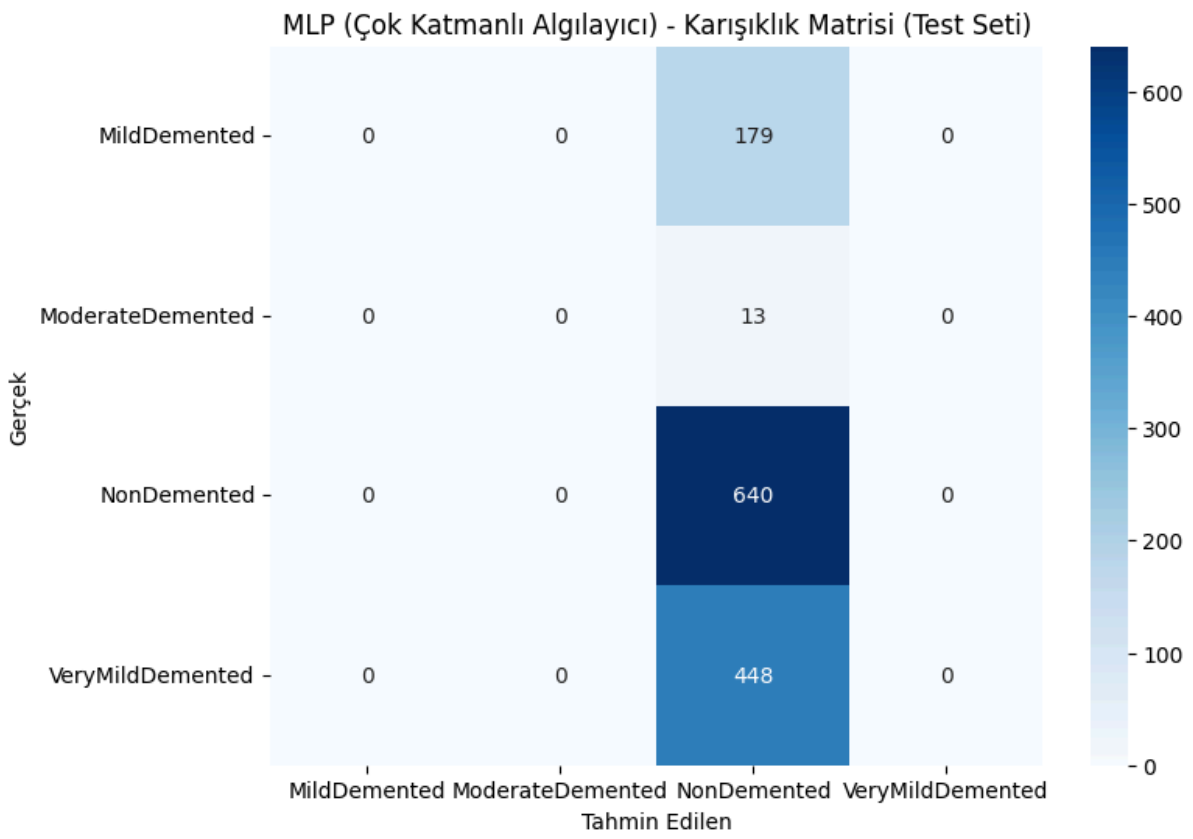
```
    plt.legend(loc="lower right")
    plt.show()

    # Özet tablo için sonuçları sakla
    dl_final_results[model_name] = report


# --- Derin Öğrenme Modellerinin Sonuçlarını Özetleme ---
print("\n" + "="*80)
print("--- DERİN ÖĞRENME MODELLERİ K-FOLD CV SONUÇ ÖZETİ ---")
dl_cv_results_df = pd.DataFrame(dl_cv_results)
print(dl_cv_results_df)

print("\n--- DERİN ÖĞRENME MODELLERİ NİHAİ TEST SETİ PERFORMANS ÖZETİ
---")
summary_list = []
for name, report in dl_final_results.items():
    summary_list.append({
        "Model": name,
        "Doğruluk (ACC)": report['accuracy'],
        "F1-Skoru (Makro)": report['macro avg']['f1-score'],
        "F1-Skoru (Ağırlıklı)": report['weighted avg']['f1-score']
    })

dl_final_results_df = pd.DataFrame(summary_list)
print(dl_final_results_df)

dl_final_results_df.set_index('Model').plot(kind='bar', figsize=(10,
6))
plt.title('Derin Öğrenme Modelleri - Nihai Test Seti Performansı')
plt.ylabel('Skor')
plt.xticks(rotation=45)
plt.show()


print("\n\n################################################################
####################")
print("# DERİN ÖĞRENME AŞAMASI TAMAMLANDI #")
print("################################################################
################")
```

