**CONTENTS**

**Page**

**1. Definition of Problem 2**

**2. What is CNN (Convolutional Neural Networks) 2**

**3. MNIST Dataset 3**

**4. Progression 4-11**

**5. Results 12**

**6. Discussion and Conclusion 13**

1. **Definition of Problem: Handwritten Digit Recognation**

With the inclusion of artificial intelligence in all areas of our lives, it has brought many conveniences. The aim of the project is to recognize and classify handwritten figures in real time using image processing and machine learning techniques. The use of this technology includes banking, automation, translation, assistive technologies for visually impaired individuals, etc. It is possible to see active places.

1. **Convolutional Neural Networks**

Convolutional Neural Networks (CNNs) are a subfield of deep learning and are generally used for the analysis of visual information. Their common areas of use include image and video recognition, image classification, medical image analysis, and natural language processing.

CNN is the most suitable deep learning method for this project, as our project will also need images to recognize numbers. In addition, for a real-time recognition process, the snapshots taken from the camera must be processed and a decision must be made with the trained model.

To obtain the above functionality, CNN processes the image through various layers. These layers are:

• Convolutional Layer — Used to identify features

• Pooling (Downsampling) Layer — Reduces the number of weights and checks for compatibility

• Flattening Layer — Prepares data for the classic neural network

• Fully-Connected Layer — The standard neural network used for classification

diyagram, taslak, çizgi, plan içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 1. How does CNN Work

1. **MNIST Dataset**

The MNIST dataset is a widely used benchmark dataset in the field of machine learning and computer vision. It stands for the Modified National Institute of Standards and Technology database. The dataset consists of a collection of handwritten digits (0-9) and is commonly used for training and testing image classification algorithms.

The MNIST dataset was created by modifying a previous dataset called the NIST Special Database 3. The original NIST dataset was compiled by the National Institute of Standards and Technology (NIST) and included samples of handwritten digits from American Census Bureau employees and high school students.

To create the MNIST dataset, a subset of 60,000 training examples and 10,000 test examples was extracted from the NIST dataset. The images were normalized to fit into a fixed-size bounding box and then converted into grayscale images with a resolution of 28x28 pixels. The pixel intensities were also scaled to a range of 0 to 1.

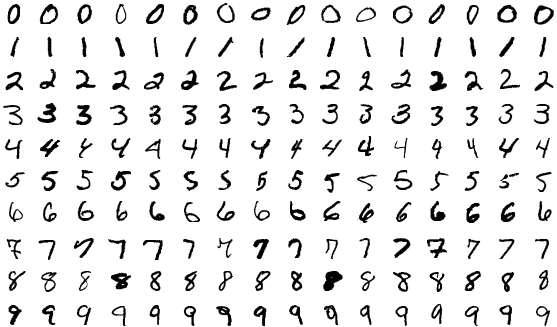


Figure 2. Samples from MNIST Dataset

The MNIST dataset has been widely used in the machine learning community for developing and evaluating image classification algorithms, particularly those based on deep learning. It provides a standardized and easily accessible dataset for researchers to compare their models' performance.

1. **Progression**

After determining the required dataset for the project (MNIST), the necessary libraries and dataset need to be loaded. We loaded our dataset using “Keras” library. Here are all the steps we used while writing the code:

**1. Required libraries are imported:**

- `cv2`: imports the OpenCV library to use for image processing.

- `numpy`: Imports a Python library used for numerical calculations.

- `matplotlib.pyplot`: Imports a library used for plotting graphics.

metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu - `tensorflow.keras`: Imports Keras API via TensorFlow.

Figure 3. Imports for the Project

**2. Loading the training dataset:**

- Loading MNIST dataset using `keras.datasets.mnist`.

- Assigning training data to `train\_images` and `train\_labels` variables.

- Assigning test data to variables `test\_images` and `test\_labels`.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 4. Loading Dataset

**3. The dataset is reshaped and normalized:**

- Reshape of training and test images to be `(28, 28, 1)`.

- The data is converted to `float32` data type and normalized in the range of 0-1.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 5. Reshaping Dataset

**4. Creating the model:**

- Generating `Sequential` model.

