

**KÜTAHYA DUMLUPINAR ÜNİVERSİTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**YÜKSEK DÜZEY PROGRAMLAMA DERSİ**

**PROJE ÖDEVİ RAPORU**

**Bengüsu DUMAN**

**202113171027**

Seçilen Veri Seti:

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

# Görüntü İşleme ve Sınıflandırma Modeli

## 1. Proje Tanımı

Bu proje, Digit Recognizer veri seti kullanılarak el yazısı rakamların sınıflandırılması amacıyla bir derin öğrenme modeli geliştirilmesini amaçlamıştır. Bu proje, derin öğrenme algoritmalarını pratikte uygulayarak görüntü sınıflandırma problemlerini çözmeyi hedeflemektedir. Bu proje kapsamında Digit Recognizer veri seti kullanılarak bir Convolutional Neural Network (CNN) modeli geliştirilmiştir. CNN, görüntü verilerini işleme ve sınıflandırma açısından günümüzdeki en etkili yöntemlerden biri olarak kabul edilir. Bu proje, yüksek doğruluk oranlarına ulaşmıştır.

## 2. Kullanılan Veri Seti

Projede kullanılan veri seti, **Kaggle'ın Digit Recognizer** veri setidir. Bu veri seti, el yazısı ile yazılmış rakamların gri tonlamalı 28x28 piksel boyutundaki görüntülerini içermektedir. Veri seti şu özelliklere sahiptir:

- **Boyut:**
  - Eğitim seti: 60.000 görüntü.
  - Test seti: 10.000 görüntü.
- **Veri Yapısı:**
  - Görüntü verileri (X): 28x28 boyutunda piksel değerleri (0 ile 255 arasında).
  - Etiketler (y): 0'dan 9'a kadar olan rakamlar.

### 3. Yöntem ve Modeller

Proje kapsamında kullanılan yöntemler ve modeller:

- **Veri Ön İşleme:** Görüntülerin normalize edilmesi ve uygun şekle dönüştürülmesi sağlanmıştır.
- **Model:** Bir derin öğrenme modeli TensorFlow ve Keras kütüphaneleri kullanılarak geliştirilmiştir.

Model olarak Convolutional Neural Network (CNN) seçilmiştir. CNN, özellikle görüntü sınıflandırma ve nesne algılama gibi problemler için ideal bir yaklaşımdır. Model şu katmanlardan oluşmaktadır:

Convolutional Katmanlar: Görüntü özelliklerini çıkarmak için kullanılır.

