Signature Extraction and Classification from Images

Mohammad Faizan

September 29, 2024

Abstract

This report presents image processing, data organization, and image classification tasks to create a comprehensive system for detecting contours within images, organizing the images into training and validation datasets, and then training a Convolutional Neural Network (CNN) model for image classification.

Contents

1 Introduction

Signature extraction and matching is essential for fraud prevention, identity verification, legal compliance, and efficiency across various sectors. In financial services, it helps verify signatures on checks and loan documents. Government agencies use it for authenticating signatures on legal documents and IDs. The healthcare industry employs it to confirm patient signatures on consent forms, while educational institutions utilize it to validate student identities on exams and registration forms. The process involves detecting, locating, and cropping handwritten signatures from documents.
Given a dataset of 16 pages, each containing a 12 by 5 table filled with signatures, the task was to train a Convolutional Neural Network (CNN) model to classify each signature according to a unique ID.

2 Techniques

To train the model, we first needed to extract the data from different columns and save each piece in separate folders so the model could read and extract features from them. The current task was to extract the signatures from the images and save them in a hierarchical form for the model to read and use. The following methods were exported for the extraction of the signatures.

2.1 Contour Detection and Image Processing

The following methods were explored.

2.1.1 Edge Detection

Canny edge detection was initially applied to highlight the edges in the images. However, due to the low resolution of the images, this method was ineffective for all images, leading to inconsistent results in edge detection. To ensure the images fit within the maximum allowed dimensions (4650 pixels), the calculate scale percent function dynamically calculates the scaling percentage based on the largest dimension of the image. The image is then resized accordingly, preserving the aspect ratio.[2].

2.1.2 Contour Detection

Contour detection was utilized to identify the boundaries of signatures. While it provided a way to visualize the shapes, challenges arose when determining the appropriate contours due to the diversity of signature forms[1].

2.1.3 Grayscale Conversion and Thresholding

The color image is converted into a grayscale image, after which adaptive thresholding is applied. The thresholding step helps to segment the image into binary

form, simplifying the task of identifying contours.[1].

2.1.4 Clustering

KNN clustering aimed to mark potential signature areas and apply bounding boxes. Unfortunately, the dynamic determination of the required clusters for each image proved challenging, leading to inaccurate results as the formed clusters did not align well with the actual signatures[3].

2.1.5 8-Way Neighborhood

The 8-way neighborhood technique failed as some signatures overlapped with others, making it difficult to accurately separate individual signatures in crowded images.

2.1.6 Contour Area

Contour area measurement faced challenges due to the varying areas of the images and improper signatures that fell outside the expected bounding boxes. This technique was thus limited in its effectiveness[1].

2.1.7 Morphological Operations

Morphological operations (closing) are applied to clean up the binary image by filling small holes and gaps. This step ensures that contours are detected more accurately.

2.1.8 Contours with Hierarchy

Contours with hierarchy provided an effective way to segment images by considering the relationships between contours. This method allowed for a clearer segmentation of signatures and proved to be the most effective technique employed[4].

3 Preprocessing Logic

The preprocessing logic involved several steps to optimize the extraction of signatures from images:

- **Image Loading:** All images were loaded from the specified directory.
- **Contour Detection:** Contours were extracted from the images using a hierarchical tree. We noticed that some signatures were still being detected with this method, so we added conditions to ignore certain shapes in the contours. For example, if there was a specific carve in the contour or if signatures were overlapping, we decided to ignore those. This helped to significantly clean the data.

- **Area Filtering:** The signatures were then cropped and saved in folders. However, some images contained random contours instead of signatures, which were missed during the previous preprocessing. To solve this, we set a specific threshold for the image area to remove all the unnecessary images.
- **Image Rotation:** To maintain a consistent number of images across groups, images were rotated to ensure each group contained four images. This process involved rotating images 90 degrees clockwise and duplicating existing images if necessary.

4 Model Training

- **Import Libraries:** The code starts by importing necessary libraries for building and training a neural network using PyTorch, a popular deep learning framework.
- **Check Device:** It checks if a GPU (Graphics Processing Unit) is available for faster computations. If not, it will use the CPU (Central Processing Unit).
- **Clear GPU Memory:** It clears the GPU memory to avoid running out of memory during training.
- **Set Random Seed:** A random seed is set for reproducibility, meaning that running the code multiple times will give the same results.
- **Define Data Path:** It sets the directory where the images are stored, which are organized into folders by class.
- **Set Image and Training Parameters:** It defines the height and width of the images, batch size (number of images processed together), and the number of training epochs (how many times the model will see the entire dataset).
- **Image Transforms:** It specifies transformations to be applied to the images, like resizing them, converting them to tensor format, and normalizing pixel values.
- **Load Dataset:** It loads the images into a dataset using the specified transformations, allowing the model to access them during training.
- **Split Dataset:** The dataset is divided into training (80%) and validation (20%) sets. The training set is used to teach the model, and the validation set is used to evaluate its performance.
- **Create Data Loaders:** Data loaders are created for both training and validation sets, which handle loading the data in batches during training.

