YÜKSEK DÜZEY PROGRAMLAMA DERSİ

İbrahim Emre Baydur

202213172029 NÖ

Digit Recognizer, genellikle **el yazısı ile yazılmış rakamları tanıma** problemiyle ilgili bir uygulamadır. Bu, makine öğrenimi ve özellikle görüntü işleme alanında sıkça karşılaşılan bir görevdir. Amaç, bir görselde yer alan el yazısıyla yazılmış bir rakamı (0-9 arasında) otomatik olarak tanımak ve doğru bir şekilde sınıflandırmaktır.

Kullanım Alanları

- **Otomatik Form Okuma:** Bankacılık veya sınav kağıtlarında el yazısı rakamların tanınması.
- **Trafik İşareti Tanıma:** Araç plakalarındaki rakamların tanımlanması.
- **Mobil Uygulamalar:** El yazısını dijital metne dönüştüren uygulamalarda rakamları tanıma.

Proje kodları :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.utils import to_categorical
print("Tüm kütüphaneler başarıyla yüklendi!")
```

```
Tüm kütüphaneler başarıyla yüklendi!
```

Gerekli kütüphaneler (ör. TensorFlow, NumPy, Pandas) projeye dahil edildi.

```
[8]: import pandas as pd

     # CSV dosyasını yükle
     data = pd.read_csv("train.csv")  # Dosya adını uygun şekilde değiştir

     # İlk birkaç satırı gör
     print(data.head())

     # Veri setinin genel bilgilerini kontrol et
     print(data.info())
```

```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0
1      0       0       0       0       0       0       0       0       0
2      1       0       0       0       0       0       0       0       0
3      4       0       0       0       0       0       0       0       0
4      0       0       0       0       0       0       0       0       0

   pixel8  ...  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  ...         0         0         0         0         0         0
1       0  ...         0         0         0         0         0         0
2       0  ...         0         0         0         0         0         0
3       0  ...         0         0         0         0         0         0
4       0  ...         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
None
```

Bir CSV dosyası yüklendi, ardından veri setinin ilk 5 satırı incelendi. Ayrıca veri setindeki sütun adları, toplam veri sayısı, veri türleri ve bellek kullanımı gibi özet bilgiler görüntülendi.

[21]:
```
from tensorflow.keras.utils import to_categorical

# Etiketleri kategorik hale getir
y = to_categorical(y, num_classes=10)

print(f"Veri şekli: {X.shape}")
print(f"Etiket şekli: {y.shape}")
```

```
Veri şekli: (42000, 28, 28, 1)
Etiket şekli: (42000, 10)
```

[23]:
```
from sklearn.model_selection import train_test_split

# Veriyi eğitim ve test setine böl
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"Eğitim verisi şekli: {X_train.shape}")
print(f"Test verisi şekli: {X_test.shape}")
```

```
Eğitim verisi şekli: (33600, 28, 28, 1)
Test verisi şekli: (8400, 28, 28, 1)
```

[27]:
```
from tensorflow.keras.layers import Input

model = Sequential([
    Input(shape=(28, 28, 1)),  # İlk katmanda Input nesnesi kullanıyoruz
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
```

[29]:
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Eğitim ve test verileri oluşturuldu, ardından bir CNN modeli tasarlandı ve derleme işlemi tamamlandı.

[31]:
```
history = model.fit(X_train, y_train,
                    epochs=10,  # 10 epoch boyunca eğit
                    validation_data=(X_test, y_test),
                    batch_size=32)  # Her iterasyonda 32 örnek kullan
```