- Adding convolutional (Conv2D), maximum pooling (MaxPooling2D), flattening (Flatten) and fully connected (Dense) layers.

- "relu" and "softmax" are used as activation functions.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 6. Configuration of Training

**5. Model compiling:**

- `adam` optimizer and "sparse\_categorical\_crossentropy" loss function are used.

- The "accuracy" metric is monitored during training.

**6. The model is being trained:**

- The model is being trained using the `fit` function.

- Training images, training tags, epoch count, batch size and validation data are given as parameters.

- The education history is saved in the variable (`history`).

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 7. Training Model

**7.** **Training and validation loss graph plotted:**

- Graphing using the training loss (`loss`) and validation loss (`val\_loss`) values in the `history` variable.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 8. Code for Training and validation Loss Graph

**8. Training and validation accuracy graph is plotted:**

- Training accuracy in `history` variable (`accuracy`) and validation accuracy (`val\_accuracy`) values are plotted.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 9. Code for Training and Validation Accuracy Graph

**9. Graphing the training loss:**

- A graph is drawn using the training loss (`loss`) values in the `history` variable.

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 10. Code for Training Loss Graph

**10. Plotting a validation loss graph:**

- Graphing using the validation loss (`val\_loss`) values in the `history` variable.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 11. Code for Validation Loss Graph

**11. Plotting the training accuracy graph:**

- A graph is drawn using the training accuracy (`accuracy`) values in the `history` variable.

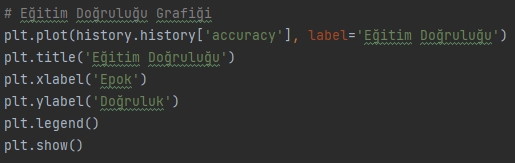


Figure 12. Code for Training Accuracy Graph

**12. The trained model is saved as `mnist\_model.h5`.**

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 12. Code for Saving Trained Model

**13-) Guessing User-Drawn Numbers Using the Model Trained**

A block of code separated by comment lines performs the process of guessing user-drawn numbers using the model trained to recognize handwritten numbers. Explanations of each code block shown at the upper side of code as comment.

# Eğitilmiş modeli yükleme  
model = keras.models.load\_model('mnist\_model.h5')  
  
  
# Sayıyı tahmin etme işlemini gerçekleştiren fonksiyon  
def predict\_number(image):  
 # Resmi yeniden boyutlandırma ve normalleştirme  
 image = cv2.resize(image, (28, 28))  
 image = image.astype('float32') / 255.0  
  
 # Modelin beklentisi olan şekle dönüştürme  
 image = np.expand\_dims(image, axis=0)  
 image = np.expand\_dims(image, axis=-1)  
  
 # Sayıyı tahmin etme  
 prediction = model.predict(image)  
 predicted\_label = np.argmax(prediction)  
  
 return predicted\_label  
  
# Kullanıcı tarafından çizilen sayıyı tahmin etme  
def recognize\_number():  
 drawing = False  
 last\_point = (0, 0)  
 canvas = np.zeros((400, 400), dtype='uint8')  
  
 # Çizim işlevleri için bir geri çağırma fonksiyonu  
 def draw\_callback(event, x, y, flags, param):  
 nonlocal drawing, last\_point, canvas  
  
 if event == cv2.EVENT\_LBUTTONDOWN:  
 drawing = True  
 last\_point = (x, y)  
  
 elif event == cv2.EVENT\_MOUSEMOVE:  
 if drawing:  
 cv2.line(canvas, last\_point, (x, y), (255, 255, 255), 30)  
 last\_point = (x, y)  
  
 elif event == cv2.EVENT\_LBUTTONUP:  
 drawing = False  
  
 # Çizim alanını oluşturma  
 cv2.namedWindow('Draw a Number')  
 cv2.setMouseCallback('Draw a Number', draw\_callback)  
  
 while True:  
 cv2.imshow('Draw a Number', canvas)  
 key = cv2.waitKey(1) & 0xFF  
  