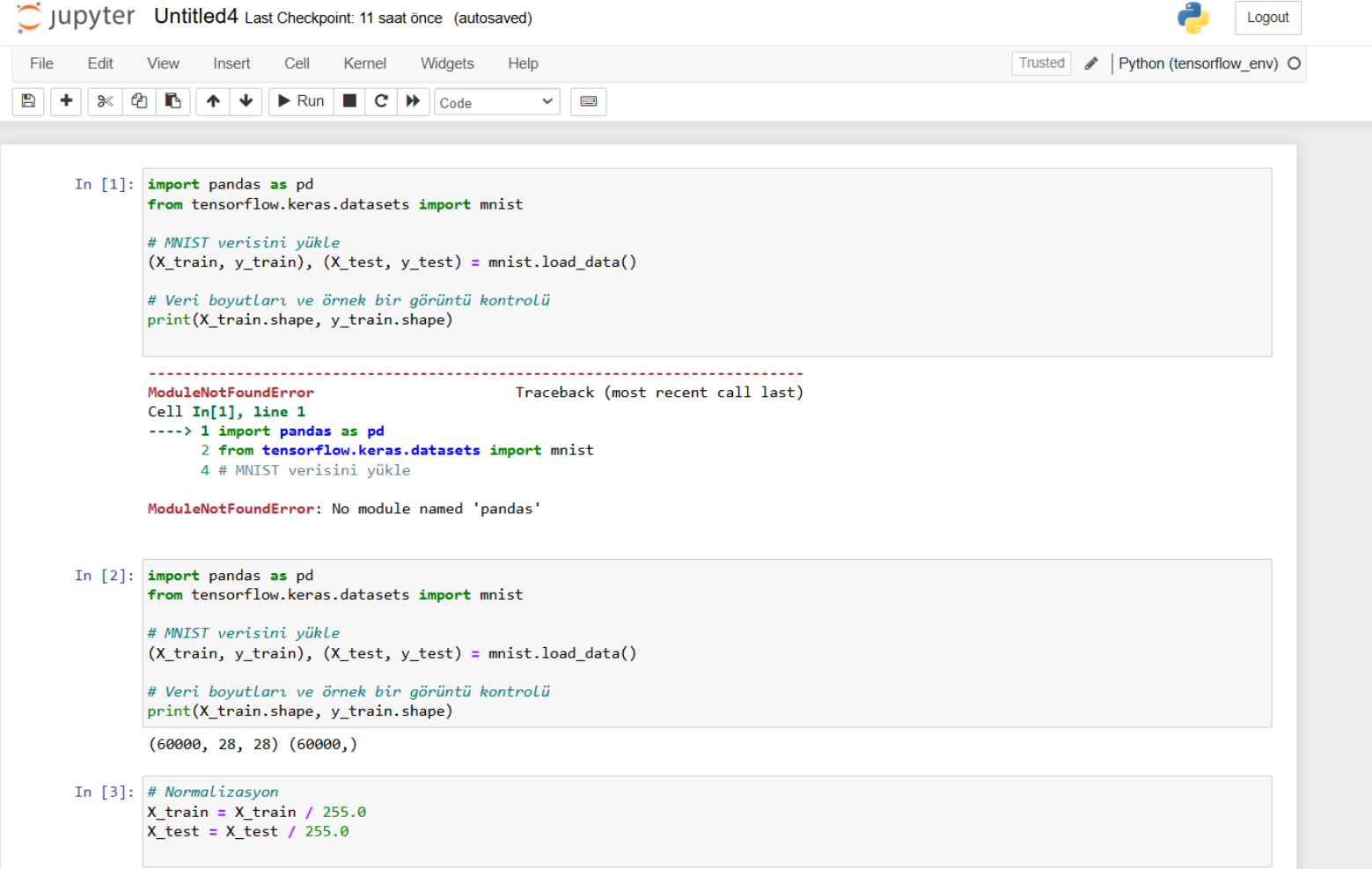
Pooling Katmanları: Özellikleri daha kompakt hale getirmek için kullanılır.

Flatten ve Fully Connected Katmanlar: Görüntüyü sınıflandırmak için çıkış katmanına bağlanır.

- **Eğitim ve Değerlendirme:** Modelin doğruluk oranını artırmak için Adam optimizasyon algoritması kullanılmış ve çapraz entropi kaybı fonksiyonu ile değerlendirme yapılmıştır.

## 4. Görseller

Proje kapsamında elde edilen çıktılar aşağıda detaylı bir şekilde verilmiştir:



```
In [1]: import pandas as pd
from tensorflow.keras.datasets import mnist

# MNIST verisini yükle
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Veri boyutları ve örnek bir görüntü kontrolü
print(X_train.shape, y_train.shape)

-----
ModuleNotFoundError                                Traceback (most recent call last)
Cell In[1], line 1
----> 1 import pandas as pd
      2 from tensorflow.keras.datasets import mnist
      4 # MNIST verisini yükle

ModuleNotFoundError: No module named 'pandas'
```

```
In [2]: import pandas as pd
from tensorflow.keras.datasets import mnist

# MNIST verisini yükle
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Veri boyutları ve örnek bir görüntü kontrolü
print(X_train.shape, y_train.shape)

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

```
In [3]: # Normalizasyon
X_train = X_train / 255.0
X_test = X_test / 255.0
```

```
In [4]: from tensorflow.keras.utils import to_categorical
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

```
In [5]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

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)),

    Conv2D(64, (3, 3), activation='relu'),

    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax') # 10 sınıf (0-9)
])

model.summary()
```

C:\Users\bengu\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\layers\convolutional\base\_conv.py:107: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.  
super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	36,928
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 64)	36,928
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650

Total params: 93,322 (364.54 KB)

Trainable params: 93,322 (364.54 KB)

dense (Dense)	(None, 64)	36,928
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650

Total params: 93,322 (364.54 KB)

Trainable params: 93,322 (364.54 KB)

