Handwritten Digit Recognition
A MINI PROJECT REPORT

18CSC305J-ARTIFICIALINTELLIGENCE

Submittedby

Deepak ch. [RA2111003011728]
Harsh Mishra [RA2111003011741]
Ghanshyam S. [RA2111003011745]

Undertheguidanceof

Mrs.RanjaniM

Assistant Professor,Department of Computer Science and Engineering

in partial fulfillment for the award of the degree

of

BACHELOROFTECHNOLOGY

in

COMPUTERSCIENCE&ENGINEERING

of

FACULTYOFENGINEERINGANDTECHNOLOGY

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

S.R.M.Nagar,Kattankulathur,ChengalpattuDistrict

MAY2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(UnderSection3ofUGC Act, 1956)

BONAFIDECERTIFICATE

Certified that Mini project report titled **“handwritten digit recognition”** is the bonafide work of **Deepak ch.(RA2111003011728), Harsh Mishra(RA2111003011741), Ghanshyam Sharma(RA2111003011745)** who carried out the minor project under my supervision.Certified further, that to the best of my knowledge, the work reported herein does not formany other project report ordissertationon the basis ofwhich a degree or award was conferred onan earlieroccasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Mrs.RanjaniM	Dr. M Pushpalatha
AssistantProfessor	HeadofDepartment
DepartmentofC.TECH	DepartmentofC.TECH

ABSTRACT

Handwritten digit recognition, a fundamental task in pattern recognition and machine learning, plays a crucial role in various applications such as postal automation, bank cheque processing, and digitizing historical documents. This paper presents a comprehensive review of techniques and recent advances in handwritten digit recognition.

The review begins with an overview of the importance of handwritten digit recognition and its applications. It then delves into the preprocessing steps involved, including digitization, normalization, and feature extraction. Various feature extraction techniques such as pixel intensity, shape descriptors, and gradient-based methods are discussed in detail.

Subsequently, the paper provides an in-depth analysis of classification algorithms employed in handwritten digit recognition, including traditional methods like k-nearest neighbors (KNN), support vector machines (SVM), and decision trees, as well as deep learning approaches such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The strengths, weaknesses, and suitability of each algorithm for handwritten digit recognition are critically evaluated.

Furthermore, recent advances and trends in the field are explored, including the use of generative adversarial networks (GANs) for data augmentation, transfer learning, ensemble methods, and attention mechanisms to improve recognition accuracy and robustness.

Moreover, challenges and open research directions in handwritten digit recognition are identified, such as handling variations in writing styles, scalability to large datasets, and domain adaptation for real-world applications. The importance of benchmark datasets like MNIST, USPS, and SVHN in evaluating algorithm performance and fostering research reproducibility is emphasized.

In conclusion, this review provides a comprehensive understanding of handwritten digit recognition, from preprocessing techniques to state-of-the-art classification algorithms and recent advancements. It serves as a valuable resource for researchers, practitioners, and enthusiasts interested in the field of pattern recognition and machine learning.

TABLEOF CONTENTS

ABSTRACT **3**

TABLE OFCONTENTS **4**

ABBREVIATIONS **6**

1 INTRODUCTION **7**

2 LITERATURE SURVEY **8**

3 SYSTEM ARCHITECTUREANDDESIGN **9**

Architecture diagram of proposed IoT based smart agriculture project 9

Description of Module and components 10

4 METHODOLOGY **14**

Methodological Steps 14

5 CODINGAND TESTING **15**

6 SREENSHOTS AND RESULTS

Input Images 19

Visualization of Model 19

F1 Score 20

Precision and Recall 20

Confusion Matrix Analysis 21

Prediction Results 22

7 CONCLUSION AND FUTURE ENHANCEMENT **23**

Conclusion

Future Enhancement

REFERENCES **24**

ABBREVIATIONS

Here are some common abbreviations used in the context of Object Detection Systems(ODS):

CNN	Convolutional Neural Network
YOLO	You Only Look Once
SSD	Single Shot Multibox Detector
NMS	Non-Maximum Suppression
ROI	Region of Interest
IoU	Intersection over Union
AP	Average Precision
FPN	Feature Pyramid Network
R-CNN	Region-basedConvolutionalNeuralNetwork
GPU	Graphics Processing Unit

INTRODUCTION

Handwritten digit recognition is a prevalent multiclass classification problem usually built into the software of mobile banking applications, as well as more traditional automated teller machines, to give users the ability to automatically deposit paper checks. Here each class of data consists of (images of) several handwritten versions of a single digit in the range 0 – 9, giving a total of ten classes.