 # Çizimi temizleme  
 if key == ord('c'):  
 canvas = np.zeros((400, 400), dtype='uint8')  
  
 # Tahmin yapma  
 elif key == ord('p'):  
 predicted\_label = predict\_number(canvas)  
 print("Tahmin: ", predicted\_label)  
  
 # Çıkış yapma  
 elif key == ord("q"):  
 break  
  
 cv2.destroyAllWindows()  
  
# Sayıyı tahmin etme işlemini başlatma  
recognize\_number()

After running the this part of code, a black window will be opened. User can draw any digit here. If user presses “p” after drawing a number, program will guess its class using trained model. If user draws something wrong, it can be deleted by pressing “c”. User also can close the window and end process by pressing “q”.

An example of usage:



Figure 12. User Drawed “1” and Model Guessed Number as “1”

**14-) Recognizing Numbers in Realtime Using the Camera**

A block of code lines uses the trained model to recognize numbers in real time using the camera grabber. It processes the image taken from the camera, frames the recognized numbers and prints them on the screen. Explanations of each code block shown at the upper side of code as comment.

import cv2  
import numpy as np  
from tensorflow import keras  
  
# Eğitilmiş modeli yükleme  
model = keras.models.load\_model('mnist\_model.h5')  
  
# Sayıyı tahmin etme fonksiyonu  
def predict\_number(image):  
 # Resmi yeniden boyutlandırma ve normalleştirme  
 image = cv2.resize(image, (28, 28))  
 image = image.astype('float32') / 255.0  
  
 # Modelin beklentisi olan şekle dönüştürme  
 image = np.expand\_dims(image, axis=0)  
 image = np.expand\_dims(image, axis=-1)  
  
 # Sayıyı tahmin etme  
 prediction = model.predict(image)  
 predicted\_label = np.argmax(prediction)  
  
 return predicted\_label  
  
# Kamera yakalayıcısını başlatma  
cap = cv2.VideoCapture(0)  
  
while True:  
 # Kamera görüntüsünü okuma  
 ret, frame = cap.read()  
  
 # Görüntüyü iyileştirme ve gri tonlamaya dönüştürme  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
 blurred = cv2.GaussianBlur(gray, (5, 5), 0)  
  
 # ROI'leri bulma  
 \_, threshold = cv2.threshold(blurred, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)  
 contours, \_ = cv2.findContours(threshold, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 numbers = [] # Tanınan sayıları tutmak için liste  
  
 for contour in contours:  
 # Kontur alanını kontrol etme  
 area = cv2.contourArea(contour)  
 if 500 < area: # Minimum ve maksimum alan değerlerini ihtiyaçlarınıza göre ayarlayabilirsiniz  
 # ROI'yi işleme  
 x, y, w, h = cv2.boundingRect(contour)  
 roi\_gray = gray[y:y + h, x:x + w]  
  
 # Sayıyı tahmin etme  
 predicted\_label = predict\_number(roi\_gray)  
  
 # Tanınan sayıyı listeye ekleme  
 numbers.append((predicted\_label, (x, y, w, h)))  
  
 # Tanınan sayıları kare içine alma ve ekrana yazdırma  
 for number, (x, y, w, h) in numbers:  
 cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)  
 cv2.putText(frame, str(number), (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (0, 255, 0), 2)  
  
 # Görüntüyü ekrana çizme  
 cv2.imshow('Kamera', frame)  
  
 # ESC tuşuna basılınca döngüden çıkma  
 if cv2.waitKey(1) == 27:  
 break  
  
# Kamera yakalayıcısını ve pencereleri kapatma  
cap.release()

 After running this part of code block, camera windows will be opened after seconds. Then program will be trying to detect handwritten numbers in real time. User can end the process by pressing “Esc” Recognition of two written two numbers on phone notepad:

metin, küçük alet, mobil telefon, İletişim Cihazı içeren bir resim

Açıklama otomatik olarak oluşturuldu

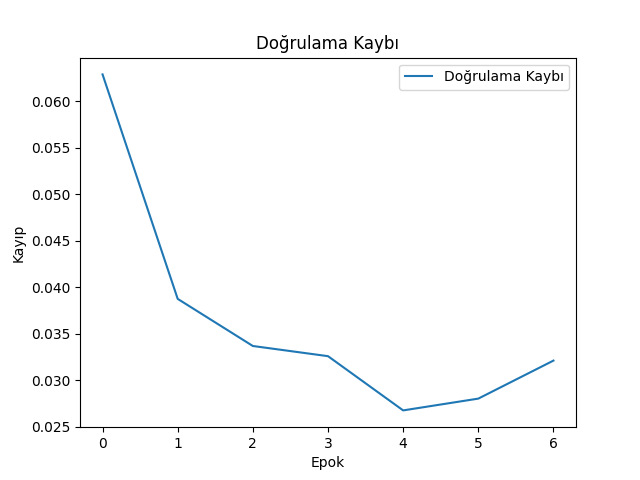
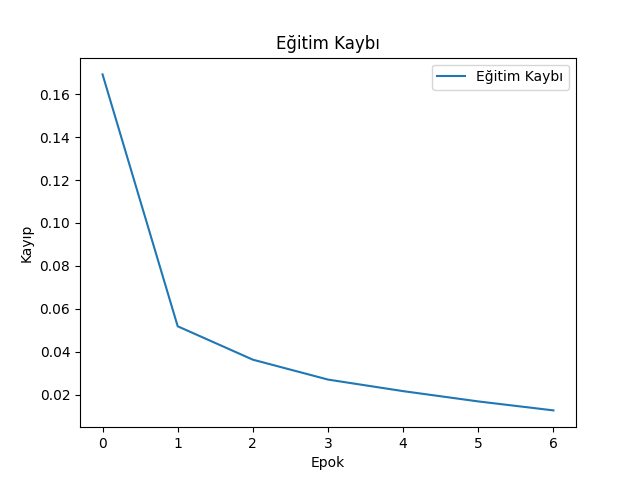
1. **Results:**

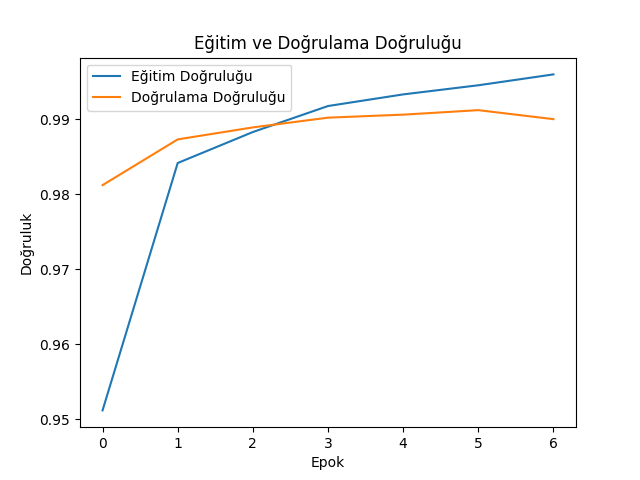
After many tries of training, best results were approximately at epoch 6. Also batch size for 64 gave better results. So we trained model for 6 epochs, “batch size=64”. Accuracy score for training and testing was over %99. We got results for Training Loss %0.02 and Validation loss %0.033.

Graphs of Trained Model:

metin, çizgi, diyagram, öykü gelişim çizgisi; kumpas; grafiğini çıkarma içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, diyagram, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu 



1. **Discussion and Conclusion:**

The project is running successfully, but many improvements are needed in real-time recognition. The content of the trained model should be better understood and the perception of non-numeric names as numbers should be corrected. Also, while the trained model works perfectly for handwriting, it may have problems reading digital or printed numbers. The reason for this can be attributed to variables such as font difference, font thickness, color, light. With the development of the project and the enrichment of the data set, it will be possible to turn the project into a mobile application that can be used almost everywhere. For example, with this project, it is potentially possible to read and transmit your friend's IBAN number with a phone camera instead of sending it as a message. We already use similar projects in our daily life. Models trained using more complex data sets are used in transactions such as real-time language translation and credit card recognition. Many of us know these technologies as OCR (Optical Character Recognition).