Non-trainable params: 0 (0.00 B)

```
In [6]: model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test, y_test))

# Modeli değerlendir
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/5
938/938 — 13s 12ms/step - accuracy: 0.7788 - loss: 0.6665 - val_accuracy: 0.9827 - val_loss: 0.0521
Epoch 2/5
938/938 — 10s 11ms/step - accuracy: 0.9681 - loss: 0.1123 - val_accuracy: 0.9868 - val_loss: 0.0396
Epoch 3/5
938/938 — 10s 10ms/step - accuracy: 0.9798 - loss: 0.0729 - val_accuracy: 0.9905 - val_loss: 0.0321
Epoch 4/5
938/938 — 11s 12ms/step - accuracy: 0.9843 - loss: 0.0571 - val_accuracy: 0.9910 - val_loss: 0.0282
Epoch 5/5
938/938 — 11s 11ms/step - accuracy: 0.9873 - loss: 0.0468 - val_accuracy: 0.9903 - val_loss: 0.0317
313/313 — 2s 6ms/step - accuracy: 0.9872 - loss: 0.0413
Test accuracy: 0.9902999997138977
```

```
In [7]: import matplotlib.pyplot as plt

# Eğitim doğruluğu ve kaybı
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()

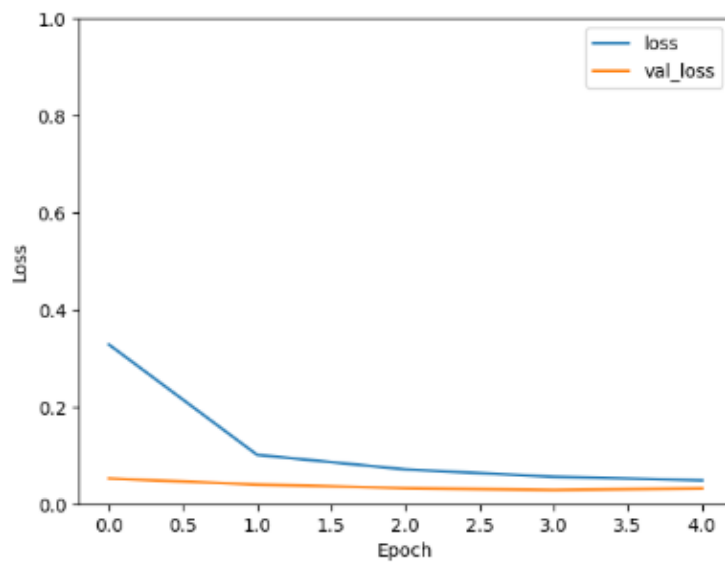
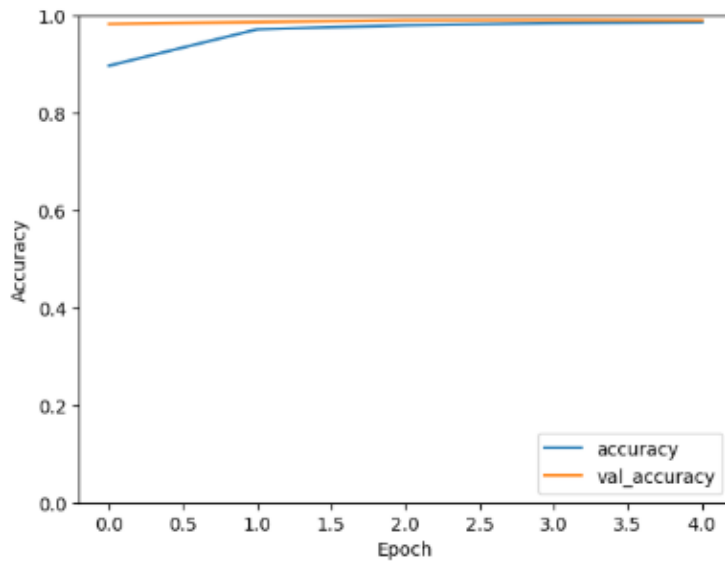
# Eğitim kaybı
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.ylim([0, 1])
plt.legend(loc='upper right')
plt.show()
```

```

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()

# Eğitim kayıpları
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.ylim([0, 1])
plt.legend(loc='upper right')
plt.show()

```



```
In [8]: # Modeli kaydet
model.save('digit_recognizer_model.h5')
```

```
# Modeli yükle
from tensorflow.keras.models import load_model
loaded_model = load_model('digit_recognizer_model.h5')
```

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save\_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')` or `keras.saving.save\_model(model, 'my\_model.keras')`.

