# Digit Recognition Using Shape Matching and Histogram of Oriented Gradients

Saketh Sai Ram – EE22B022 | Purushottam – EE22B073

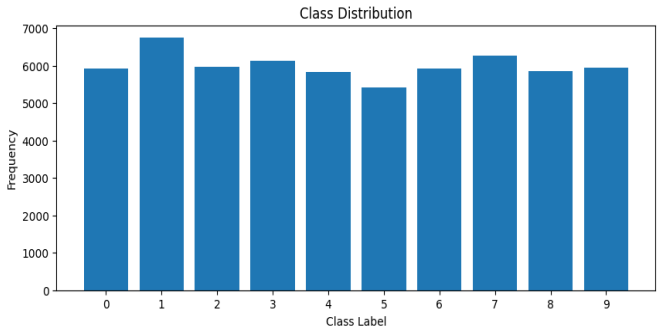
## 1. Project Overview

This project focuses on recognizing handwritten digits from input images and classifying them into one of the digits from 0 to 9. Two algorithms—Shape Matching Technique (SM) and Histogram of Oriented Gradients (HOG)—have been implemented to accomplish this. Their performance will be evaluated and compared against each other in various cases, such as rotation, scaling, and a combination of them.

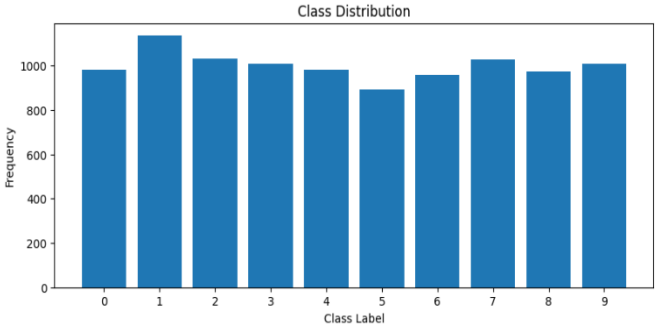
Subsequently, the project extends to recognizing digits within images containing multiple handwritten digits. It first extracts individual digits by detecting contours and then classifies them using the previously developed algorithms.

## 2. Dataset

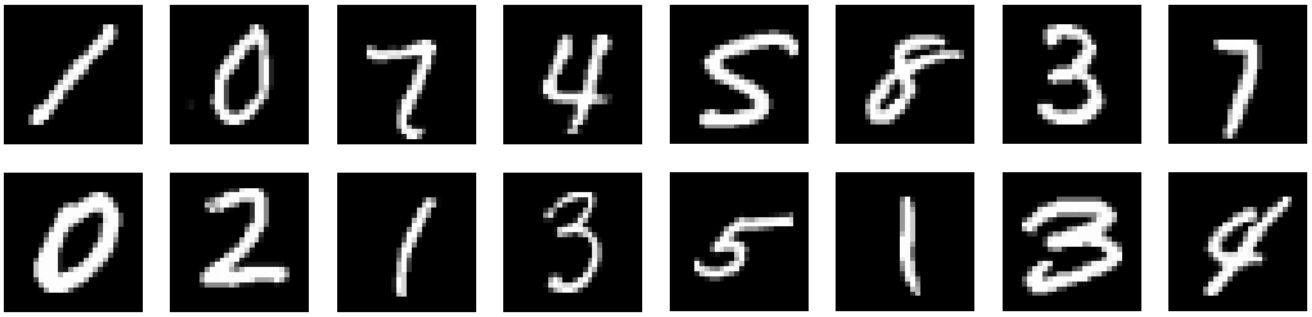
The algorithms are tested using the MNIST dataset, available through the Keras library. This dataset consists of 60,000 training samples and 10,000 testing samples, where each sample is a 28×28 grayscale image of a handwritten digit. The digit distribution is nearly uniform, ensuring minimal bias toward any specific class.




a) Training Data Distribution




b) Testing Data Distribution



c) Sample Images from the Training Dataset

## 3. Shape Matching

Shape matching is a fundamental technique in image processing that aims to compare and recognize shapes within images, even under transformations like rotation, scaling, or deformation. Mathematically, it tries to find a transformation $T$ that aligns one shape $A$ with another shape $B$ as closely as possible, minimizing the distance function between them. The pipeline consists of the following steps:

- ***Edge Detection*** **-** Extracting the contours of the shape.
- ***Shape Context Descriptor*** **-** Creating a feature representation for each point on the shape.
- ***Matching and Cost Computation*** **-** Using various cost functions to match corresponding points between shapes.

