YSA İLE MNSİT(EL YAZISI RAKAMLAR) VERİ SETİ ANALİZİ

MNIST Veri Seti ile El Yazısı Rakam Sınıflandırma

Bu proje, derin öğrenme ve bilgisayarlı görü alanlarının en temel ve popüler veri setlerinden biri olan **MNIST**'yi kullanarak el yazısı rakamları tanıyan bir Yapay Sinir Ağı (YSA) modeli oluşturmayı ve eğitmeyi amaçlamaktadır.

MNIST Veri Seti Nedir?

MNIST (Modified National Institute of Standards and Technology database), makine öğrenmesi ve bilgisayarlı görü algoritmalarını test etmek ve eğitmek için yaygın olarak kullanılan büyük bir veri setidir. "Derin Öğrenmenin Merhaba Dünyası" olarak da anılır.

- **İçerik:** 0'dan 9'a kadar olan el yazısı rakamların gri tonlamalı (grayscale) resimlerini içerir.
- **Boyut:** Her bir resim 28×28 piksel boyutundadır.
- **Veri Miktarı:** Veri seti, modelin öğrenmesi için **60,000 eğitim** resmi ve modelin performansını test etmek için **10,000 test** resmi olmak üzere ikiye ayrılmıştır.

Projemiz, bu 28×28 piksellik resimlere bakarak içindeki rakamın ne olduğunu %95'in üzerinde bir doğrulukla tahmin edebilen bir model inşa edecektir.

---

1. Kütüphanelerin Yüklenmesi

Projenin ilk adımı, gerekli olan tüm Python kütüphanelerini ve modüllerini çalışma ortamımıza dahil etmektir.

Python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
```

Açıklama:

- `numpy`: Sayısal ve matris işlemleri için temel kütüphane.
- `matplotlib.pyplot`: Veri görselleştirme ve grafik çizimi için kullanılır.
- `tensorflow.keras`: Derin öğrenme modelleri oluşturmak için kullandığımız ana kütüphane.
  - `Dense`, `Flatten`: Modelimizin katmanlarını oluşturmak için kullanacağımız katman türleri.
  - `Sequential`: Katmanları doğrusal bir sıra halinde ekleyerek model oluşturmamızı sağlayan model türü.
  - `mnist`: Keras'ın içinde hazır olarak bulunan MNIST veri setini yüklemek için kullanılır.
  - `to_categorical`: Etiketlerimizi "One-Hot Encoding" yöntemine dönüştürmek için kritik bir yardımcı fonksiyon.
  - `EarlyStopping`: Modelin aşırı öğrenmesini (overfitting) engellemek için bir kontrol mekanizması.

---

2. Veri Setinin Yüklenmesi ve İncelenmesi

Keras kütüphanesinin sağladığı kolaylık sayesinde MNIST veri setini tek bir satır kod ile indirip eğitim ve test setleri olarak ayrılmış şekilde yüklüyoruz.

```
(X_train, y_train) , (X_test, y_test) = mnist.load_data()
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