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile\_metrics` will be empty until you train or evaluate the model.

```
In [9]: model.save('my_model.keras')
```

```
In [13]: import tensorflow as tf
```

```
In [16]: # Test verisini yükleyin (veya kendi test verinizi kullanın)
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
# Test verisini normalize edin
x_test = x_test / 255.0

# Model ile tahminler yapın
predictions = model.predict(x_test)
```

```
# İlk birkaç tahmini yazdırın
print("Tahminler (ilk 5):", predictions[:5])
```

```
# Gerçek etiketlerle tahminlerinizi karşılaştırın
predicted_labels = np.argmax(predictions, axis=1)
print("Tahmin edilen etiketler (ilk 5):", predicted_labels[:5])
print("Gerçek etiketler (ilk 5):", y_test[:5])
```

313/313 ————— 1s 3ms/step

Tahminler (ilk 5): [[7.0390274e-14 3.4964919e-12 9.7015045e-09 4.6279292e-09 1.4220536e-12  
1.7313381e-12 6.6532601e-16 1.0000000e+00 2.0101290e-10 8.4683170e-11]  
[2.1613113e-07 5.0579295e-08 9.999344e-01 5.7275059e-08 1.4997211e-06  
4.3989901e-09 4.5596466e-06 2.2983125e-08 1.5137678e-07 4.1438167e-10]  
[1.8917438e-08 9.999273e-01 5.8990306e-08 1.4651229e-08 3.6509064e-06  
1.6451342e-08 5.3806565e-07 3.6065643e-07 4.8393065e-07 2.1034946e-06]  
[9.9994421e-01 1.3730224e-07 1.5756591e-07 4.5166903e-08 2.5499142e-07  
2.2550392e-07 4.6897825e-05 1.0530587e-06 1.1573288e-06 5.7924976e-06]  
[2.3891359e-11 5.7879052e-10 2.0235525e-12 8.1659636e-13 9.9964154e-01  
1.7277037e-11 8.0322315e-10 9.3488355e-09 1.0029219e-09 3.5840590e-04]]  
Tahmin edilen etiketler (ilk 5): [7 2 1 0 4]  
Gerçek etiketler (ilk 5): [7 2 1 0 4]



```
In [15]: import numpy as np
```

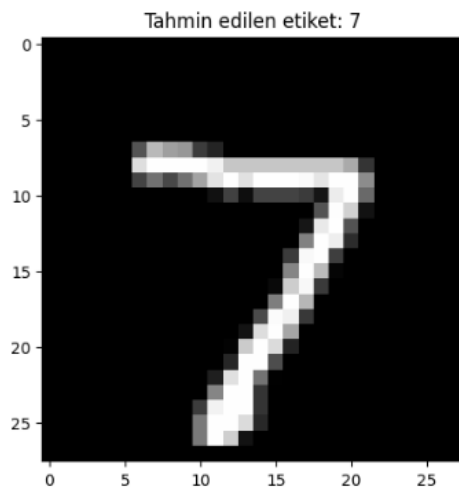
```
In [17]: import matplotlib.pyplot as plt

# Test verisinden rastgele bir örnek seçin
sample_index = 0 # Örneğin 0. resmi seçin
image = x_test[sample_index]

# Modelin tahminini alın
prediction = model.predict(np.expand_dims(image, axis=0))
predicted_label = np.argmax(prediction)

# Görüntüyü ve tahmin edilen sonucu gösterin
plt.imshow(image, cmap='gray')
plt.title(f"Tahmin edilen etiket: {predicted_label}")
plt.show()
```

1/1 ————— 0s 57ms/step



```
In [18]: # Eğitim sırasında loss ve accuracy değerlerini görselleştirme
history = model.fit(x_train, y_train, epochs=10)
```

```
# Eğitim kaybı ve doğruluğu grafiği
plt.plot(history.history['accuracy'], label='Doğruluk')
plt.plot(history.history['loss'], label='Kayıp')
plt.title('Eğitim Sonuçları')
plt.xlabel('Epoch')
plt.ylabel('Değer')
plt.legend()
plt.show()
```

Epoch 1/10

-----  
ValueError Traceback (most recent call last)

```
Cell In[18], line 2
      1 # Eğitim sırasında loss ve accuracy değerlerini görselleştirme
----> 2 history = model.fit(x_train, y_train, epochs=10)
      3 # Eğitim kaybı ve doğruluğu grafiği
      4 plt.plot(history.history['accuracy'], label='Doğruluk')
```