## 3.1. Edge Detection

The edge detection algorithm employs Sobel operators, Gaussian smoothing, Non-Maximum Suppression (NMS), double thresholding, and hysteresis to identify edges or contours in the input image. Additionally, it offers the option to utilize OpenCV's Canny Edge Detector to enhance the accuracy and performance of the algorithm. The process of detecting edges is explained below.

### 3.1.1. Gradient Computation (Sobel Operators)

The gradient measures the rate of change in intensity at each pixel. In the Sobel method, the gradient is computed by convolving the image with the Sobel kernels:

- Along the X-direction, $G_x = [[-1, -2, -1]^T \quad [0, 0, 0]^T \quad [1, 2, 1]^T]$
- Along the Y-direction, $G_y = [[-1, 0, 1]^T \quad [-2, 0, 2]^T \quad [-1, 0, 1]^T]$

When convolved with the image, these kernels detect the edges in the horizontal ($G_x$) and vertical ($G_y$) directions, respectively. The gradient magnitude and angle are then computed at each pixel:

- Gradient Magnitude, $M(x, y) = \sqrt{G_x(x, y)^2 + G_y(x, y)^2}$
- Gradient angle (orientation of the edge), $\theta(x, y) = arctan2\big(G_y(x, y), G_x(x, y)\big) \times \frac{180}{\pi}$ – This angle is mapped between $0^0$ and $180^0$ to standardize directions.

### 3.1.2. Gaussian Smoothing

It is applied to the gradient results to reduce noise and smooth out the image. It uses a Gaussian kernel with a standard deviation $\sigma$, which smooths the gradient images before computing the magnitude. To produce smoothed versions of gradient images, the gradients $G_x$ and $G_y$ are convolved with the Gaussian kernel.

### 3.1.3. Non-Maximum Suppression (NMS)

It is used to thin the edges detected by the Sobel operator. The goal is to preserve only the local maxima in the gradient direction and discard the rest. The procedure to perform this,

- For each pixel, check the two neighbouring pixels in the gradient direction $\theta(x, y)$ and compare the gradient magnitude.
- If the pixel's gradient magnitude is greater than both its neighbours, it is preserved; otherwise, it is suppressed (set to zero).

Mathematically, for a pixel $(x, y)$, check its neighbors in the direction $\theta(x, y)$ and keep it only if:

$$M(x, y) > M(x + \delta_x, y + \delta_y) \text{ and } M(x, y) > M(x - \delta_x, y - \delta_y)$$

- $\big(\delta_x, \delta_y\big)$ are the offsets based on the gradient direction.

### 3.1.4. Thresholding

After applying NMS, the edges need to be categorized into strong edges, weak edges, and non-edges using double thresholding.

*Double Thresholding*

- Strong edge: A pixel with a gradient magnitude above a high threshold (given by `self.high`).
- Weak edge: A pixel with a gradient magnitude between the low and high threshold (given by `self.low`).
- Non-edge: A pixel with a gradient magnitude below the low threshold.

### 3.1.5. Hysteresis

After identifying strong and weak edges, hysteresis is used to connect weak edges that are connected to strong edges. This step ensures that isolated weak edges (which are likely noise) are removed. The procedure followed is explained below.

