YÜKSEK DÜZEY PROGRAMLAMA DERSİ

BEYZA ZENGİN / 202013171067 / Normal Öğretim

ÖDEV:

Digit Recognizer: https://www.kaggle.com/competitions/digit-recognizer/data

Digit Recognizer veri seti, el yazısı rakamları sınıflandırmayı içeriyor. Bu tür bir problemde Convolutional Neural Network (CNN) kullandım. Bunun nedeni, CNN'lerin özellikle görüntü tabanlı problemler için tasarlanmış olmasıdır.

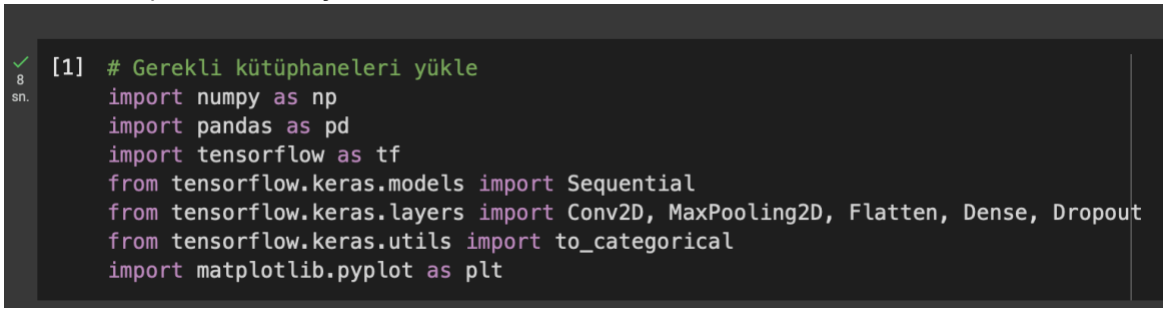
CNN, el yazısı rakam sınıflandırması gibi görüntü tabanlı problemler için en uygun yöntemlerden biridir. Çünkü:

1. Görüntüdeki uzaysal bilgiyi korur.
2. Parametre verimliliği sağlar.
3. Çeviriye ve pozisyon farklılıklarına dayanıklıdır.
4. Bu veri seti için en yüksek doğruluk oranlarına ulaşır.

PROJE KODLARI AÇIKLAMALARI:

Kaggle üzerinden veri Seti indirme işlemini, eğitim ve test kodlarını oluşturma işlemleri gibi işlemleri "Google Colab" kullanarak çalıştırdım. Colab üzerinden hızlı prototipleme yaparak farklı model deneyimleri elde ederek hızlı sonuçlar alabiliriz. CNN mimarisini kullanırken de Colab oldukça hızlı bir sonuç vermiştir.

1- Öncelikle Python ile makine öğrenimi ve derin öğrenme projelerinde sıkça kullanılan kütüphanelerimizi içeri aktaralım.



2- Bu kod, Kaggle API anahtarını (kaggle.json) yapılandırarak, Kaggle'dan veri seti indirebilmek için gerekli ortamı hazırlıyor.

3- Bu kod Kaggle üzerinden veri setini indirir ve indirilen ZIP dosyasını mevcut çalışma dizinine çıkarır.

```
[4]  !kaggle competitions download -c digit-recognizer
     !unzip digit-recognizer.zip

Downloading digit-recognizer.zip to /content
 65% 10.0M/15.3M [00:00<00:00, 100MB/s]
100% 15.3M/15.3M [00:00<00:00, 130MB/s]
Archive:  digit-recognizer.zip
  inflating: sample_submission.csv
  inflating: test.csv
  inflating: train.csv
```

4- Bu kod, **Digit Recognizer** veri setini **eğitim (train.csv)** ve **test (test.csv)** Pandas kullanarak yükler ve boyutlarını (shape) ekrana yazdırır:

```
[5]  # Veri setlerini yükle
     train_data = pd.read_csv('train.csv')
     test_data = pd.read_csv('test.csv')

     # Eğitim ve test verilerini incele
     print("Eğitim verisi şekli:", train_data.shape)
     print("Test verisi şekli:", test_data.shape)

Eğitim verisi şekli: (42000, 785)
Test verisi şekli: (28000, 784)
```

5- Bu kod, eğitim ve test verilerini işleyerek modeli eğitmek için uygun hale getirir:

1. X **(Giriş Verisi):**

• Eğitim verisindeki piksel değerlerini alır (iloc[:, 1:]), 28x28 boyutunda yeniden şekillendirir ve 1 kanal ekler (gri tonlama).