Çıktı:

```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

Açıklama:

- `mnist.load_data()`: Veri setini otomatik olarak indirir ve `(eğitim_resimleri, eğitim_etiketleri)` ile `(test_resimleri, test_etiketleri)` olarak iki demet (tuple) halinde döndürür.
- `X_train.shape`: Eğitim setinde **60,000** adet, **28×28** piksel boyutunda resim olduğunu gösterir.
- `y_train.shape`: Bu 60,000 resmin her birine karşılık gelen bir rakam etiketi (0-9 arası) olduğunu gösterir.

---

3. Veri Ön İşleme: One-Hot Encoding'in Büyüsü

Bu adım, projenin en can alıcı ve anlaşılması en önemli kısımlarından biridir. Modelimizin anlayabilmesi için etiketlerimizi (`y_train`, `y_test`) sayısal bir formattan kategorik bir formata dönüştürüyoruz.

Neden Gerekli?

Modelimizin son katmanı, 10 rakamdan (0-9) her biri için bir olasılık değeri üreten softmax aktivasyon fonksiyonunu kullanacak. Örneğin, bir resim için modelin çıktısı şöyle bir olasılık listesi olabilir: [0.01, 0.05, 0.0, 0.85, ..., 0.02]. Bu liste, resmin %85 ihtimalle "3" olduğunu söyler.

Ancak bizim orijinal etiketlerimiz `3`, `5`, `7` gibi tek bir sayıdan oluşuyor. Modelin `[0.01, ..., 0.85, ...]` şeklindeki tahminini, `3` gibi tek bir sayıyla doğrudan karşılaştırıp hatasını hesaplaması matematiksel olarak zordur. İşte bu noktada **One-Hot Encoding** devreye girer.

One-Hot Encoding Nedir?

Bu yöntem, kategorik bir etiketi, tüm olası kategorileri temsil eden bir binary (0 ve 1'lerden oluşan) vektöre dönüştürür. Bizim durumumuzda 10 rakam (kategori) var.

Örnek:

- `5` rakamının etiketi `5`'tir.
- One-Hot Encoding ile bu etiket **10 elemanlı** bir vektöre dönüşür.
- Bu vektörde sadece **5. indeksteki** (indeksleme 0'dan başlar) değer **1**, diğer tüm değerler **0** olur.

Yani:

- Etiket `2` → `[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]`
- Etiket `7` → `[0, 0, 0, 0, 0, 0, 0, 1, 0, 0]`
- Etiket `0` → `[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]`

Bu dönüşüm sayesinde, modelin `softmax` çıktısı olan olasılık vektörü ile gerçek etiket vektörü artık aynı formattadır ve `categorical_crossentropy` gibi kayıp fonksiyonları ile kolayca karşılaştırılabilir.

```
# Bu kod, her bir etiketi tek tek dönüştürür.
temp = []
for i in range(len(y_train)):
    temp.append(to_categorical(y_train[i], num_classes=10))
y_train = np.array(temp)

temp = []
for i in range(len(y_test)):
    temp.append(to_categorical(y_test[i], num_classes=10))
y_test = np.array(temp)
# Daha pratik ve hızlı bir alternatif yol:
# y_train = to_categorical(y_train, num_classes=10)
# y_test = to_categorical(y_test, num_classes=10)
```

---

4. Yapay Sinir Ağı Modelini Oluşturma

Modelimizi Keras'ın `Sequential` API'si ile katman katman inşa ediyoruz. Bu, katmanları doğrusal bir şekilde üst üste eklediğimiz basit ve güçlü bir yöntemdir.

```
model = Sequential()