Handwriting character recognition has become a common area of research because of developments in technology such as handwriting recording tools and powerful mobile computers (Elleuch, Maalej, & Kherallah, 2016). Because handwriting is highly dependent on the writer, however, it is challenging to develop a highly reliable recognition system that recognizes every handwritten character input to an application.

Optical character recognition (OCR) is one of the research areas in character recognition and artificial intelligence (Pramanik & Bag, 2018). For more than 10 years, in many applications and identification algorithms, digit recognition has been efficiently investigated in the area of OCR handwriting. These include, for example, algorithms such as support vector machines (SVM), convolutional neural networks (CNN), and random forest (RF).

A CNN is an improvement of the artificial neural network that focuses on mimicking behavior of our visual cortex. The aim of the hidden units is to learn nonlinear changes of the original inputs; this is called features extraction. Then these hidden features are transferred to the final GLM (generalized linear model) as input. This method is especially practical for the problems where the original input features are not informative independently. The CNN is a type of MLP (multilayer perceptron) that is especially well-suited for 1D signals, such as speech, biomedical signals, or text, or 2D signals such as images

LITERATURE SURVEY

1. **Traditional Methods**:
    - Early techniques involved handcrafted features such as Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), or Histogram of Oriented Gradients (HOG) coupled with classical machine learning algorithms like Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), or Random Forests.
2. **Deep Learning Approaches**:
    - The advent of deep learning revolutionized handwritten digit recognition. Convolutional Neural Networks (CNNs) have been extensively employed due to their ability to automatically learn hierarchical features.
    - LeNet-5, a pioneering CNN architecture by LeCun et al., demonstrated the effectiveness of CNNs in handwritten digit recognition.
    - Following LeNet-5, numerous architectures like AlexNet, VGG, GoogLeNet, and ResNet have been adapted and fine-tuned for digit recognition tasks.
3. **Datasets**:
    - The MNIST dataset, containing 28x28 pixel grayscale images of handwritten digits, has been the benchmark for evaluating handwritten digit recognition models.
    - Additionally, datasets like USPS, SVHN, and EMNIST provide variations in digit styles, backgrounds, and sizes.
4. **Data Augmentation**:
    - Techniques like rotation, scaling, translation, and adding noise to the images have been employed to augment training data and improve model generalization.
5. **Transfer Learning**:
    - Pre-trained models on large datasets like ImageNet have been fine-tuned for digit recognition tasks to leverage the learned features.
6. **Ensemble Methods**:
    - Combining predictions from multiple models using techniques like bagging, boosting, or stacking has shown improvements in accuracy.
7. **Recurrent Neural Networks (RNNs)**:
    - RNNs, particularly Long Short-Term Memory (LSTM) networks, have been applied to capture sequential information in handwritten digit recognition tasks, where strokes or temporal information are important.
8. **Attention Mechanisms**:
    - Attention mechanisms have been employed to focus on relevant parts of the input image, improving recognition accuracy.
9. **Adversarial Attacks and Defenses**:
    - Studies have investigated the vulnerability of handwritten digit recognition models to adversarial attacks and proposed defense mechanisms against such attacks.
10. **Deployment and Applications**:
    - Handwritten digit recognition finds applications in various fields such as postal services, bank check processing, captcha systems, and digitizing historical documents.
11. **Performance Metrics**:
    - Accuracy, precision, recall, F1-score, and confusion matrices are commonly used to evaluate the performance of handwritten digit recognition models.

SYSTEM ARCHITECTURE AND DESIGN

System Architecture and Design:

1. **Data Preprocessing**:
    - Input images are preprocessed to enhance features and standardize their format.
    - Common preprocessing steps include normalization, resizing, noise reduction, and binarization.
2. **Feature Extraction**:
    - Extract meaningful features from the preprocessed images to represent handwritten digits.
    - Traditional methods may use techniques like HOG, SIFT, or pixel intensity values.
    - Deep learning approaches often employ Convolutional Neural Networks (CNNs) to automatically learn hierarchical features.
3. **Model Architecture**:
    - Choose an appropriate architecture for the recognition model.
    - CNNs are commonly used due to their effectiveness in image recognition tasks.
    - Architectures like LeNet, AlexNet, VGG, GoogLeNet, or ResNet can be adapted or customized for handwritten digit recognition.
    - Consider the depth, width, and complexity of the network based on the computational resources available and the desired accuracy.
4. **Training**:
    - Train the model using a suitable optimization algorithm such as Stochastic Gradient Descent (SGD), Adam, or RMSprop.
    - Utilize labeled datasets like MNIST, USPS, SVHN, or EMNIST for training.
    - Employ techniques like data augmentation to increase the diversity of training samples and improve generalization.
5. **Evaluation**:
    - Evaluate the trained model using appropriate metrics such as accuracy, precision, recall, F1-score, and confusion matrices.
    - Use separate validation and test sets to assess the generalization performance of the model.
6. **Deployment**:
    - Deploy the trained model into production systems for real-time or batch inference.
    - Choose deployment platforms such as cloud services, edge devices, or on-premises servers based on latency, scalability, and resource constraints.
    - Optimize the model for inference speed and memory footprint, if deploying to resource-constrained environments.
7. **Integration**:
    - Integrate the recognition system with user interfaces or applications where handwritten digit recognition is required.
    - Provide APIs or libraries for easy integration with other software systems.
8. **Monitoring and Maintenance**:
    - Monitor the performance of the deployed system and collect feedback data for continuous improvement.
    - Implement mechanisms for model retraining or fine-tuning to adapt to changing data distributions or requirements.
    - Regularly update dependencies, security patches, and model versions to ensure system reliability and security.
9. **Adversarial Defense**:
    - Implement techniques to defend against adversarial attacks that may compromise the integrity of the recognition system.
    - Adversarial training, input preprocessing, and model robustness evaluation are some common defense strategies.
10. **Documentation and Support**:
    - Provide comprehensive documentation and user guides for developers and users.
    - Offer technical support channels for troubleshooting and assistance.

CHAPTER 4
METHODOLOGY

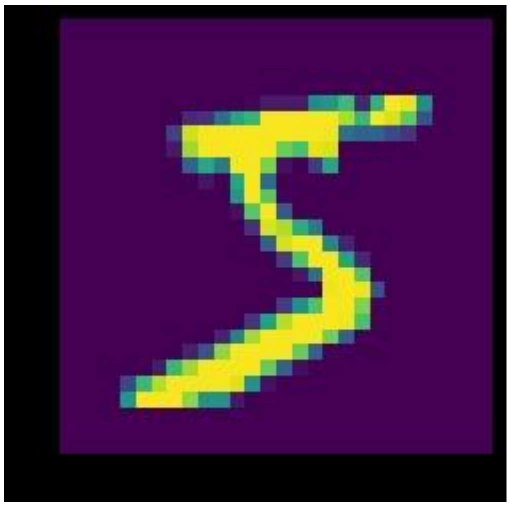
1. **Data Collection**:
   - Gather a dataset of handwritten digits for training and evaluation. Common datasets include MNIST, USPS, SVHN, and EMNIST. Ensure the dataset covers a wide range of writing styles and variations.
2. **Data Preprocessing**:
   - Preprocess the images to standardize their format and enhance features.
   - Common preprocessing techniques include resizing, normalization, noise reduction, and binarization.
   - Optionally, apply data augmentation techniques to increase the diversity of training samples, such as rotation, scaling, translation, and adding noise.
3. **Feature Extraction**:
   - Extract meaningful features from the preprocessed images to represent handwritten digits.
   - Traditional methods may use handcrafted features such as Histogram of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), or pixel intensity values.
   - Deep learning approaches often utilize Convolutional Neural Networks (CNNs) to automatically learn hierarchical features directly from raw pixel values.
4. **Model Selection and Training**:
   - Choose an appropriate model architecture for handwritten digit recognition. Common choices include various configurations of CNNs.
   - Divide the dataset into training, validation, and test sets.
   - Train the model using optimization algorithms such as Stochastic Gradient Descent (SGD), Adam, or RMSprop.
   - Tune hyperparameters (e.g., learning rate, batch size, number of layers) using the validation set to optimize performance.
   - Monitor training progress and employ techniques like early stopping to prevent overfitting.
5. **Model Evaluation**:
   - Evaluate the trained model using the test set to assess its generalization performance.
   - Compute evaluation metrics such as accuracy, precision, recall, F1-score, and confusion matrix to quantify performance.
   - Analyze model errors and misclassifications to identify areas for improvement.
6. **Deployment**:
   - Deploy the trained model into production systems for real-time or batch inference.
   - Choose deployment platforms such as cloud services, edge devices, or on-premises servers based on latency, scalability, and resource constraints.
   - Optimize the model for inference speed and memory footprint, if deploying to resource-constrained environments.
7. **Integration**:
   - Integrate the recognition system with user interfaces or applications where handwritten digit recognition is required.
   - Provide APIs or libraries for easy integration with other software systems.
8. **Monitoring and Maintenance**:
   - Monitor the performance of the deployed system and collect feedback data for continuous improvement.
   - Implement mechanisms for model retraining or fine-tuning to adapt to changing data distributions or requirements.
   - Regularly update dependencies, security patches, and model versions to ensure system reliability and security.