• **Normalize edilir**: Piksel değerlerini 0-255 yerine 0-1 arasına dönüştürür.

2. y **(Çıkış Verisi):**

• label sütununu alır ve **kategorik (one-hot encoded)** forma dönüştürür (ör. 3 → [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]).

3. X_test**:**

• Test verisi de 28x28 boyutunda yeniden şekillendirilir ve normalize edilir (etiketler test setinde olmadığı için yalnızca giriş verisi hazırlanır).

4. **Eğitim ve Doğrulama Seti Ayırma:**

• train_test_split kullanarak giriş ve çıkış verileri, %80 eğitim ve %20 doğrulama olarak ikiye ayrılır.

• random_state=42: Aynı sonuçları tekrar almak için sabitlenmiştir.

```
[6]  # Giriş ve çıkış verilerini ayır
     X = train_data.iloc[:, 1:].values.reshape(-1, 28, 28, 1) / 255.0  # Normalizasyon
     y = to_categorical(train_data['label'], num_classes=10)

     # Test verisini normalize et
     X_test = test_data.values.reshape(-1, 28, 28, 1) / 255.0

     # Eğitim ve doğrulama veri setlerini ayır
     from sklearn.model_selection import train_test_split
     X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

6- Bu kod, bir **Convolutional Neural Network (CNN)** modeli oluşturur ve derler:

1. **Model Yapısı (**Sequential**):**

• Katmanlar:

1. **Conv2D (32 filtre):** Görüntü özniteliklerini çıkarmak için kullanılır. 3x3 çekirdek boyutunda filtreler ve ReLU aktivasyon fonksiyonu ile çalışır.
2. **MaxPooling2D:** Boyutları küçültür ve öznitelikleri yoğunlaştırır (2x2 havuzlama).
3. **Conv2D (64 filtre):** Daha karmaşık öznitelikleri öğrenir.
4. **MaxPooling2D:** Boyutları tekrar küçültür.
5. **Flatten:** Çok boyutlu tensörü tek boyutlu vektöre dönüştürür.
6. **Dense (128 nöron):** Tam bağlantılı katman, sınıflandırmaya hazırlık yapar (ReLU aktivasyonu).
7. **Dropout (0.5):** Aşırı öğrenmeyi önlemek için %50 oranında nöronları rastgele devre dışı bırakır.
8. **Dense (10 nöron):** Çıkış katmanı, 10 sınıf (0-9 rakamları) için olasılık hesaplar (Softmax aktivasyonu).

2. **Model Derleme (**compile**):**

• optimizer='adam': Adam optimizasyon algoritması kullanılır (hızlı ve etkili).

• loss='categorical_crossentropy': Çok sınıflı sınıflandırma için uygun kayıp fonksiyonu.

• metrics=['accuracy']: Performansı doğruluk ile ölçer.

• model.summary()**:** Modelin katmanlarını, giriş-çıkış şekillerini ve parametre sayılarını özetler.

Sonuç: **El yazısı rakam sınıflandırması** için temel bir CNN modeli oluşturur ve eğitilmeye hazır hale getirir.

```python
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

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

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_3 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_4 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_4 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_1 (Flatten)	(None, 1600)	0
dense_3 (Dense)	(None, 128)	204,928
dropout_2 (Dropout)	(None, 128)	0
dense_4 (Dense)	(None, 10)	1,290

```
Total params: 225,034 (879.04 KB)
Trainable params: 225,034 (879.04 KB)
Non-trainable params: 0 (0.00 B)
```

7- Bu kod, modelin eğitim sürecini başlatır ve aşağıdaki işlemleri yapar:

1. model.fit() **Fonksiyonu:**
• Modeli, eğitim verileri (X_train, y_train) üzerinde eğitir ve doğrulama verileri (X_val, y_val) ile test eder.