# 1. Düzleştirme Katmanı (Flatten)
model.add(Flatten(input_shape = (28,28)))

# 2. Gizli Katman
model.add(Dense(units=128, activation='tanh'))

# 3. Gizli Katman
model.add(Dense(units=64, activation='tanh'))

# 4. Çıkış Katmanı
model.add(Dense(units=10, activation='softmax'))

model.summary()
```

Açıklama:

1. `model = Sequential()`: Üzerine katmanlar ekleyeceğimiz boş, sıralı bir model oluşturur.
2. `Flatten(input_shape = (28,28))`: Bu ilk katman çok önemlidir. Girdi olarak aldığı **28×28** boyutundaki 2 boyutlu resim matrisini, **784** (28 * 28) elemanlı tek boyutlu bir vektöre "düzleştirir". Çünkü `Dense` katmanları 2D veriyi değil, 1D vektörleri işleyebilir.
3. `Dense(units=128, activation='tanh')`: 128 nörondan oluşan tam bağlantılı bir gizli katmandır. Bu katman, resimdeki desenleri ve özellikleri öğrenmeye başlar. `tanh` aktivasyon fonksiyonu kullanılır.
4. `Dense(units=64, activation='tanh')`: 64 nörondan oluşan ikinci bir gizli katman. Modelin daha karmaşık ilişkileri öğrenmesine yardımcı olur.
5. `Dense(units=10, activation='softmax')`: Bu, modelin karar verdiği **çıkış katmanıdır**.

- `units=10`: Sınıflandırmak istediğimiz 10 farklı rakam olduğu için 10 nöron bulunur. Her nöron bir rakamı temsil eder.
- `activation='softmax'`: Bu aktivasyon fonksiyonu, 10 nöronun ham çıktılarını alır ve bunları bir olasılık dağılımına dönüştürür. Yani, 10 nöronun çıktılarının toplamı 1 olur ve her bir çıktı, resmin o rakama ait olma olasılığını gösterir. En yüksek olasılığa sahip nöron, modelin tahminidir.

6. `model.summary()`: Oluşturulan modelin mimarisini, katmanlarını ve parametre sayısını özetleyen bir tablo sunar.

---

5. Modeli Derleme

Model mimarisini oluşturduktan sonra, eğitim sürecini yapılandırmak için onu "derlememiz" gerekir.

```
model.compile(loss = 'categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Açıklama:

- `loss = 'categorical_crossentropy'`: Kayıp fonksiyonu olarak seçilir. Bu fonksiyon, **One-Hot Encoding** formatındaki gerçek etiketler ile modelin `softmax` katmanından çıkan olasılık tahminleri arasındaki farkı ölçmek için özel olarak tasarlanmıştır. Sınıflandırma problemlerinde standart bir seçimdir.
- `optimizer='adam'`: Modelin, kayıp fonksiyonunu minimize etmek için ağırlıklarını nasıl güncelleyeceğini belirleyen optimizasyon algoritmasıdır. `adam` genellikle hızlı ve verimli çalışır.
- `metrics=['accuracy']`: Eğitim ve test sırasında modelin performansını izlemek için "doğruluk" (accuracy) metriğini kullanacağımızı belirtiriz. Bu, doğru tahmin edilen resimlerin toplam resimlere oranıdır.

---

6. Modeli Eğitme

Artık modelimiz eğitime hazır. `fit` fonksiyonu ile eğitim verilerini kullanarak modelin öğrenme sürecini başlatıyoruz.

```
callbacks = EarlyStopping(monitor='val_loss',
                          patience=10,
                          verbose = 1)

history = model.fit(X_train,
          y_train,
          batch_size=10,
          epochs=50,
          validation_data=(X_test, y_test),
          callbacks=[callbacks])
```

Açıklama:

- `EarlyStopping`: Modelin performansının doygunluğa ulaştığı veya kötüleşmeye başladığı noktada eğitimi otomatik olarak durduran bir mekanizmadır. `monitor='val_loss'` ile test verisi üzerindeki kaybı izler ve `patience=10` ile 10 epoch boyunca kayıp değeri iyileşmezse eğitimi sonlandırır. Bu, zaman kazanmamızı ve aşırı öğrenmeyi (overfitting) önlememizi sağlar.
- `model.fit()`:
  - `epochs=50`: Model, tüm eğitim verisi üzerinden maksimum 50 tur dönecektir.
  - `batch_size=10`: Model, ağırlıklarını her güncellemeden önce 10 resimlik gruplara bakacaktır.
  - `validation_data=(X_test, y_test)`: Her epoch sonunda, modelin performansını daha önce hiç görmediği test verileri üzerinde ölçer. Bu, modelin genelleme yeteneğini izlemek için hayati öneme sahiptir.
- `history`: `fit` fonksiyonu, eğitim sürecindeki doğruluk ve kayıp değerlerini her epoch için bir `history` nesnesinde saklar. Bu nesneyi daha sonra eğitim sürecini görselleştirmek için kullanacağız.

---

7. Sonuçların Görselleştirilmesi

Modeli eğittikten sonra, tahminlerini görsel olarak inceleyerek ne kadar başarılı olduğunu ve nerelerde hata yaptığını analiz ediyoruz.