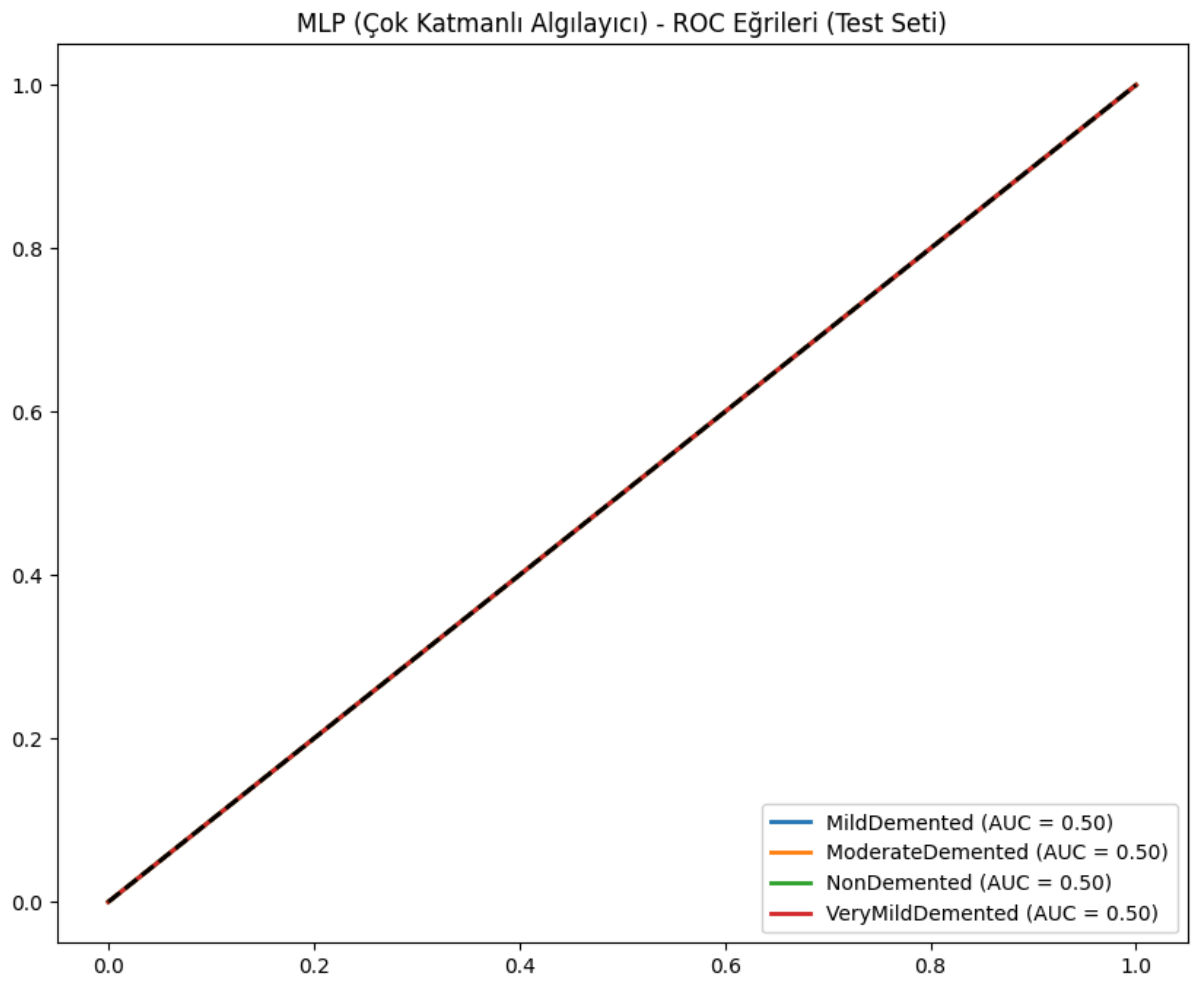

Temel CNN - Karışıklık Matrisi (Test Seti)
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Gerçek
107
0
11
61
9
0
0
4
66
0
411
163
125
0
108
215
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Tahmin Edilen
400
350
300
250
200
150
100
50
0


Temel CNN - ROC Eğrileri (Test Seti)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
MildDemented (AUC = 0.81)
ModerateDemented (AUC = 0.94)
NonDemented (AUC = 0.79)
VeryMildDemented (AUC = 0.69)


Derin CNN (BN+Dropout) - Karışıklık Matrisi (Test Seti)
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Gerçek
0
0
18
161
0
0
0
13
0
0
440
200
0
0
156
292
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Tahmin Edilen
400
350
300
250
200
150
100
50
0


Derin CNN (BN+Dropout) - ROC Eğrileri (Test Seti)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
MildDemented (AUC = 0.74)
ModerateDemented (AUC = 0.75)
NonDemented (AUC = 0.76)
VeryMildDemented (AUC = 0.63)


MLP (Çok Katmanlı Algılayıcı) - Karışıklık Matrisi (Test Seti)
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Gerçek
0
0
179
0
0
0
13
0
0
0
640
0
0
0
448
0
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Tahmin Edilen
600
500
400
300
200
100
0