2. **Parametreler:**
• X_train**,** y_train**:** Eğitim verisi (giriş ve etiketler). Model bu verilerle öğrenir.
• validation_data=(X_val, y_val)**:** Doğrulama seti. Eğitim sırasında doğrulama verisinin performansı izlenir.
• epochs=10**:** Eğitim verisinin tamamı üzerinde 10 kez geçiş yapılır.
• batch_size=64**:** Veriler, her adımda 64'er örneklik parçalar halinde modele gönderilir.

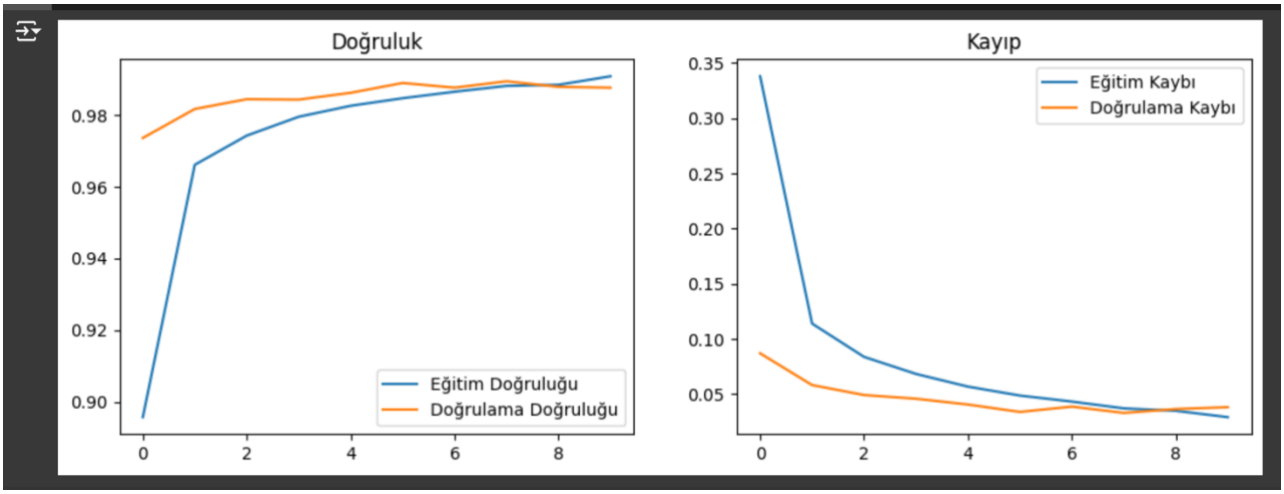
3. **Sonuç:**
• Eğitim süreci boyunca **doğruluk (accuracy)** ve **kayıp (loss)** gibi metrikler kaydedilir.
• history değişkeni, eğitim ve doğrulama metriklerini içerir (ör. history.history['accuracy']).

```
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=10, batch_size=64)

Epoch 1/10
525/525 ━━━━━━━━━━━━━━━━━━━━ 34s 62ms/step - accuracy: 0.7853 - loss: 0.6762 - val_accuracy: 0.9737 - val_loss: 0.0870
Epoch 2/10
525/525 ━━━━━━━━━━━━━━━━━━━━ 32s 62ms/step - accuracy: 0.9623 - loss: 0.1280 - val_accuracy: 0.9818 - val_loss: 0.0582
Epoch 3/10
525/525 ━━━━━━━━━━━━━━━━━━━━ 30s 57ms/step - accuracy: 0.9752 - loss: 0.0849 - val_accuracy: 0.9845 - val_loss: 0.0492
Epoch 4/10
525/525 ━━━━━━━━━━━━━━━━━━━━ 31s 59ms/step - accuracy: 0.9793 - loss: 0.0663 - val_accuracy: 0.9844 - val_loss: 0.0459
Epoch 5/10
525/525 ━━━━━━━━━━━━━━━━━━━━ 39s 56ms/step - accuracy: 0.9822 - loss: 0.0566 - val_accuracy: 0.9863 - val_loss: 0.0406
Epoch 6/10
525/525 ━━━━━━━━━━━━━━━━━━━━ 41s 57ms/step - accuracy: 0.9846 - loss: 0.0487 - val_accuracy: 0.9890 - val_loss: 0.0339
Epoch 7/10
525/525 ━━━━━━━━━━━━━━━━━━━━ 30s 57ms/step - accuracy: 0.9868 - loss: 0.0441 - val_accuracy: 0.9877 - val_loss: 0.0387
Epoch 8/10
525/525 ━━━━━━━━━━━━━━━━━━━━ 45s 66ms/step - accuracy: 0.9887 - loss: 0.0349 - val_accuracy: 0.9895 - val_loss: 0.0330
Epoch 9/10
525/525 ━━━━━━━━━━━━━━━━━━━━ 38s 61ms/step - accuracy: 0.9880 - loss: 0.0338 - val_accuracy: 0.9880 - val_loss: 0.0365
Epoch 10/10
525/525 ━━━━━━━━━━━━━━━━━━━━ 39s 57ms/step - accuracy: 0.9910 - loss: 0.0289 - val_accuracy: 0.9877 - val_loss: 0.0382
```