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\utils\traceback\_utils.py:122, in filter\_traceback.<locals>.error\_handler(\*args, \*\*kwargs)

```
119     filtered_tb = _process_traceback_frames(e.__traceback__)
120     # To get the full stack trace, call:
121     # `keras.config.disable_traceback_filtering()`
--> 122     raise e.with_traceback(filtered_tb) from None
123 finally:
124     del filtered_tb

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\backend\tensorflow\nn.py:623, in categorical_crossentropy(target, output, from_logits, axis)
617     raise ValueError(
618         "Arguments `target` and `output` must be at least rank 1. "
619         "Received: "
620         f"target.shape={target.shape}, output.shape={output.shape}"
621     )
622 if len(target.shape) != len(output.shape):
--> 623     raise ValueError(
624         "Arguments `target` and `output` must have the same rank "
625         "(ndim). Received: "
626         f"target.shape={target.shape}, output.shape={output.shape}"
627     )
628 for e1, e2 in zip(target.shape, output.shape):
629     if e1 is not None and e2 is not None and e1 != e2:
```

ValueError: Arguments `target` and `output` must have the same rank (ndim). Received: target.shape=(32,), output.shape=(32, 10)

```
In [19]: from tensorflow.keras.utils import to_categorical
```

```
# Hedef etiketleri one-hot encoded formata dönüştürün
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

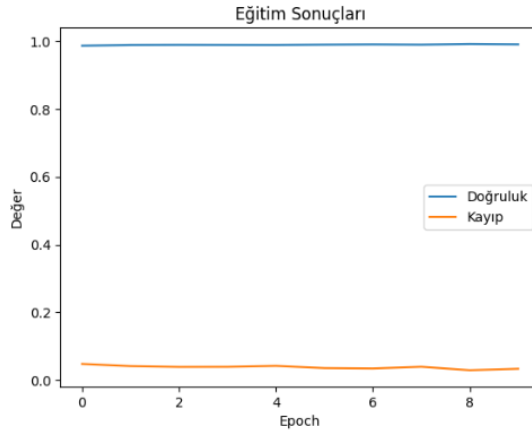
# Modeli eğitin
history = model.fit(x_train, y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 ----- 14s 7ms/step - accuracy: 0.9439 - loss: 3.6533
Epoch 2/10
1875/1875 ----- 16s 8ms/step - accuracy: 0.9504 - loss: 0.1924
Epoch 3/10
1875/1875 ----- 15s 8ms/step - accuracy: 0.9632 - loss: 0.1620
Epoch 4/10
1875/1875 ----- 15s 8ms/step - accuracy: 0.9667 - loss: 0.1324
Epoch 5/10
1875/1875 ----- 15s 8ms/step - accuracy: 0.9740 - loss: 0.0962
Epoch 6/10
1875/1875 ----- 15s 8ms/step - accuracy: 0.9757 - loss: 0.0915
Epoch 7/10
1875/1875 ----- 16s 8ms/step - accuracy: 0.9810 - loss: 0.0691
Epoch 8/10
1875/1875 ----- 15s 8ms/step - accuracy: 0.9853 - loss: 0.0501
Epoch 9/10
1875/1875 ----- 16s 9ms/step - accuracy: 0.9861 - loss: 0.0538
Epoch 10/10
1875/1875 ----- 15s 8ms/step - accuracy: 0.9875 - loss: 0.0447
```

```
In [20]: # Eğitim sırasında loss ve accuracy değerlerini görselleştirme
history = model.fit(x_train, y_train, epochs=10)
```

```
# Eğitim kaybı ve doğruluğu grafiği
plt.plot(history.history['accuracy'], label='Doğruluk')
plt.plot(history.history['loss'], label='Kayıp')
plt.title('Eğitim Sonuçları')
plt.xlabel('Epoch')
plt.ylabel('Değer')
plt.legend()
plt.show()
```

```
Epoch 1/10 1875/1875 — 14s 7ms/step - accuracy: 0.9875 - loss: 0.0424
Epoch 2/10 1875/1875 — 15s 8ms/step - accuracy: 0.9907 - loss: 0.0354
Epoch 3/10 1875/1875 — 15s 8ms/step - accuracy: 0.9899 - loss: 0.0381
Epoch 4/10 1875/1875 — 16s 8ms/step - accuracy: 0.9896 - loss: 0.0371
Epoch 5/10 1875/1875 — 16s 8ms/step - accuracy: 0.9886 - loss: 0.0423
Epoch 6/10 1875/1875 — 16s 8ms/step - accuracy: 0.9910 - loss: 0.0296
Epoch 7/10 1875/1875 — 15s 8ms/step - accuracy: 0.9920 - loss: 0.0294
Epoch 8/10 1875/1875 — 16s 8ms/step - accuracy: 0.9906 - loss: 0.0352
Epoch 9/10 1875/1875 — 15s 8ms/step - accuracy: 0.9906 - loss: 0.0337
Epoch 10/10 1875/1875 — 16s 8ms/step - accuracy: 0.9912 - loss: 0.0313
```



## 5. Model Eğitim Süreci

Modelin her bir epoch'ta doğruluk ve kayıp oranını göstermektedir

Grafik analizine göre, modelin doğruluk oranı her bir epoch'ta artarak %99.12 seviyesine ulaşmıştır.