Kalan 2 Derin Öğrenme

```
# --- Gerekli Kütüphaneler ---
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split

print("--- Hazırlık Hücresi Çalıştırılıyor ---")


data_dir = '/content/Alzheimer_s Dataset/train'
if not os.path.exists(data_dir):
    print("HATA: Veri seti klasörü bulunamadı. Lütfen zip dosyasını
yükleyip !unzip komutunu çalıştırdığınızdan emin olun.")
else:
    filepaths, labels = [], []
```

```
    for label in os.listdir(data_dir):
        class_path = os.path.join(data_dir, label)
        if os.path.isdir(class_path):
            for file in os.listdir(class_path):
                filepaths.append(os.path.join(class_path, file))
                labels.append(label)
    image_df = pd.DataFrame({'filepath': filepaths, 'label': labels})

    # Sentetik Veri
    def generate_synthetic_data(label):
        if label == 'NonDemented': age, mmse, educ =
np.random.randint(60, 75), np.random.uniform(27, 30),
np.random.randint(12, 20)
        elif label == 'VeryMildDemented': age, mmse, educ =
np.random.randint(65, 80), np.random.uniform(24, 27),
np.random.randint(10, 18)
        elif label == 'MildDemented': age, mmse, educ =
np.random.randint(70, 85), np.random.uniform(20, 24),
np.random.randint(8, 16)
        else: age, mmse, educ = np.random.randint(75, 90),
np.random.uniform(10, 20), np.random.randint(6, 14)
        gender = np.random.choice(['M', 'F'])
        return age, round(mmse, 2), educ, gender

    synthetic_data = image_df['label'].apply(generate_synthetic_data)
    synthetic_df = pd.DataFrame(synthetic_data.tolist(),
columns=['Age', 'MMSE', 'EDUC', 'Gender'])
    final_df = pd.concat([image_df, synthetic_df], axis=1)
    print("Adım 1: Sentetik verilerle birleştirilmiş ana DataFrame
'final_df' oluşturuldu.")

    # --- Veriyi Bölme ---

    global train_val_df, test_df, class_names, N_CLASSES
    global train_datagen, test_datagen, IMG_SIZE, BATCH_SIZE

    train_val_df, test_df = train_test_split(final_df, test_size=0.20,
random_state=42, stratify=final_df['label'])
    class_names = sorted(train_val_df['label'].unique())
    N_CLASSES = len(class_names)
    print("Adım 2: Veri, 'train_val_df' ve 'test_df' olarak ikiye
ayrıldı.")
```

```
    # --- Derin Öğrenme için Veri Üreteçleri---
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    IMG_SIZE = (128, 128)
    BATCH_SIZE = 32
    train_datagen = ImageDataGenerator(rescale=1./255,
rotation_range=20, width_shift_range=0.15, height_shift_range=0.15,
shear_range=0.15, zoom_range=0.15, horizontal_flip=True,
fill_mode='nearest')
    test_datagen = ImageDataGenerator(rescale=1./255)
    print("Adım 3: Derin öğrenme için veri üreteçleri 'train_datagen'
ve 'test_datagen' tanımlandı.")

    print("\n--- Hazırlık Tamamlandı! Artık istediğiniz modelin eğitim
hücresini çalıştırabilirsiniz. ---")
```

```
# Gerekli kütüphaneler
import tensorflow as tf
from tensorflow.keras import layers, Model
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix,
roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns

print("--- Model 8: Vision Transformer (ViT) için hazırlıklar başlıyor.
---")

# --- ViT için Özel Katmanlar ve Fonksiyonlar ---
class Patches(layers.Layer):
    def __init__(self, patch_size, **kwargs):
        super(Patches, self).__init__(**kwargs)
        self.patch_size = patch_size
    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(images=images, sizes=[1,
self.patch_size, self.patch_size, 1], strides=[1, self.patch_size,
self.patch_size, 1], rates=[1, 1, 1, 1], padding="VALID")
        patch_dims = patches.shape[-1]
```

```
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches
    def get_config(self):
        config = super().get_config(); config.update({"patch_size":
self.patch_size}); return config

class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim, **kwargs):
        super(PatchEncoder, self).__init__(**kwargs)
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding =
layers.Embedding(input_dim=num_patches, output_dim=projection_dim)
    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) +
self.position_embedding(positions)
        return encoded
    def get_config(self):
        config = super().get_config(); config.update({"num_patches":
self.num_patches, "projection_dim": self.projection_dim}); return
config

# --- ViT Model Mimarisi ---
def build_simple_vit(input_shape, n_classes, patch_size=16,
projection_dim=64, num_transformer_layers=4, num_heads=4,
mlp_head_units=[128, 64]):
    inputs = layers.Input(shape=input_shape)
    patches = Patches(patch_size)(inputs)
    num_patches = (input_shape[0] // patch_size) ** 2
    encoded_patches = PatchEncoder(num_patches,
projection_dim)(patches)
    for _ in range(num_transformer_layers):
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        attention_output =
layers.MultiHeadAttention(num_heads=num_heads, key_dim=projection_dim,
dropout=0.1)(x1, x1)
        x2 = layers.Add()([attention_output, encoded_patches])
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        x3 = layers.Dense(projection_dim * 2, activation="gelu")(x3)
        x3 = layers.Dense(projection_dim)(x3)
        encoded_patches = layers.Add()([x3, x2])
```

```
    representation =
layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    for units in mlp_head_units:
        representation = layers.Dense(units,
activation="gelu")(representation)
        representation = layers.Dropout(0.3)(representation)
    outputs = layers.Dense(n_classes,
activation="softmax")(representation)
    model = Model(inputs=inputs, outputs=outputs, name="Simple_ViT")
    return model

# --- Model Eğitimi ve Değerlendirmesi ---
model_name = "Basit Vision Transformer"
model_builder = build_simple_vit
print(f"\n{'#' * 30}\n# Model: {model_name.upper()}\n{'#' * 30}")

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
fold_scores, fold_no = [], 1

for train_index, val_index in kfold.split(train_val_df,
train_val_df['label']):
    print(f"\n--- {model_name} | Fold {fold_no}/{kfold.get_n_splits()}
---")
    train_fold_df, val_fold_df = train_val_df.iloc[train_index],
train_val_df.iloc[val_index]

    train_generator =
train_datagen.flow_from_dataframe(dataframe=train_fold_df,
x_col='filepath', y_col='label', target_size=IMG_SIZE,
batch_size=BATCH_SIZE, class_mode='categorical', shuffle=True)
    validation_generator =
test_datagen.flow_from_dataframe(dataframe=val_fold_df,
x_col='filepath', y_col='label', target_size=IMG_SIZE,
batch_size=BATCH_SIZE, class_mode='categorical', shuffle=False)

    model = model_builder(input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3),
n_classes=N_CLASSES)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
```

```
    model.fit(train_generator, epochs=50,
validation_data=validation_generator, verbose=1)

    loss, accuracy = model.evaluate(validation_generator)
    print(f"Fold {fold_no} Sonucu: Accuracy = {accuracy:.4f}")
    fold_scores.append(accuracy)
    fold_no += 1

print(f"\n--- {model_name} | K-Fold Sonuç Özeti ---")
print(f"Ortalama Doğruluk: {np.mean(fold_scores):.4f} ±
{np.std(fold_scores):.4f}")

# --- Nihai Test Seti Değerlendirmesi ---
print(f"\n--- {model_name} | Nihai Test Seti Değerlendirmesi ---")
full_train_generator =
train_datagen.flow_from_dataframe(dataframe=train_val_df,
x_col='filepath', y_col='label', target_size=IMG_SIZE,
batch_size=BATCH_SIZE, class_mode='categorical', shuffle=True)
test_generator = test_datagen.flow_from_dataframe(dataframe=test_df,
x_col='filepath', y_col='label', target_size=IMG_SIZE,
batch_size=BATCH_SIZE, class_mode='categorical', shuffle=False)

final_model = model_builder(input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3),
n_classes=N_CLASSES)
final_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.
001), loss='categorical_crossentropy', metrics=['accuracy'])
final_model.fit(full_train_generator, epochs=50,
validation_data=test_generator, verbose=1)

y_pred_probs = final_model.predict(test_generator)
y_pred_classes = np.argmax(y_pred_probs, axis=1)
y_true_classes = test_generator.classes

# Detaylı Rapor
print(f"\n--- {model_name} | Detaylı Sınıflandırma Raporu (Test Seti)
---")
report = classification_report(y_true_classes, y_pred_classes,
target_names=class_names, output_dict=True)
print(pd.DataFrame(report).transpose())

# Karışıklık Matrisi
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_true_classes, y_pred_classes)
```