8- Bu kod, modelin eğitim ve doğrulama performansını görselleştirmek için doğruluk ve kayıp grafikleri oluşturur.

1. plt.figure(figsize=(12, 4))**:**
• Grafiklerin boyutunu belirler (genişlik 12, yükseklik 4 birim).
2. **1. Grafik - Doğruluk (Accuracy):**
• plt.subplot(1, 2, 1): Grafiği 1 satır ve 2 sütun halinde düzenler; bu, ilk grafiktir.
• plt.plot(history.history['accuracy'], label='Eğitim Doğruluğu'): Eğitim doğruluğunu çizer.
• plt.plot(history.history['val_accuracy'], label='Doğrulama Doğruluğu'): Doğrulama doğruluğunu çizer.
• plt.legend(): Çizgiler için bir açıklama ekler (Eğitim ve Doğrulama Doğruluğu).
• plt.title('Doğruluk'): Grafiğe başlık ekler.
3. **2. Grafik - Kayıp (Loss):**
• plt.subplot(1, 2, 2): Bu, ikinci grafiktir.
• plt.plot(history.history['loss'], label='Eğitim Kaybı'): Eğitim kaybını çizer.
• plt.plot(history.history['val_loss'], label='Doğrulama Kaybı'): Doğrulama kaybını çizer.
• plt.legend(): Çizgiler için bir açıklama ekler (Eğitim ve Doğrulama Kaybı).
• plt.title('Kayıp'): Grafiğe başlık ekler.

```python
# Doğruluk ve kayıp grafikleri
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Eğitim Doğruluğu')
plt.plot(history.history['val_accuracy'], label='Doğrulama Doğruluğu')
plt.legend()
plt.title('Doğruluk')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Eğitim Kaybı')
plt.plot(history.history['val_loss'], label='Doğrulama Kaybı')
plt.legend()
plt.title('Kayıp')
plt.show()
```



9- 1. **Tahminlerin Hesaplanması:**

• predictions = model.predict(X_test): Test verisi (X_test) üzerinde modelin yaptığı tahminleri hesaplar.

• Her bir örnek için 10 sınıfın (0-9) olasılıklarını döndürür.

• predicted_labels = np.argmax(predictions, axis=1):

• En yüksek olasılığa sahip sınıfı seçerek her görüntünün tahmin edilen etiketini alır.

2. **Görselleştirme Ayarları:**

• plt.figure(figsize=(10, 5)): Görseller için 10 birim genişlik ve 5 birim yükseklikte bir figür oluşturur.

• for i in range(10):: İlk 10 test örneği için döngü kurar.