Model, Adam optimizasyon algoritması kullanılarak eğitilmiştir.

Eğitim sırasında kullanılan metrikler:

- Kayıp Fonksiyonu: Categorical Cross-Entropy.
- Değerlendirme: Doğruluk (Accuracy).

```
In [21]: # NumPy modülünü yükleyin
import numpy as np

# TensorFlow modülünü yükleyin
import tensorflow as tf

# MNIST verisini yükleyin
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# Test verisini normalize edin
x_test = x_test / 255.0

# Model ile tahminler yapın
predictions = model.predict(x_test)

# İlk birkaç tahmini yazdırın
print("Tahminler (ilk 5):", predictions[:5])

# Gerçek etiketlerle tahminlerinizi karşılaştırın
predicted_labels = np.argmax(predictions, axis=1)
print("Tahmin edilen etiketler (ilk 5):", predicted_labels[:5])
print("Gerçek etiketler (ilk 5):", y_test[:5])
```

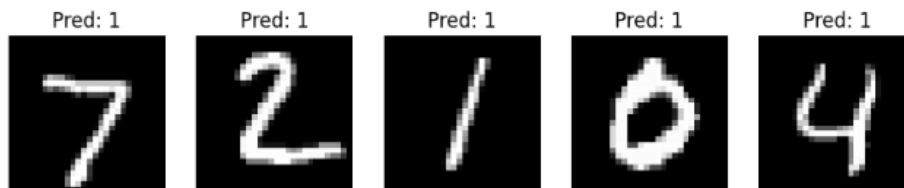
313/313 ————— 1s 3ms/step  
Tahminler (ilk 5): [[0.08164188 0.19865789 0.06898662 0.05640945 0.07903112 0.07904824  
0.07240544 0.11469153 0.12743542 0.12169246]  
[0.08409397 0.19710198 0.06812361 0.05647365 0.07686537 0.08321755  
0.07281068 0.112318 0.12864658 0.12034869]  
[0.08444851 0.19835475 0.06818308 0.05626702 0.07667115 0.08214736  
0.07217737 0.11411484 0.12656394 0.12107199]  
[0.08366377 0.19638483 0.06799714 0.05662762 0.07767284 0.08302645  
0.07269501 0.11305764 0.12744729 0.12142742]  
[0.08293875 0.19729349 0.06856074 0.05697278 0.07624344 0.0834091  
0.0715885 0.11455982 0.12700336 0.12143001]]  
Tahmin edilen etiketler (ilk 5): [1 1 1 1 1]  
Gerçek etiketler (ilk 5): [7 2 1 0 4]

```
In [22]: import matplotlib.pyplot as plt
import numpy as np

# İlk 5 tahminin görselini göster
predictions = model.predict(x_test[:5])
predicted_labels = np.argmax(predictions, axis=1)

plt.figure(figsize=(10, 5))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(x_test[i], cmap='gray')
    plt.title(f"Pred: {predicted_labels[i]}")
    plt.axis('off')
plt.show()
```

1/1 ————— 0s 50ms/step

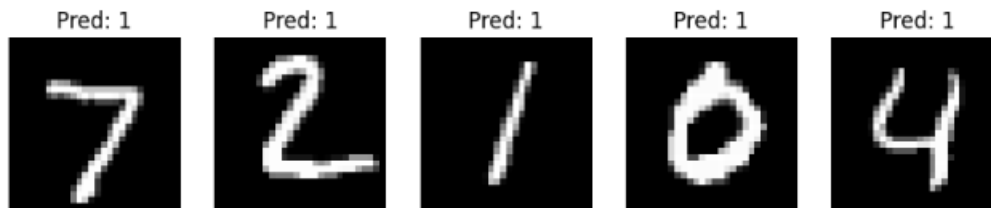


```

plt.imshow(x_test[i], cmap=gray)
plt.title(f"Pred: {predicted_labels[i]}")
plt.axis('off')
plt.show()