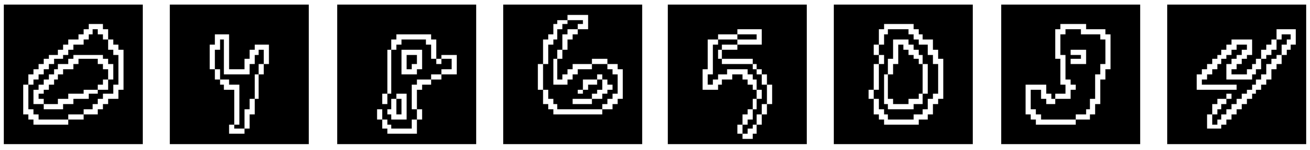
- Start by marking all strong edges as true edges.
- For each weak edge, check if it is connected to any strong edge (using a 3x3 connectivity kernel). If connected, classify the weak edge as a true edge.
- Continue this process until no more weak edges can be connected.

This is done iteratively using dilations (expansion of the strong edge mask) until no further changes occur.

#### 3.1.6. Final Edge Detection

The final edge map is generated by applying hysteresis to the edges identified after double thresholding. The result is a binary image where 1 (or 255) indicates an edge, and 0 indicates no edge.

a) Detected edges using the implemented algorithm

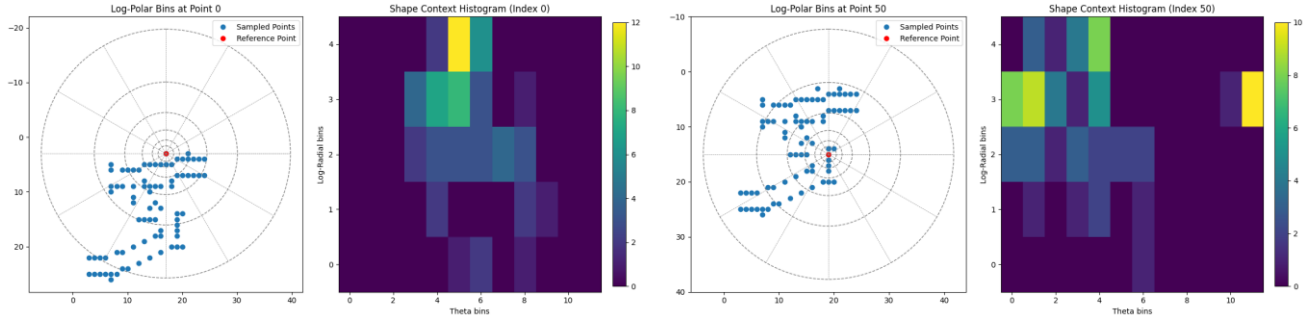


b) Detected edges using the canny edge detector

### 3.2. Shape Context Descriptor

It computes shape context histograms for a set of 2D points, typically obtained from edge detection on an object silhouette. Each histogram captures the spatial arrangement of other points relative to a reference point in log-polar space (radius and angle), effectively encoding the local shape geometry. The steps followed to compute the descriptor are as follows.

- Uniformly samples edge points for an image up to a specified maximum (`max_points`). If the total number of points is already below this threshold, all points are retained. Sampling is performed by selecting evenly spaced indices across the input set. This process ensures efficiency. $S$ denotes this sample set.
- For each reference point $s_i \in S$, compute the relative position to every other point $s_j \in S, j \neq i$.
  - Euclidean Distance: $r_{ij} = \sqrt{ss^T}$ where, $s_i = [x_i, y_i]$, $s_j = [x_j, y_j]$ and $s = s_i - s_j$.
  - Angle: $\theta_{ij} = arctan2(y_i - y_j, x_i - x_j)\ mod\ 2\pi$.
  - To ensure scale invariance, all distances are normalized by the mean of all non-zero distances.
- ***Log-Polar Binning*:** Define bin edges as `r_bin_edges`: logarithmically spaced from inner radius to outer radius; `theta_bin_edges`: uniformly spaced over 0 to $2\pi$.
- ***Histogram Construction*:** For each point, determine the bin indices for each other point based on its $(r, \theta)$. Then increment the corresponding bin in the 2D histogram. The flattened version of this 2D histogram represents the feature descriptor for the given image.




a) Shape Context Descriptor and Histogram plots for two different reference points

### 3.3. Shape Matching and Cost Computation

After computing the Shape Context descriptors, the next task is to match corresponding points between shapes and measure the similarity. This is done using various cost functions.