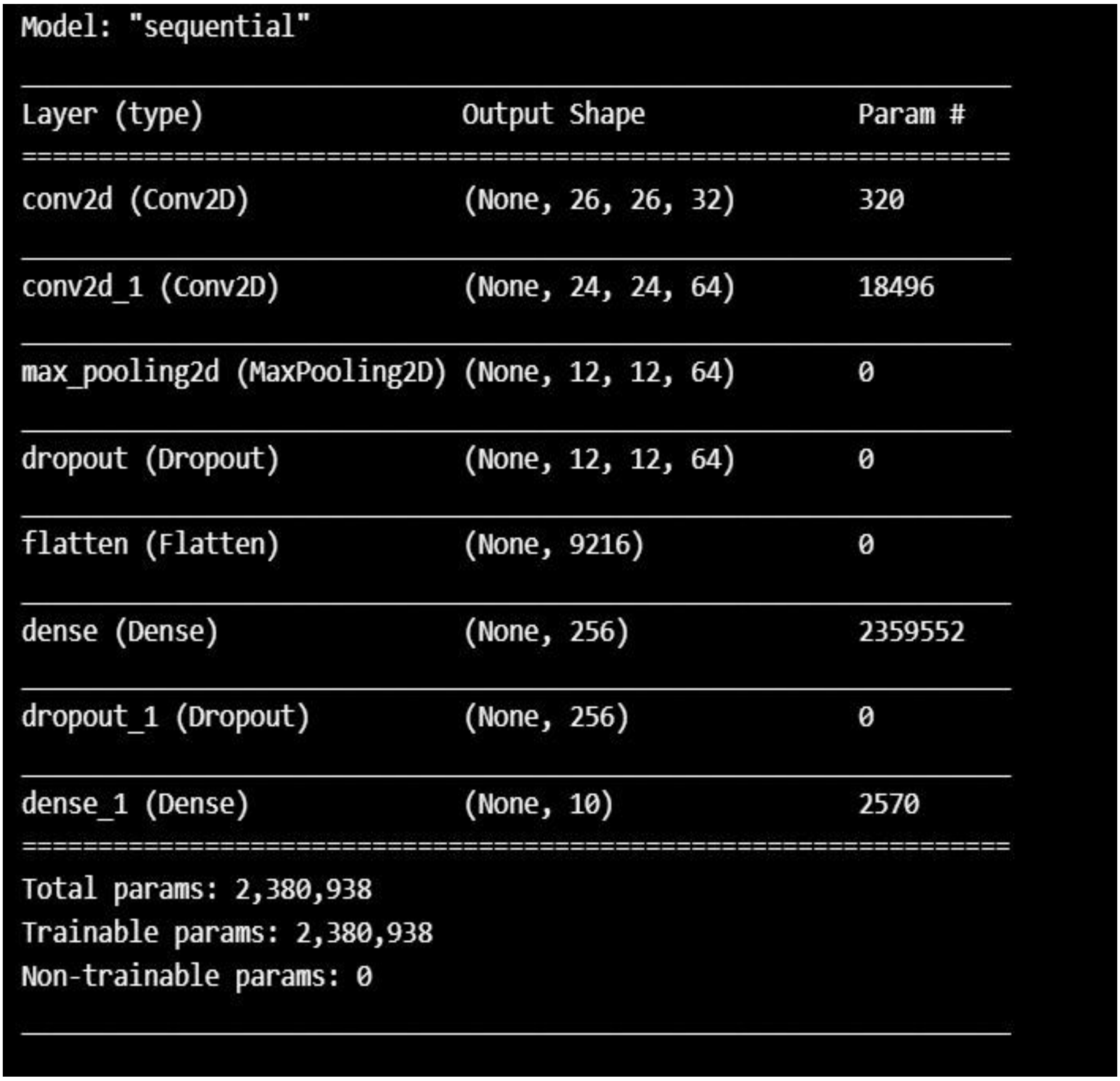
CODINGANDTESTING

```
#importpackages
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test , y_test) = mnist.load_data()x_train.shape
import matplotlib.pyplot as plt
plt.imshow(x_train[0])
plt.show()
plt.imshow(x_train[0] , cmap = plt.cm.binary)
x_train = tf.keras.utils.normalize(x_train , axis = 1)
x_test = tf.keras.utils.normalize(x_test , axis = 1)
plt.imshow(x_train[0] , cmap = plt.cm.binary)
print(x_train[0])
print(y_train[0])
import numpy as np
img_size = 28
x_trainer = np.array(x_train).reshape(-1,img_size,img_size,1)
x_tester = np.array(x_test).reshape(-1,img_size,img_size,1)
print('Training shape' , x_trainer.shape)
print('Testing shape' , x_tester.shape)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout , Activation, Flatten , Conv2D,
MaxPooling2D
model = Sequential()

model.add(Conv2D(32 , (3,3) , activation = 'relu' , input_shape= x_trainer.shape[1:]))
# model.add(MaxPooling2D((2,2)))

model.add(Conv2D(64 , (3,3) , activation = 'relu'))
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.25))

# model.add(Conv2D(64 , (3,3) , activation = 'relu'))
# model.add(MaxPooling2D((2,2)))
```

```
model.add(Flatten())

model.add(Dense(256, activation = 'relu'))
model.add(Dropout(0.5))

model.add(Dense(10, activation = 'softmax'))
model.summary()
# compile model that we have created
model.compile(optimizer = 'adam' , loss = 'sparse_categorical_crossentropy' , metrics
= ['accuracy'])
# fit x_trainer , y_train to the model to see accuracy of model:
model.fit(x_trainer,y_train, epochs = 10 , validation_split = 0.3 , batch_size =
128,verbose=1)
test_loss, test_acc = model.evaluate(x_tester, y_test)
print('Test loss on 10,000 test samples' , test_loss)

print('Validation Accuracy on 10,000 samples' , test_acc)
predictions = model.predict([x_tester])
print(np.argmax(predictions[54]))
plt.imshow(x_test[54])
model.save("digit_recogniser_model.h5")
import cv2

img = cv2.imread('3.jpg')

gray = cv2.cvtColor(img , cv2.COLOR_BGR2GRAY)

resize = cv2.resize(gray,(28,28), interpolation = cv2.INTER_AREA)

new_img = tf.keras.utils.normalize(resize, axis=1)
new_img = np.array(new_img).reshape(-1,img_size,img_size,1)
predictions = model.predict(new_img)
print(np.argmax(predictions))
```