7.1. Doğru Tahminlerin Görselleştirilmesi

Test setindeki ilk 10 resim için modelin tahminlerini görselleştirelim.

```
test_predict = model.predict(X_test)
test_predict = np.argmax(test_predict, axis=1)

fig, axes = plt.subplots(ncols = 10, sharex = False, sharey = True, figsize = (2
0,4))
for i in range(10):
    axes[i].set_title(test_predict[i])
    axes[i].imshow(X_test[i], cmap = 'gray')
    axes[i].get_xaxis().set_visible(False)
    axes[i].get_yaxis().set_visible(False)
plt.show()
```

Açıklama:

- `model.predict(X_test)`: Test setindeki her resim için 10 elemanlı bir olasılık vektörü üretir.
- `np.argmax(test_predict, axis=1)`: Her bir olasılık vektöründe en yüksek değere sahip olan indeksi bularak (yani en olası rakamı) nihai tahmini elde eder.
- Kodun geri kalanı, Matplotlib kullanarak bu 10 resmi ve üzerlerine modelin tahminlerini yazdırır.

7.2. Yanlış Tahminlerin Analizi

Modelin hangi rakamları karıştırdığını görmek, onun zayıf yönlerini anlamamızı sağlar.

```
# Gerçek etiketleri (one-hot'tan) sayıya dönüştürme
y_test_labels = np.argmax(y_test, axis=1)

# Yanlış tahminleri bulma
```

```
wrong_predictions = np.where(test_predict != y_test_labels)[0]

# Yanlış tahmin edilen ilk 10 resmin dizinini alma
wrong_indices = wrong_predictions[:10]

# Görselleştirme
fig, axes = plt.subplots(ncols=10, nrows=1, figsize=(20, 4))
for i, index in enumerate(wrong_indices):
    axes[i].set_title(f'Tahmin: {test_predict[index]}\nGerçek: {y_test_labels[inde
x]}')
    axes[i].imshow(X_test[index], cmap='gray')
    axes[i].axis('off')
plt.show()
```

Açıklama:

Bu kod bloğu, modelin yanlış tahmin ettiği resimlerden ilk 10 tanesini bulur ve hem modelin yanlış tahminini hem de olması gereken doğru rakamı başlık olarak ekleyerek görselleştirir. Bu, örneğin modelin "4" ile "9"u veya "7" ile "1"i karıştırdığını görmemizi sağlar.

7.3. Eğitim Sürecinin Grafiği

`history` nesnesini kullanarak eğitim süresince modelin doğruluk ve kayıp değerlerinin nasıl değiştiğini çizdirebiliriz. Bu, aşırı öğrenme (overfitting) gibi durumları tespit etmek için en iyi yoldur.

```
# Grafik için verileri alma
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(len(acc))

# Doğruluk (Accuracy) ve Kayıp (Loss) grafikleri
plt.figure(figsize=(14, 6))
```

```
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Eğitim Doğruluğu')
plt.plot(epochs_range, val_acc, label='Doğrulama Doğruluğu')
plt.legend()
plt.title('Eğitim ve Doğrulama Doğruluğu')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Eğitim Kaybı')
plt.plot(epochs_range, val_loss, label='Doğrulama Kaybı')
plt.legend()
plt.title('Eğitim ve Doğrulama Kaybı')
plt.show()
```

Açıklama:

- **Doğruluk Grafiği:** İdeal durumda, hem eğitim hem de doğrulama (validation) doğruluk çizgileri birlikte yükselir ve birbirine yakın bir seviyede stabilize olur. Eğer eğitim doğruluğu artmaya devam ederken doğrulama doğruluğu düşmeye başlarsa, bu aşırı öğrenmenin bir işaretidir.
- **Kayıp Grafiği:** İdeal olarak, her iki kayıp çizgisi de birlikte azalır. Eğitim kaybı azalırken doğrulama kaybı artmaya başlarsa, bu yine aşırı öğrenmenin güçlü bir göstergesidir.