#### 3.3.1. Shape Context Cost

The cost between two Shape Contexts $h_i$ and $h'_j$ (corresponding to points $p_i$ and $q_j$) is computed using a $\chi^2$ distance between the histograms.

$$C_{ij}^{SC} = \frac{1}{2}\sum_{k=1}^{K}\frac{\left(h_i(k) - h'_j(k)\right)^2}{h_i(k) + h'_j(k)}$$

$K$ is the total number of bins for each possible $r$ and $\theta$, this gives the measure of how similar the spatial distributions of the points around $p_i$ and $q_j$.

#### 3.3.2. Appearance Cost

If appearance (such as intensity or texture) is available around the contour points, an additional appearance cost can be incorporated to refine the matching.

$$C_{ij}^{AC} = \|I(p_i) - I(q_j)\|_2^2$$

This term penalizes large differences in the appearance of the points.

#### 3.3.3. Thin Plate Spline Cost

For non-rigid deformations (e.g., stretching or bending), we use Thin Plate Spline (TPS) to model the transformation between shapes. TPS fits a smooth mapping from $P$ to $Q$ by minimizing both the matching error and the smoothness of the transformation.

$$C_{TPS} = \sum_{i=1}^{n}\|f(p_i) - q_i\|^2 + \lambda \times BindingEnergy(f)$$

The mapping function is,

$$f(x) = a_1 + a_2x + a_3y + \sum w_iU(\|x - p_i\|).$$

Where, $U(r) = r^2 \log r$ is the radial basis function. $\{a_i\}$ and $\{w_i\}$ are the parameters found by solving a system of linear equations.

Combining all three costs, the effective cost would be $\alpha C^{SC} + \beta C_{TPS} + \gamma C^{AC}$.

The values of $\alpha, \beta$, and $\gamma$ are taken from the paper [PDF].

#### 3.3.4. Best Correspondence

An image $I$ is considered the best match among the training set if its matching cost is lower than that of all other images in the dataset. To enhance classification efficiency, the k-Nearest Neighbours (k-NN) algorithm is employed.

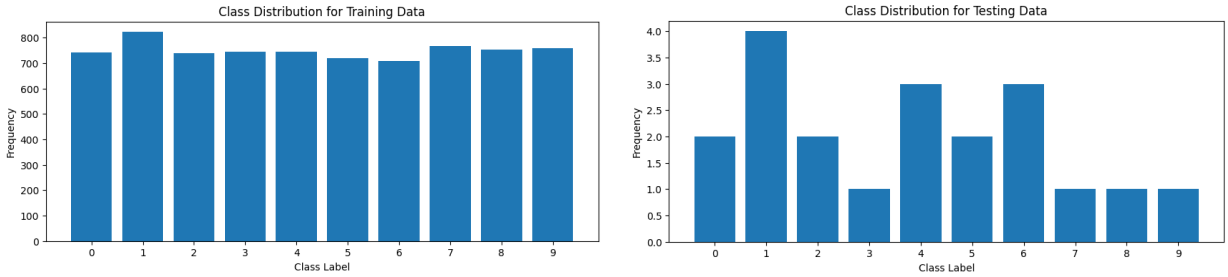
k-NN is a classification algorithm that works by comparing a new, unseen data point to the labelled examples in the training set. Specifically, KNN identifies the 'K' training points that are closest to the new point using a distance metric (typically Euclidean distance), and then makes a prediction based on the majority class of these neighbours.

Since KNN is a lazy learning algorithm, it doesn't build an explicit model during training and instead relies on storing all training data for use during prediction. As a result, when the dataset is large, computing distances to every training point for each new sample becomes computationally intensive. To mitigate this, we can limit the number of training samples used—for example, by selecting a representative subset of the training data. This significantly reduces computation time while still allowing us to classify a small number of images (like 20) from the dataset effectively, though potentially at the cost of some accuracy.