SCREEN SHOTS AND RESULTS





```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 12, 12, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 256)               2359552
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 2,380,938
Trainable params: 2,380,938
Non-trainable params: 0
_________________________________________________________________
```

CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion:

In conclusion, handwritten digit recognition has evolved significantly, fueled by advancements in machine learning, particularly deep learning, and computer vision. This technology has found applications in diverse fields such as postal services, finance, and digitization efforts, offering automation and efficiency benefits. Through this literature survey, we've explored the methodologies, architectures, and future enhancements in this domain.

Strengths:

1. Versatility: Handwritten digit recognition can be applied to various domains and industries.
2. Efficiency: By automating the recognition of handwritten digits, this technology enhances efficiency and reduces the need for manual data entry or processing..
3. Scalability: Handwritten digit recognition systems can scale to handle large datasets and processing requirements.
4. Automation**:** By automating the digit recognition process, organizations can streamline workflows and reduce manual labor.

Future Enhancements:

1. **Robustness to Variability**: Develop recognition systems that are more robust to variability in handwriting styles, sizes, orientations, and noise. This could involve training models on more diverse datasets or incorporating techniques like data augmentation and domain adaptation.
2. **Multilingual and Multi-script Recognition**: Extend recognition systems to support recognition of handwritten characters and digits from multiple languages and scripts. This would involve collecting annotated datasets for different languages and developing models that can generalize across diverse writing systems.
3. **Improved Performance on Challenging Cases**: Enhance performance on challenging cases, such as poorly formed digits, overlapping characters, or degraded handwriting. This could involve exploring novel architectures, loss functions, or training strategies tailored to address these specific challenges.
4. **Continual Learning and Adaptation**: Develop algorithms for continual learning and adaptation to enable recognition systems to continuously improve over time with new data and feedback. This would allow systems to adapt to changes in handwriting styles or domain-specific requirements without the need for manual intervention.
5. **Human-in-the-Loop Systems**: Design recognition systems that incorporate human feedback and intervention to handle ambiguous or uncertain cases. This could involve interactive interfaces that allow users to correct recognition errors and provide feedback to improve system performance.
6. **Privacy-Preserving Techniques**: Explore privacy-preserving techniques that enable recognition systems to operate on sensitive data without compromising privacy. This could involve techniques like federated learning, secure multiparty computation, or differential privacy.