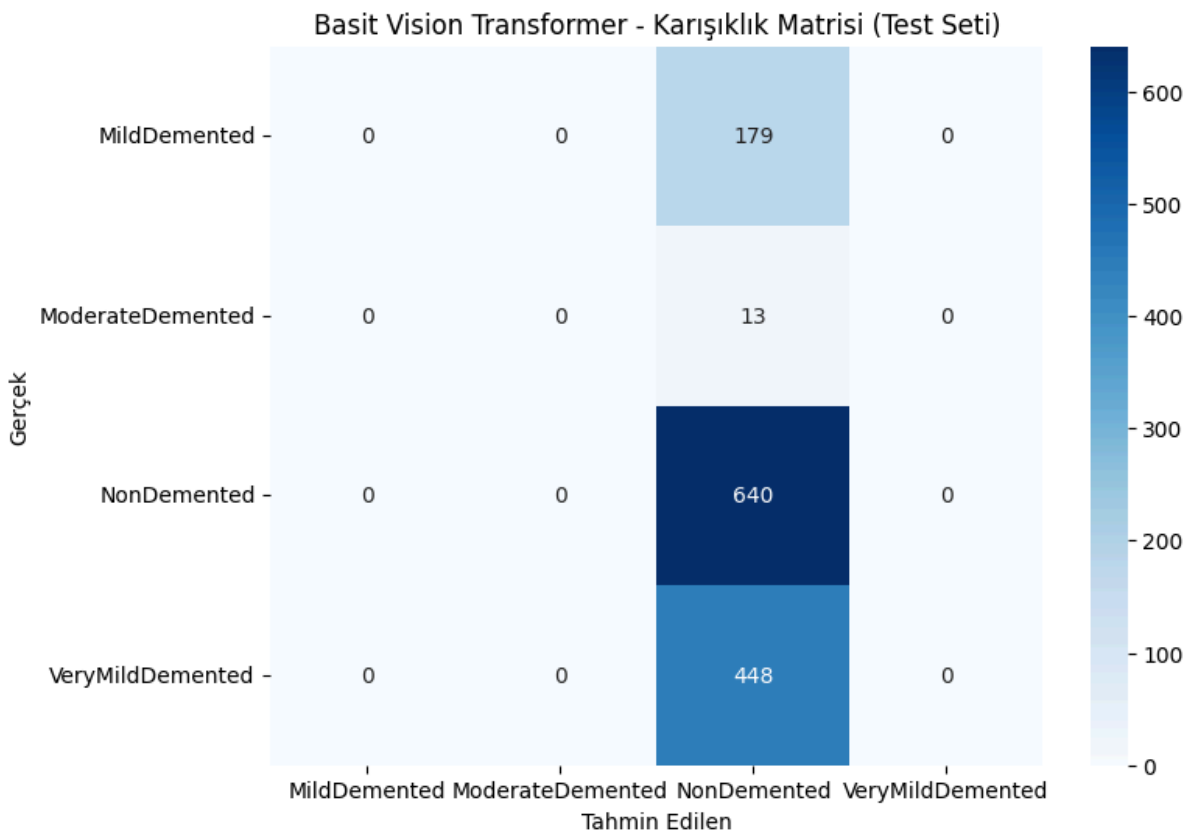
```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
plt.title(f'{model_name} - Karışıklık Matrisi (Test Seti)')
plt.xlabel('Tahmin Edilen')
plt.ylabel('Gerçek')
plt.show()

# ROC Eğrileri
plt.figure(figsize=(10, 8))
for i, class_name in enumerate(class_names):
    fpr, tpr, _ = roc_curve(y_true_classes, y_pred_probs[:, i],
pos_label=i)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f'{class_name} (AUC =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.title(f'{model_name} - ROC Eğrileri (Test Seti)')
plt.legend(loc="lower right")
plt.show()

print(f"\n{'#' * 30}\n# {model_name.upper()} EĞİTİMİ TAMAMLANDI\n{'#' *
30}")
```