### 3.4. Model Testing

To reduce computational cost and avoid bias in KNN classification, we select one out of every eight samples from the training data. This approach helps ensure that the chosen subset maintains a fairly uniform distribution across classes, rather than being skewed toward any particular digit.

For the testing data, we choose 20 samples in total, carefully ensuring that the set includes at least one sample for each digit from 0 to 9. This guarantees that all classes are represented during evaluation, allowing for a more balanced assessment of the model's performance.




*The following report presents the accuracies obtained for different values of β and γ, along with the corresponding runtimes for computing shape context descriptors on the training data and predicting labels for the test samples.*

***Note:*** *Runtimes may differ across different systems.*

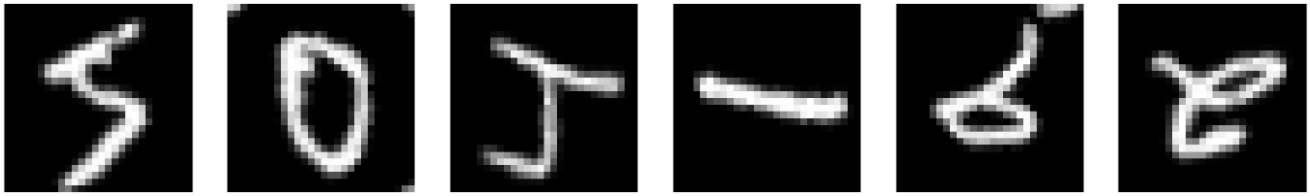
Time taken to compute the shape context descriptors for the training data: 47.23 s.

| ***β*** | ***γ*** | ***Accuracy (%)*** | ***Prediction Time (s)*** |
|---|---|---|---|
| 0.0 | 0.0 | 95.00 | 625.53 |
| 0.3 | 0.0 | 95.00 | 935.35 |
| 0.0 | 1.6 | 95.00 | 869.82 |
| 0.3 | 1.6 | 95.00 | 656.23 |

### 3.5. Model Testing under Rotation, Scaling, and Their Combination

*Original Images*

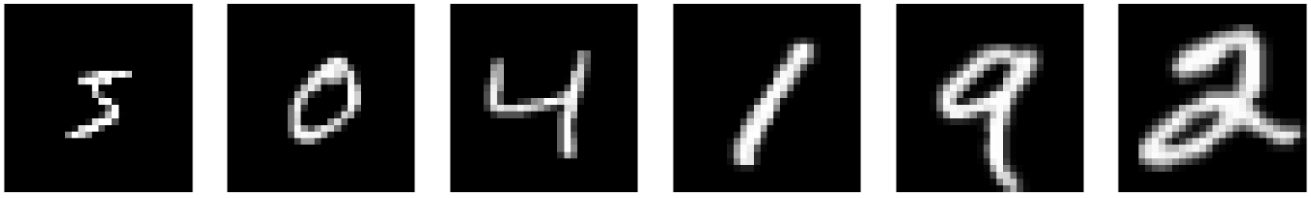
*Rotated Images (Angles: $20^0$, $50^0$, $80^0$, $110^0$, $140^0$, $170^0$)*



→ Predicted Labels: 5, 8, 5, 6, 6, 4

When the digit 4 is rotated by $80^0$, it visually resembles the digit 5 due to the extended top stroke that mimics the curved line of a 5. Similarly, rotating the digit 9 by $140^0$ makes it appear like a 6, which is a natural visual overlap and not necessarily an incorrect transformation.

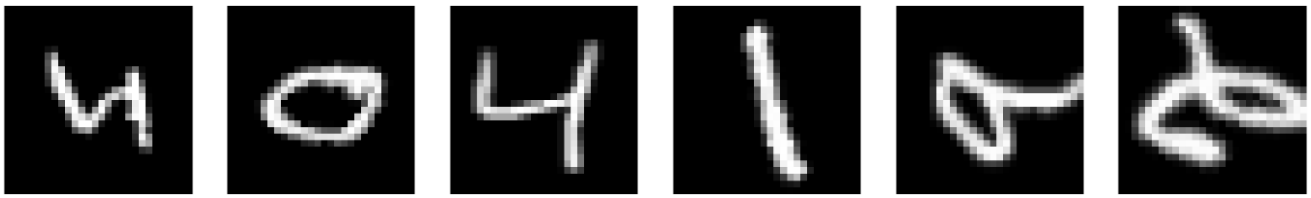
*Scaled Images (Scaling Factors: 0.5, 0.64, 0.78, 0.92, 1.06, 1.2)*