```
Epoch 1/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.8771 - loss: 0.4018 - val_accuracy: 0.9812 - val_loss: 0.0604
Epoch 2/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9817 - loss: 0.0550 - val_accuracy: 0.9854 - val_loss: 0.0490
Epoch 3/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9896 - loss: 0.0335 - val_accuracy: 0.9835 - val_loss: 0.0518
Epoch 4/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9909 - loss: 0.0275 - val_accuracy: 0.9868 - val_loss: 0.0443
Epoch 5/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9953 - loss: 0.0163 - val_accuracy: 0.9855 - val_loss: 0.0473
Epoch 6/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9957 - loss: 0.0143 - val_accuracy: 0.9886 - val_loss: 0.0403
Epoch 7/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9965 - loss: 0.0104 - val_accuracy: 0.9865 - val_loss: 0.0452
Epoch 8/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9974 - loss: 0.0083 - val_accuracy: 0.9870 - val_loss: 0.0531
Epoch 9/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9972 - loss: 0.0075 - val_accuracy: 0.9911 - val_loss: 0.0360
Epoch 10/10
1050/1050 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9973 - loss: 0.0086 - val_accuracy: 0.9882 - val_loss: 0.0456
```

[33]:
```
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test doğruluk oranı: {test_acc:.2f}")
```

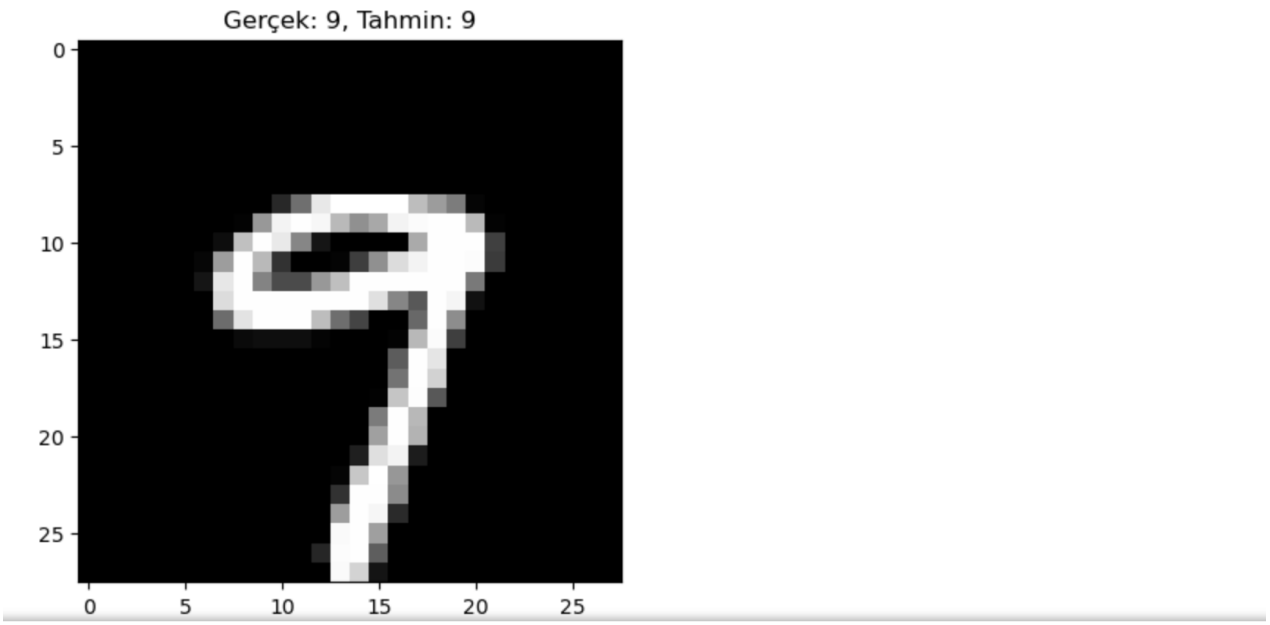
```
263/263 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9885 - loss: 0.0521
Test doğruluk oranı: 0.99
```

Model başarıyla eğitildi, test verileriyle değerlendirildi ve performans sonuçları görselleştirildi.

```python
import matplotlib.pyplot as plt

# Eğitim ve doğrulama doğruluk oranları
plt.plot(history.history['accuracy'], label='Eğitim Doğruluğu')
plt.plot(history.history['val_accuracy'], label='Doğrulama Doğruluğu')
plt.xlabel('Epoch')
plt.ylabel('Doğruluk')
plt.legend()
plt.show()

# Eğitim ve doğrulama kayıpları
plt.plot(history.history['loss'], label='Eğitim Kaybı')
plt.plot(history.history['val_loss'], label='Doğrulama Kaybı')
plt.xlabel('Epoch')
plt.ylabel('Kayıp')
plt.legend()
plt.show()
```

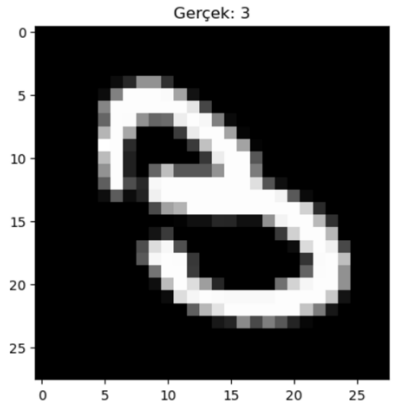
Eğitim ve doğrulama süreçlerine ait doğruluk ve kayıp değerleri grafiklerle analiz edilerek görselleştirildi.

```python
import numpy as np

# Test setinden bazı tahminler
predictions = model.predict(X_test)

# İlk 5 örneği görselleştir
for i in range(5):
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Gerçek: {np.argmax(y_test[i])}, Tahmin: {np.argmax(predictions[i])}")
    plt.show()
```



Bu projede, el yazısı ile yazılmış rakamları tanıyabilen bir sinir ağı modeli geliştirdim. Modelin performansını test verileri üzerinde değerlendirerek doğruluğunu ölçtüm. Ayrıca, rastgele seçilen bir test örneği için modelin tahminini gerçek değerle karşılaştırarak bunu görselleştirdim.

```python
# Veriyi normalize et ve yeniden şekillendir
X_train = X_train.reshape(-1, 28, 28, 1) / 255.0
X_test = X_test.reshape(-1, 28, 28, 1) / 255.0

# Test setinde etiketleri 3 olanları bul
three_indices = np.where(y_test == 3)[0]  # y_test 1D olduğu için doğrudan == kullanabilirsiniz

# İlk 3 örneğini seç
index = three_indices[0]

# Görüntüyü çizdir
plt.imshow(X_test[index].reshape(28, 28), cmap='gray')
plt.title("Gerçek: 3")
plt.show()

# Tahmin yap
prediction = model.predict(X_test[index].reshape(1, 28, 28, 1))
print(f"Modelin Tahmini: {np.argmax(prediction)}")
```



Bu kodda, test setinden "3" rakamına ait bir örnek seçilmiş, görselleştirilmiş ve modelin tahmini gerçek değerle karşılaştırılmıştır.