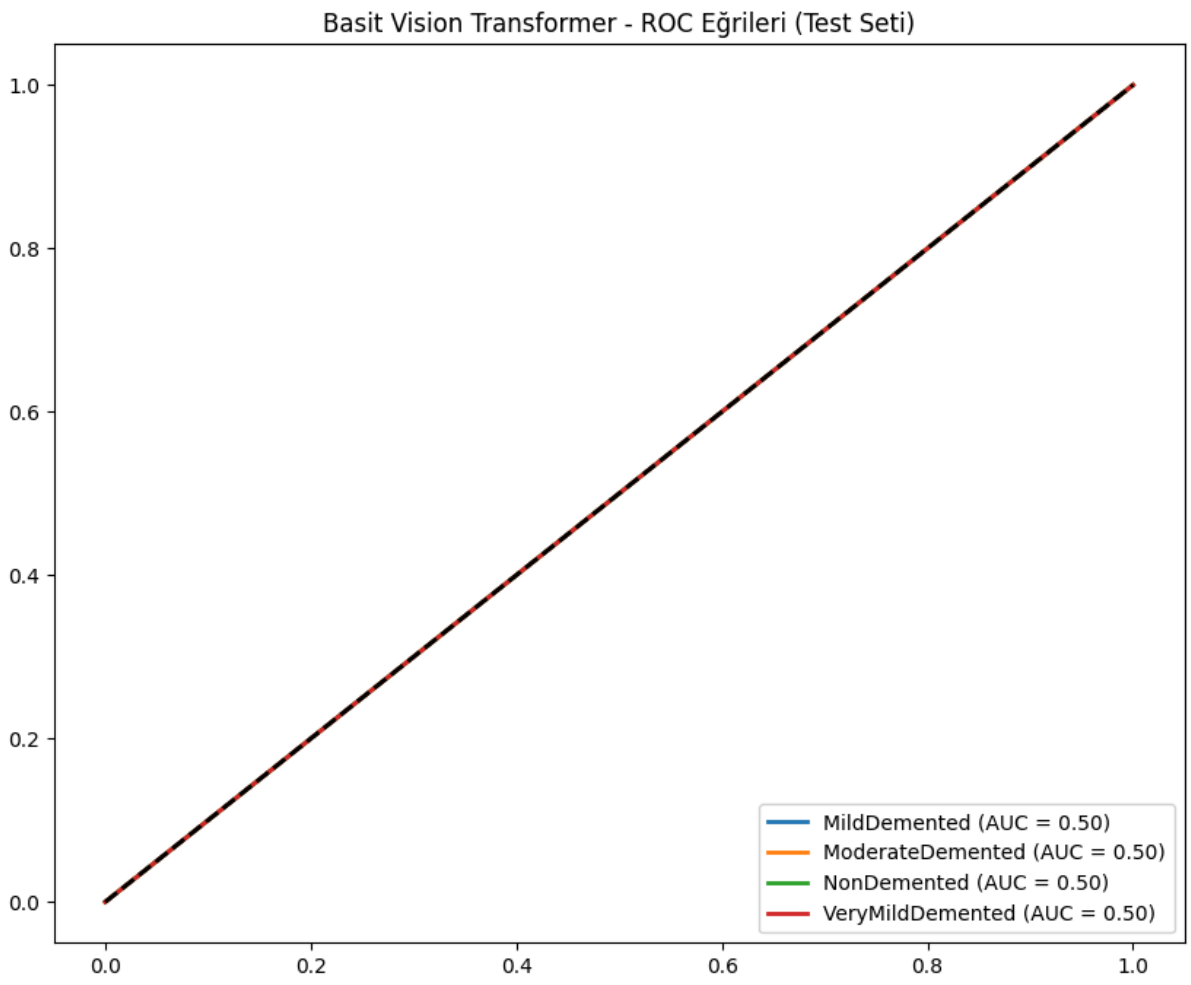
```
import tensorflow as tf
from tensorflow.keras import layers, Model
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix,
roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns

print("--- Model 10: Otomatik Kodlayıcı Tabanlı Sınıflandırıcı için
hazırlıklar başlıyor. ---")

# --- CAE Model Mimarisi ---
def build_cae_classifier(input_shape, n_classes, latent_dim=128):
    # 1. Kodlayıcı (Encoder) Kısmı
    encoder_inputs = layers.Input(shape=input_shape)
```