→ Predicted Labels: 3, 0, 4, 1, 9, 2

Five out of the six digits were classified correctly, but the first digit was misclassified. This error occurred for a similar reason as before — the presence of the upper stroke in the digit 5 made it difficult for the algorithm to differentiate it from a 3.

*Combinations ((Scale, Angle): (0.75, $270^0$), (0.85, $315^0$), (0.95, $0^0$), (1.05, $45^0$), (1.15, $90^0$), (1.25, $135^0$))*



→ Predicted Labels: 6, 0, 4, 1, 0, 6

The Shape Matching Algorithm is capable of identifying digits to a certain extent, but due to its high computational cost and slow prediction time, it is not well-suited for large datasets. Therefore, we now turn our attention to an alternative approach for digit recognition—*Histogram of Oriented Gradients (HOG)*.

## 4. Histogram of Oriented Gradients

It is a feature descriptor used in image processing for object detection and recognition. It captures the local structure of an image by analysing the distribution (histogram) of edge orientations (gradients) within localized regions.

Let $I$ be the given image of size $H \times W$, which is set to 28 × 28 by default. The image is first converted to grayscale to facilitate further processing. Following Gaussian smoothing, polar coordinates are calculated for each pixel, aligned with the methodology outlined in Sections 3.1.1 and 3.1.2.

### 4.1. Spatial Binning

The image obtained after performing the above process is divided into cells of size $N \times N$ (here, 4 x 4). For each cell,

- Quantize the angle $\theta$ into $B$ orientation bins (e.g., 9 bins over $0^0$ to $180^0$).
- Accumulate a histogram by adding the gradient magnitude $M(x, y)$ to the corresponding bin based on $\theta(x, y)$

Let $H_i(b)$ denote the histogram for the cell $i$, with bin $b \in \{1, 2, \dots, B\}$.

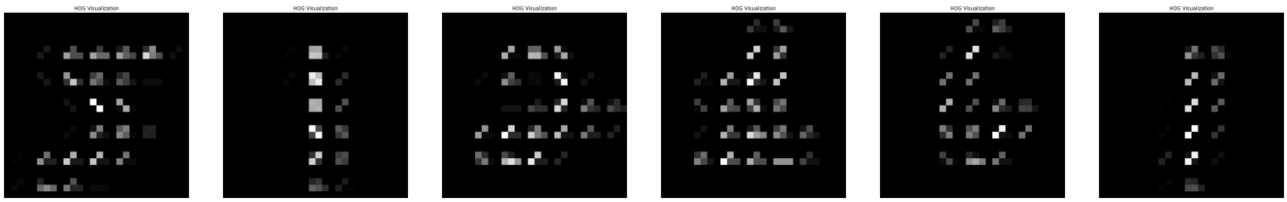
### 4.2. Block Normalization

To achieve illumination and contrast invariance. First, group neighbouring cells into overlapping blocks (e.g., 2×2 cells), then normalize the concatenated histograms within each block.

Let $v$ be the unnormalized feature vector for a block. Normalization schemes that can be used are the L2-norm or the L1-norm.

The final feature descriptor for the given image would be the concatenation of all normalized block descriptors into a single 1D feature vector.

Some examples from the training data of the HOG images have been shown below.



### 4.3. Model Testing

HOG generates feature vectors that can be fed into a classification model to predict the labels of test images. To evaluate this method, we use the same training dataset previously employed for the Shape Matching algorithm. However, given the significantly faster runtime of the HOG approach, we can afford to include one out of every eight samples from the testing data for prediction, making the evaluation more efficient. The two classification models used are a) K-Nearest Neighbours and b) Support Vector Machine.