- **Define CNN Model:** A Convolutional Neural Network (CNN) model is defined. It includes layers that extract features from images and layers that classify those features into different categories.
- **Initialize Model:** The CNN model is initialized and moved to the GPU (if available).
- **Define Loss and Optimizer:** The loss function (to measure the difference between predicted and actual labels) and optimizer (to adjust the model's weights to minimize loss) are defined.
- **Training Loop:**
  - The model goes through the training data for a User specificed of epochs:
  - For each batch of images, it computes the predictions and calculates the loss.
  - It updates the model's weights based on the loss to improve predictions.
  - It tracks training loss and accuracy.
- **Validation:** After training on the training set, the model is evaluated on the validation set to see how well it generalizes to unseen data.
- **Print Results:** After each epoch, it prints the training and validation loss and accuracy to track performance.
- **Plot Training History:** After training, it plots the training and validation accuracy and loss over epochs for visual analysis of the model's performance.
- **Save the Model:** Finally, the trained model is saved to a file so that it can be used later without retraining.

5 Result Comparison

6 Conclusion

The CNN outperforms the ANN significantly in both training loss and accuracy. The ANN struggles to learn effectively, as indicated by its high and stagnant loss and extremely low accuracy.

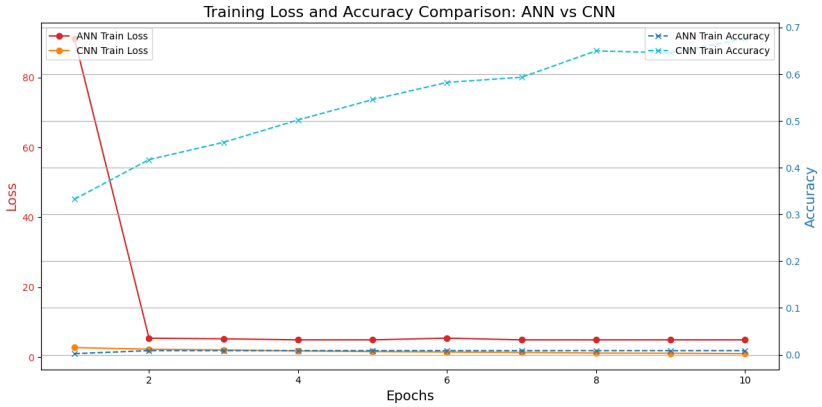
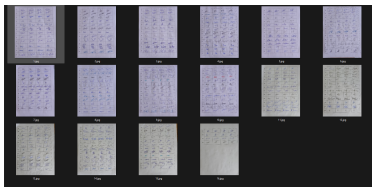


Figure 1: ANN vs CNN Results

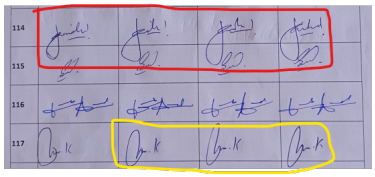
Epoch	ANN		CNN	
(lr)2-3 (lr)4-5	Loss	Accuracy	Loss	Accuracy
1	91.0769	0.0022	2.7956	0.3326
2	5.4734	0.0087	2.3207	0.3174
3	5.2950	0.0087	2.1146	0.4543
4	5.0031	0.0087	1.8314	0.5022
5	5.0031	0.0087	1.6469	0.5457
6	5.4894	0.0087	1.5251	0.5826
7	5.9972	0.0087	1.4047	0.5935
8	4.9999	0.0087	1.2133	0.6500
9	4.9886	0.0087	1.1694	0.6457
10	4.9977	0.0087	1.0551	0.6124

Table 1: Training Results for ANN and CNN

7 Images



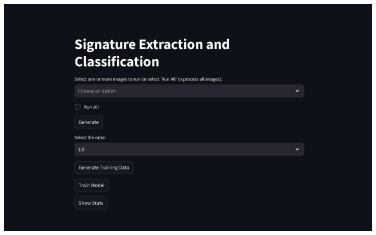
(a) Original Dataset



(b) Abnormalities

(c) Images before Contour

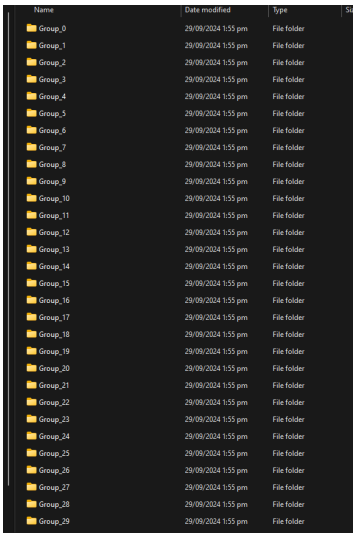
(d) Images After Contour



(e) Images Cropped after Contour

(f) Images Cleaned

(g) Images After Contour



(h) Images Sorted into groups

Figure 2: Comparison of the original dataset and images after applying contours.

References

[1] Mar 2021.

[2] OpenCV. Opencv: Canny edge detection.

[3] How to tune hyperparameters with Python and scikit-learn-PyImageSearch says. k-nn classifier for image classification, Aug 2016.

[4] Alan Wong. Contours and hierarchy with opencv, Jul 2021.