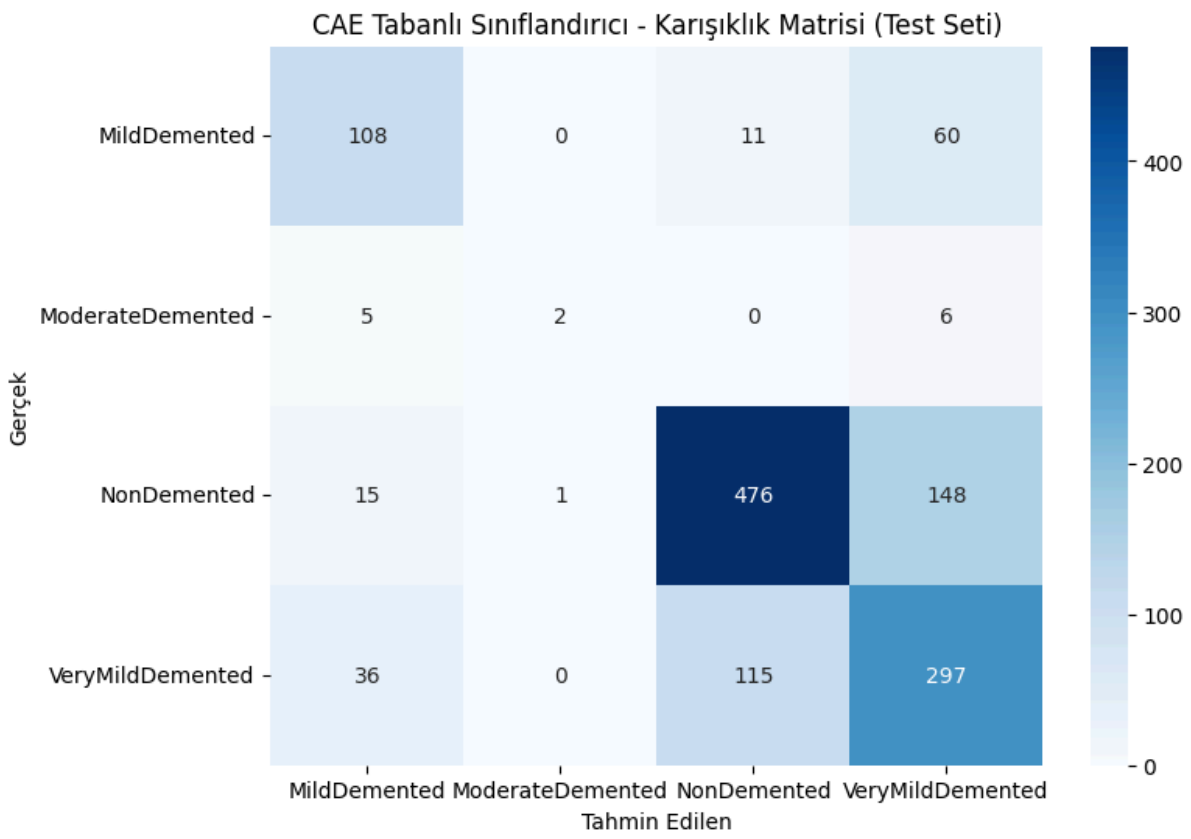
```
    x = layers.Conv2D(32, (3, 3), activation='relu',
padding='same')(encoder_inputs)
    x = layers.MaxPooling2D((2, 2), padding='same')(x)
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2), padding='same')(x)
    x = layers.Conv2D(128, (3, 3), activation='relu',
padding='same')(x)
    x = layers.MaxPooling2D((2, 2), padding='same')(x)
    x = layers.Flatten()(x)
    latent_vector = layers.Dense(latent_dim, activation='relu',
name='latent_vector')(x)

    encoder = Model(encoder_inputs, latent_vector, name='encoder')

    # 2. Sınıflandırıcı Başlığı
    classifier_outputs = layers.Dense(256,
activation='relu')(encoder.output)
    classifier_outputs = layers.Dropout(0.5)(classifier_outputs)
    classifier_outputs = layers.Dense(n_classes,
activation='softmax')(classifier_outputs)

    final_model = Model(encoder.input, classifier_outputs,
name='CAE_Classifier')
    return final_model

# --- Model Eğitimi ve Değerlendirmesi ---
model_name = "CAE Tabanlı Sınıflandırıcı"
model_builder = build_cae_classifier
print(f"\n{'#' * 30}\n# Model: {model_name.upper()}\n{'#' * 30}")

kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
fold_scores, fold_no = [], 1

for train_index, val_index in kfold.split(train_val_df,
train_val_df['label']):
    print(f"\n--- {model_name} | Fold {fold_no}/{kfold.get_n_splits()}
---")
    train_fold_df, val_fold_df = train_val_df.iloc[train_index],
train_val_df.iloc[val_index]

    train_generator =
train_datagen.flow_from_dataframe(dataframe=train_fold_df,
```