*The following report shows the accuracy and the prediction time of both KNN and SVM on the testing data of size 1250.*

Time required to train the KNN model: 36.44 s, for the SVM model: 83.95 s.

| ***Model*** | ***Accuracy (%)*** | ***Prediction Time (s)*** |
|---|---|---|
| KNN | 95.28 | 34.45 |
| SVM | 98.32 | 48.43 |

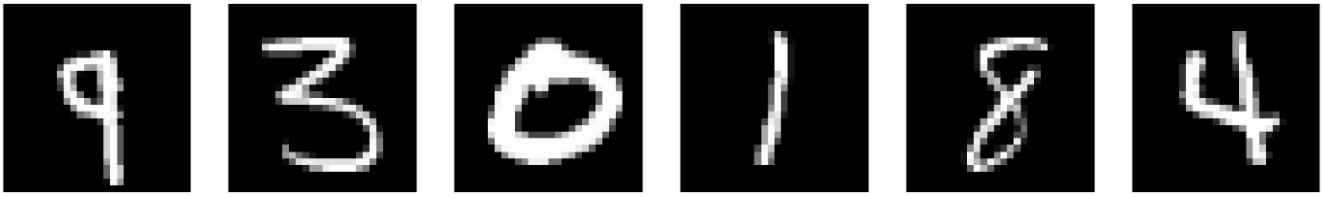
Keeping the accuracy aside, let's compare the prediction time for a single image for both methods.

- Average prediction time in case of Shape Matching = 625.53/20 =31.27 s.
- Average prediction time in case of HOG of the order = 50/1250 = 0.04 s.

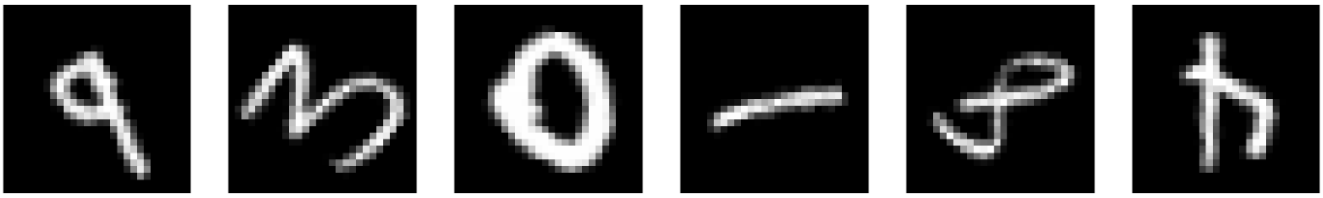
From the above analysis, we can see that prediction times differ significantly. Therefore, HOG-based feature descriptors offer a much more efficient solution for handling large datasets.

### 4.4. Model Testing under Rotation, Scaling, and Their Combination

*Original Images*

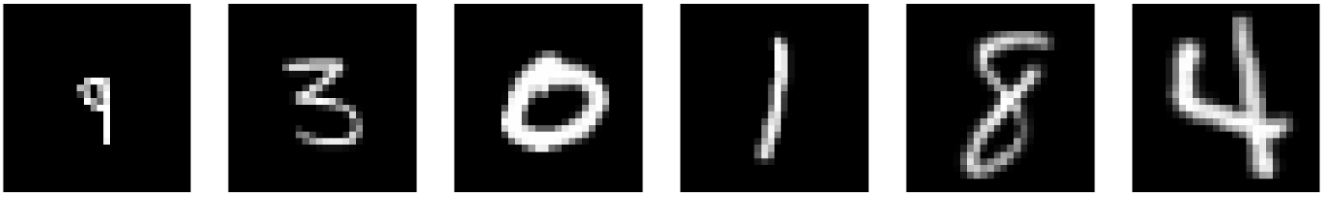


*Rotated Images (Angles: $20^0$, $50^0$, $80^0$, $110^0$, $140^0$, $170^0$)*



→ Predicted Labels: 9, 6, 0, 2, 8, 3 (KNN); 9, 8, 0, 2, 5, 3 (SVM)

*Scaled Images (Scaling Factors: 0.5, 0.64, 0.78, 0.92, 1.06, 1.2)*



→ Predicted Labels: 1, 3, 0, 1, 8, 4 (KNN); 1, 3, 0, 1, 8, 4 (SVM)

For HOG, while scaling the images yields consistent results, rotation significantly affects performance. As a result, when both rotation and scaling are applied together, the method tends to fail in making accurate predictions.