• plt.subplot(2, 5, i+1): 2 satır ve 5 sütun şeklinde bir matris düzeninde alt grafik oluşturur.

3. **Görsellerin Gösterilmesi:**

• plt.imshow(X_test[i].reshape(28, 28), cmap='gray'): İlk 10 test örneğinin 28x28 boyutundaki görüntüsünü gri tonlamalı olarak çizer.

• plt.title(f"Tahmin: {predicted_labels[i]}"): Görüntünün üstüne modelin tahmin ettiği sınıfı yazdırır.

• plt.axis('off'): X ve Y eksenlerini kaldırır.

4. **Düzenleme ve Gösterim:**

• plt.tight_layout(): Alt grafiklerin birbirine sıkışmasını engeller.

• plt.show(): Tüm grafikleri gösterir.

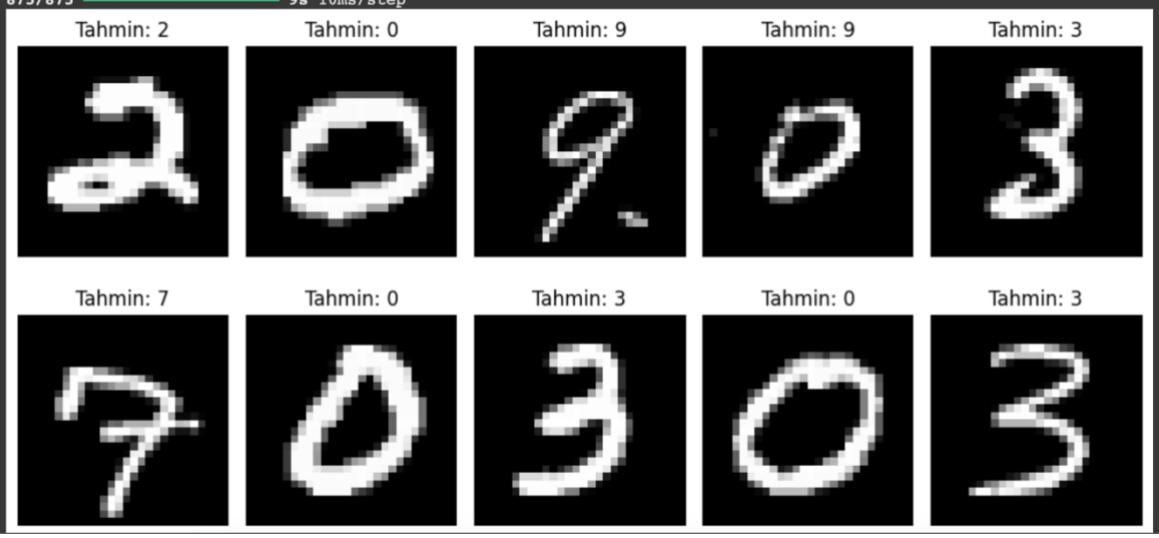
10- **Sonuç:** Bu kod:

•İlk 10 test örneğini ve modelin bu örnekler için tahmin ettiği etiketleri görselleştirir.

•Modelin doğru tahmin yapıp yapmadığını hızlıca analiz etmek için faydalıdır.

```python
# Test verisi için tahmin
predictions = model.predict(X_test)
predicted_labels = np.argmax(predictions, axis=1)

# İlk birkaç tahmini göster
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Tahmin: {predicted_labels[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

```
875/875 ━━━━━━━━━━━━━━━━━━━━ 9s 10ms/step
```



*** Bu kodlar, el yazısı rakamlarını sınıflandırmak için bir **Convolutional Neural Network (CNN)** modeli oluşturup eğitmek, model performansını değerlendirmek ve test verisi üzerinde tahminler yapmak için kullanılan bir makine öğrenimi sürecini baştan sona kapsamaktadır. Eğitim doğruluğu, kaybı ve tahmin görselleştirmeleri gibi analizlerle modelin genelleme kabiliyeti detaylı şekilde incelenmektedir. ***