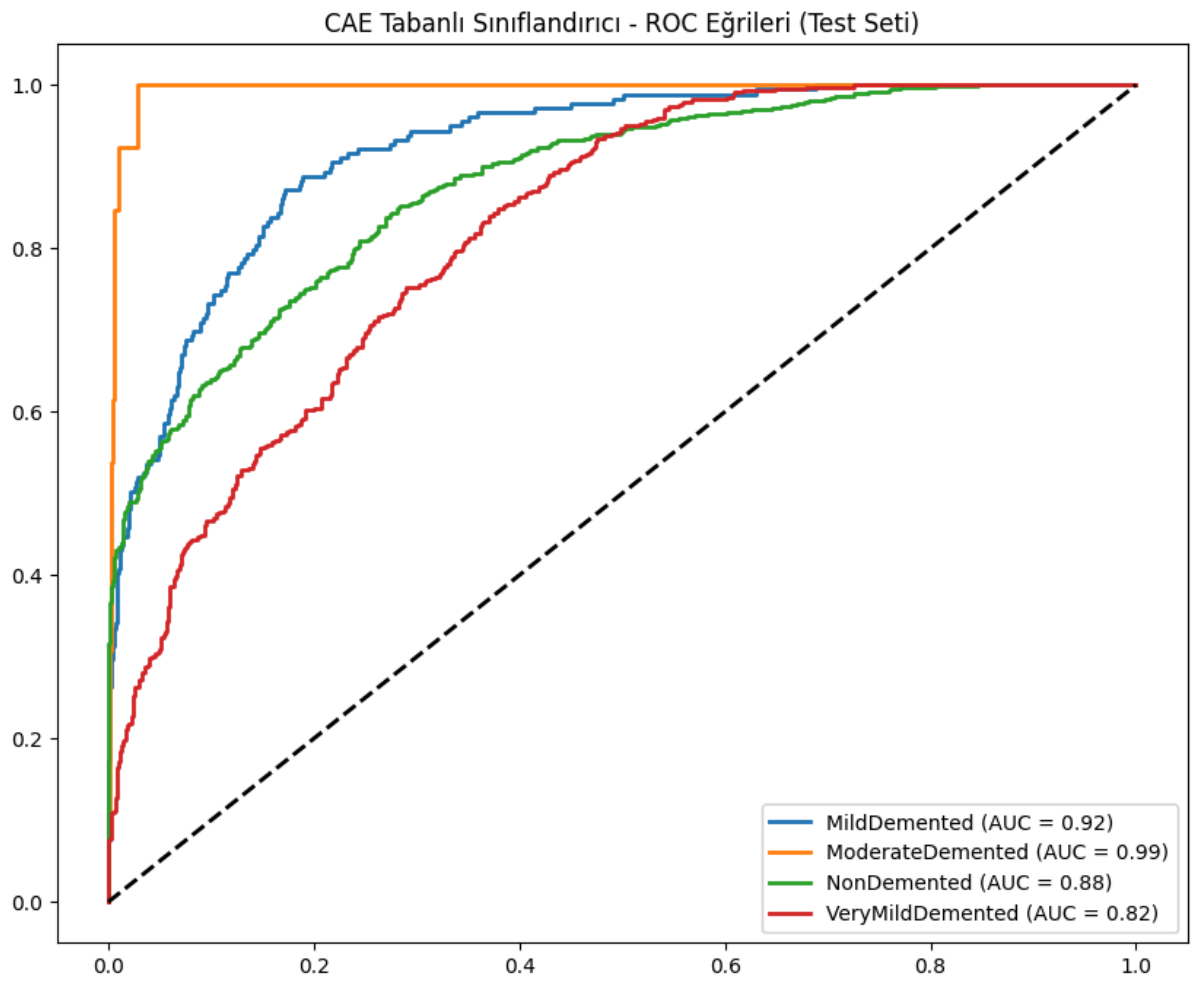
```
x_col='filepath', y_col='label', target_size=IMG_SIZE,
batch_size=BATCH_SIZE, class_mode='categorical', shuffle=True)
    validation_generator =
test_datagen.flow_from_dataframe(dataframe=val_fold_df,
x_col='filepath', y_col='label', target_size=IMG_SIZE,
batch_size=BATCH_SIZE, class_mode='categorical', shuffle=False)

    model = model_builder(input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3),
n_classes=N_CLASSES)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])

    model.fit(train_generator, epochs=50,
validation_data=validation_generator, verbose=1)

    loss, accuracy = model.evaluate(validation_generator)
    print(f"Fold {fold_no} Sonucu: Accuracy = {accuracy:.4f}")
    fold_scores.append(accuracy)
    fold_no += 1

print(f"\n--- {model_name} | K-Fold Sonuç Özeti ---")
print(f"Ortalama Doğruluk: {np.mean(fold_scores):.4f} ±
{np.std(fold_scores):.4f}")

# --- Nihai Test Seti Değerlendirmesi ---
print(f"\n--- {model_name} | Nihai Test Seti Değerlendirmesi ---")
full_train_generator =
train_datagen.flow_from_dataframe(dataframe=train_val_df,
x_col='filepath', y_col='label', target_size=IMG_SIZE,
batch_size=BATCH_SIZE, class_mode='categorical', shuffle=True)
test_generator = test_datagen.flow_from_dataframe(dataframe=test_df,
x_col='filepath', y_col='label', target_size=IMG_SIZE,
batch_size=BATCH_SIZE, class_mode='categorical', shuffle=False)

final_model = model_builder(input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3),
n_classes=N_CLASSES)
final_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.
001), loss='categorical_crossentropy', metrics=['accuracy'])
final_model.fit(full_train_generator, epochs=50,
validation_data=test_generator, verbose=1)

y_pred_probs = final_model.predict(test_generator)
```

```
y_pred_classes = np.argmax(y_pred_probs, axis=1)
y_true_classes = test_generator.classes

# Detaylı Rapor
print(f"\n--- {model_name} | Detaylı Sınıflandırma Raporu (Test Seti)
---")
report = classification_report(y_true_classes, y_pred_classes,
target_names=class_names, output_dict=True)
print(pd.DataFrame(report).transpose())

# Karışıklık Matrisi
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_true_classes, y_pred_classes)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
plt.title(f'{model_name} - Karışıklık Matrisi (Test Seti)')
plt.xlabel('Tahmin Edilen')
plt.ylabel('Gerçek')
plt.show()

# ROC Eğrileri
plt.figure(figsize=(10, 8))
for i, class_name in enumerate(class_names):
    fpr, tpr, _ = roc_curve(y_true_classes, y_pred_probs[:, i],
pos_label=i)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=2, label=f'{class_name} (AUC =
{roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.title(f'{model_name} - ROC Eğrileri (Test Seti)')
plt.legend(loc="lower right")
plt.show()

print(f"\n{'#' * 30}\n# {model_name.upper()} EĞİTİMİ TAMAMLANDI\n{'#' *
30}")
```


Basit Vision Transformer - Karışıklık Matrisi (Test Seti)
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Gerçek
0
0
179
0
0
0
13
0
0
0
640
0
0
0
448
0
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Tahmin Edilen
600
500
400
300
200
100
0


Basit Vision Transformer - ROC Eğrileri (Test Seti)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
MildDemented (AUC = 0.50)
ModerateDemented (AUC = 0.50)
NonDemented (AUC = 0.50)
VeryMildDemented (AUC = 0.50)


CAE Tabanlı Sınıflandırıcı - Karışıklık Matrisi (Test Seti)
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Gerçek
108
0
11
60
5
2
0
6
15
1
476
148
36
0
115
297
MildDemented
ModerateDemented
NonDemented
VeryMildDemented
Tahmin Edilen
400
300
200
100
0