```

1/1 — 0s 50ms/step



```

In [23]: import matplotlib.pyplot as plt
import numpy as np

# İlk 5 tahminin görselini göster
predictions = model.predict(x_test[:5])
predicted_labels = np.argmax(predictions, axis=1)

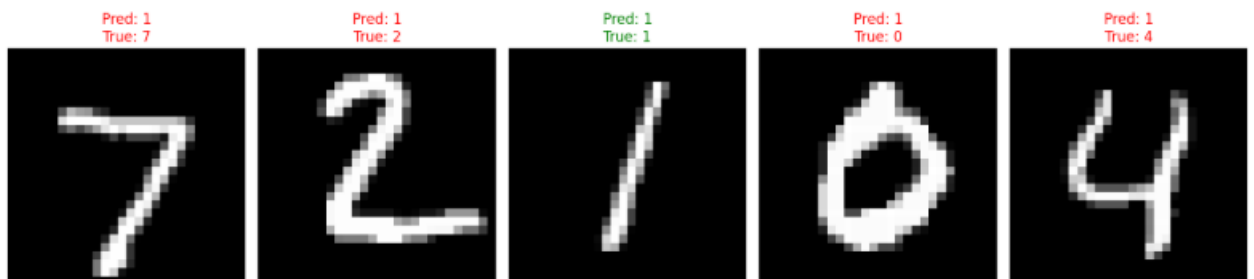
# Gerçek etiketler
true_labels = y_test[:5]

plt.figure(figsize=(15, 7))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(x_test[i], cmap='gray')
    predicted_label = predicted_labels[i]
    true_label = true_labels[i]

    # Renkli yazı ile tahminin doğruluğunu göster
    color = 'green' if predicted_label == true_label else 'red'
    plt.title(f"Pred: {predicted_label}\nTrue: {true_label}", color=color)
    plt.axis('off')
plt.tight_layout()
plt.show()

```

1/1 — 0s 48ms/step



```
In [24]: import matplotlib.pyplot as plt
import numpy as np

# İlk 5 tahminin görselini göster
predictions = model.predict(x_test[:5])
predicted_labels = np.argmax(predictions, axis=1)

# Gerçek etiketler
true_labels = y_test[:5]

# Şık ve renkli görseller için yeni ayar
plt.figure(figsize=(15, 7))
for i in range(5):
    plt.subplot(1, 5, i+1)

    # Sayıyı gradyan renk paletiyle görselleştir
    plt.imshow(x_test[i], cmap='coolwarm') # 'coolwarm' farklı renk tonları için iyi bir seçenek
    predicted_label = predicted_labels[i]
    true_label = true_labels[i]

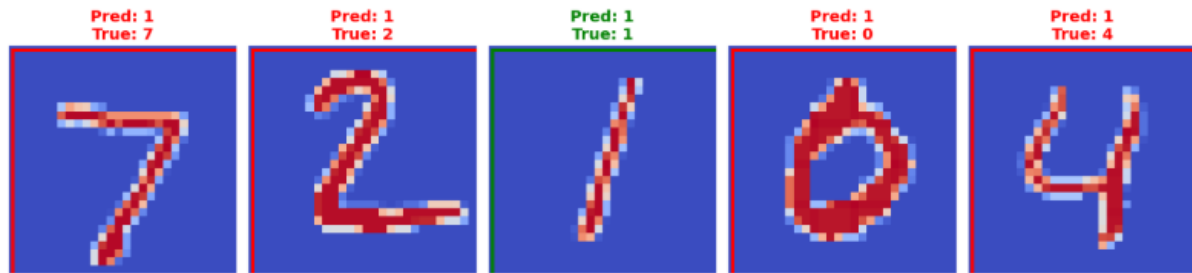
    # Başlıkları daha dikkat çekici hale getirelim
    color = 'green' if predicted_label == true_label else 'red'
    plt.title(f"Pred: {predicted_label}\nTrue: {true_label}", color=color, fontsize=14, weight='bold')

    # Çerçeve ekleyelim ve başlıkları etiketli hale getirelim
    plt.gca().add_patch(plt.Rectangle((0, 0), 28, 28, linewidth=3, edgecolor=color, facecolor='none'))

    # Sayıları gösterirken daha net yapalım
    plt.axis('off')

# Görselleştirmeyi sıkıştıralım
plt.tight_layout()
plt.show()
```

1/1 ————— 0s 58ms/step



In [ ]:

In [ ]:

```
7.8/7.8 MB 4.8 MB/s eta 0:00:00
Downloading contourpy-1.3.1-cp311-cp311-win_amd64.whl (219 kB)
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
Downloading fonttools-4.55.0-cp311-cp311-win_amd64.whl (2.2 MB)
2.2/2.2 MB 5.2 MB/s eta 0:00:00
Downloading kiwisolver-1.4.7-cp311-cp311-win_amd64.whl (56 kB)
Downloading pillow-11.0.0-cp311-cp311-win_amd64.whl (2.6 MB)
2.6/2.6 MB 5.9 MB/s eta 0:00:00
Downloading pyparsing-3.2.0-py3-none-any.whl (106 kB)
Installing collected packages: pyparsing, pillow, kiwisolver, fonttools, cycler, contourpy, matplotlib
Successfully installed contourpy-1.3.1 cycler-0.12.1 fonttools-4.55.0 kiwisolver-1.4.7 matplotlib-3.9.2 pillow-11.0.0 pyparsing-3.2.0

C:\Users\bengu>import matplotlib.pyplot as plt
'import' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\bengu>pip install pandas
Collecting pandas
  Downloading pandas-2.2.3-cp311-cp311-win_amd64.whl.metadata (19 kB)
Requirement already satisfied: numpy>=1.23.2 in c:\users\bengu\appdata\local\programs\python\python311\lib\site-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\bengu\appdata\local\programs\python\python311\lib\site-packages (from pandas) (2.9.0.post0)
Collecting pytz>=2020.1 (from pandas)
  Downloading pytz-2024.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.7 (from pandas)
  Downloading tzdata-2024.2-py2.py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: six>=1.5 in c:\users\bengu\appdata\local\programs\python\python311\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Downloading pandas-2.2.3-cp311-cp311-win_amd64.whl (11.6 MB)
11.6/11.6 MB 5.2 MB/s eta 0:00:00
Downloading pytz-2024.2-py2.py3-none-any.whl (508 kB)
Downloading tzdata-2024.2-py2.py3-none-any.whl (346 kB)
Installing collected packages: pytz, tzdata, pandas
Successfully installed pandas-2.2.3 pytz-2024.2 tzdata-2024.2

C:\Users\bengu>

C:\Users\bengu>pip install scipy
Collecting scipy
  Downloading scipy-1.14.1-cp311-cp311-win_amd64.whl.metadata (60 kB)
Requirement already satisfied: numpy<2.3,>=1.23.5 in c:\users\bengu\appdata\local\programs\python\python311\lib\site-packages (from scipy) (2.0.2)
Downloading scipy-1.14.1-cp311-cp311-win_amd64.whl (44.8 MB)
44.8/44.8 MB 3.4 MB/s eta 0:00:00
Installing collected packages: scipy
Successfully installed scipy-1.14.1

C:\Users\bengu>
```

Yüklenen gerekli kütüphaneler.

## 6. Test Veri Seti Üzerinde Tahminler

Modelin test veri seti üzerindeki performansı görselleştirilmiş ve bazı örneklerin gerçek ve tahmin edilen etiketleri karşılaştırılmıştır:

Yukarıdaki görselde:

- **Yeşil Başlık:** Doğru sınıflandırılmış örnekler
- **Kırmızı Başlık:** Yanlış sınıflandırılmış örnekler

## 7. Sonuç ve Değerlendirme

Model, Digit Recognizer veri seti üzerinde %99'un üzerinde bir doğruluk oranına ulaşmıştır. Bu sonuç, modelin el yazısı rakamlarının sınıflandırılması konusunda yüksek bir başarıya sahip olduğunu göstermektedir. Ancak, aşağıdaki geliştirme önerileri gelecekteki projeler için dikkate alınabilir:

- Daha karmaşık modellerin kullanılması (ör. transfer öğrenme).
- Veri artırma tekniklerinin uygulanması (ör. görüntü döndürme, ölçeklendirme).
- Farklı veri setleri üzerinde modelin performansının test edilmesi.

Model %99.12 doğruluk oranına ulaşmıştır. Gelecek çalışmalarda transfer öğrenme ve veri artırma tekniklerinin eklenmesiyle daha geniş veri setleri üzerinde performans artırılabilir.

## 7. Kaynaklar

- Digit Recognizer Veri Seti
- MNIST Veri Seti: MNIST Dataset
- TensorFlow ve Keras dokümantasyonu
- Python kütüphaneleri: Numpy, Matplotlib