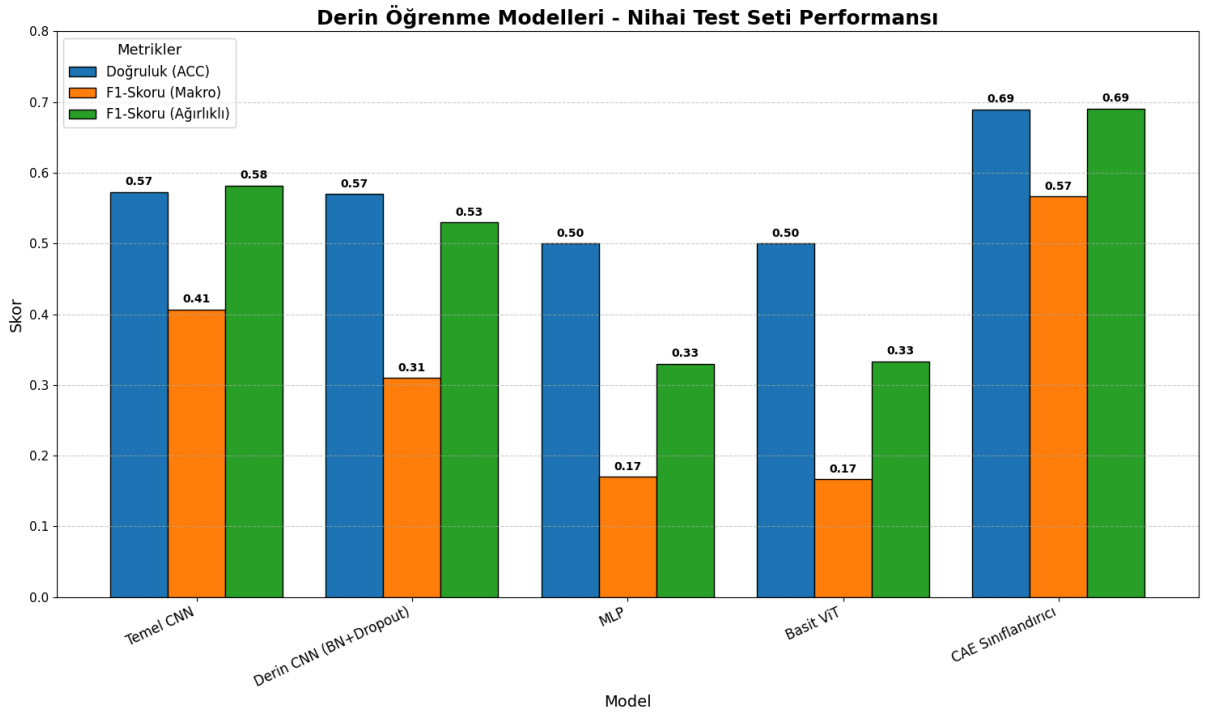


Grafikleri Birleştirme

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# --- TÜM MODELLERİN SONUÇLARINI BURAYA GİRİYORUZ ---

temel_cnn_acc = 0.5727
temel_cnn_f1_macro = 0.4064
temel_cnn_f1_weighted = 0.5818


derin_cnn_acc = 0.57
derin_cnn_f1_macro = 0.31
derin_cnn_f1_weighted = 0.53

mlp_acc = 0.50
```

```
mlp_f1_macro = 0.17
mlp_f1_weighted = 0.33


vit_acc = 0.5000
vit_f1_macro = 0.1667
vit_f1_weighted = 0.3333


cae_acc = 0.6898
cae_f1_macro = 0.5664
cae_f1_weighted = 0.6906


# --- GRAFİĞİ OLUŞTURMA ---

# Tüm verileri tek bir yapıda birleştirelim
data = {
    'Doğruluk (ACC)': [temel_cnn_acc, derin_cnn_acc, mlp_acc, vit_acc,
cae_acc],
    'F1-Skoru (Makro)': [temel_cnn_f1_macro, derin_cnn_f1_macro,
mlp_f1_macro, vit_f1_macro, cae_f1_macro],
    'F1-Skoru (Ağırlıklı)': [temel_cnn_f1_weighted,
derin_cnn_f1_weighted, mlp_f1_weighted, vit_f1_weighted,
cae_f1_weighted]
}

# Model isimleri
model_names = [
    'Temel CNN',
    'Derin CNN (BN+Dropout)',
    'MLP',
    'Basit ViT',
    'CAE Sınıflandırıcı'
]

# Verileri bir Pandas DataFrame'e dönüştürme
df = pd.DataFrame(data, index=model_names)

# Grafiği çizdirme
ax = df.plot(kind='bar', figsize=(15, 9), width=0.8, edgecolor='black')

# Başlık ve etiketleri ayarlayalım
```

```
plt.title('Derin Öğrenme Modelleri - Nihai Test Seti Performansı',
fontsize=18, weight='bold')
plt.ylabel('Skor', fontsize=14)
plt.xlabel('Model', fontsize=14)

# X eksenindeki model isimlerini daha okunaklı hale getirme
plt.xticks(rotation=25, ha="right", fontsize=12)
plt.yticks(fontsize=11)

# Y ekseni limitlerini ayarlayarak daha net bir görünüm
plt.ylim(0, max(df.max().max(), 0.7) + 0.1)

# Barların üzerine değerlerini yazdı
for p in ax.patches:
    ax.annotate(f'{p.get_height():.2f}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center',
                xytext=(0, 9),
                textcoords='offset points',
                fontsize=10,
                weight='bold')

# Grafiğin üzerine kılavuz çizgileri ekle
plt.grid(axis='y', linestyle='--', alpha=0.6)

# Lejantı (gösterge) daha belirgin hale getir
plt.legend(fontsize=12, title='Metrikler', title_fontsize='13')

# Grafiği gösterelim
plt.tight_layout() # Kenar boşluklarını otomatik ayarlar
plt.show()

# --- SONUÇLARI ÖZET TABLO OLARAK DA GÖSTERELİM ---
print("\n--- Tüm Derin Öğrenme Modelleri İçin Nihai Performans Özeti
---")
pd.options.display.float_format = '{:.4f}'.format # Ondalık formatı
ayarla
print(df)
```


Derin Öğrenme Modelleri - Nihai Test Seti Performansı
Metrikler
Doğruluk (ACC)
F1-Skoru (Makro)
F1-Skoru (Ağırlıklı)
Skor
Model
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.57
0.41
0.58
0.57
0.31
0.53
0.50
0.17
0.33
0.50
0.17
0.33
0.69
0.57
0.69
Temel CNN
Derin CNN (BN+Dropout)
MLP
Basit ViT
CAE Sınıflandırıcı