## 5. Summary of the Models

- The training time for both models is almost the same, and the accuracies were above 90%.
- The prediction times differ largely in predicting a single image; therefore, HOG can handle large datasets.
- Shape Context Descriptors are both rotationally and scale invariant, whereas HOG is not rotation invariant.

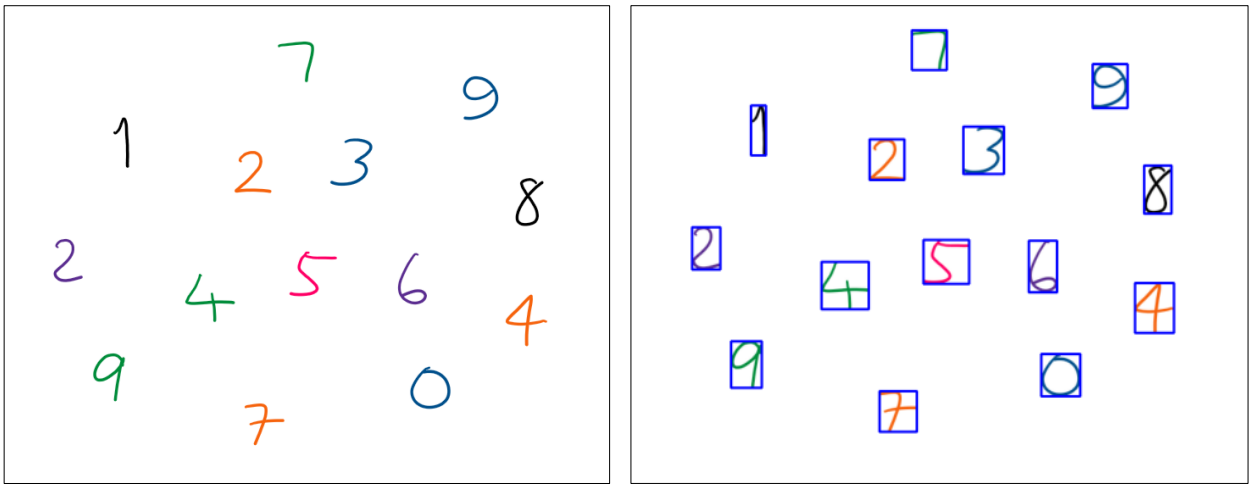
## 6. Image-Based Multi-Digit Extraction and Classification

This work focuses on developing and comparing methods for recognizing digits from images that contain multiple handwritten digits. The process begins with digit extraction, where individual digits are segmented from a composite image using preprocessing techniques such as thresholding, contour detection, and padding. Once isolated, each digit undergoes feature extraction using two key approaches: Shape Matching, which leverages structural descriptors like Shape Contexts, and Histogram of Oriented Gradients, which captures edge orientations and gradient patterns.

### 6.1. Digit Extraction

Given an image containing multiple handwritten digits (such as a scanned form), the goal of digit extraction is to isolate each digit and preprocess it into a consistent format—specifically, 28×28 grayscale images to match the MNIST dataset format.

- Convert the given image to grayscale if it is in RGB format to simplify further processing.
- The grayscale image is then smoothed using a Gaussian filter, followed by thresholding using OTSU's method to generate a binary image (pixels are either 0 or 255).
- Contours are identified in the binary image to locate individual connected regions, each typically representing a digit.
- For each contour, find the bounding rectangle, which gives the position and size of the digit in the image.
- The region inside the bounding box is cropped, padded to make it square and centred, and then resized to 28×28 pixels using interpolation to standardize input dimensions.

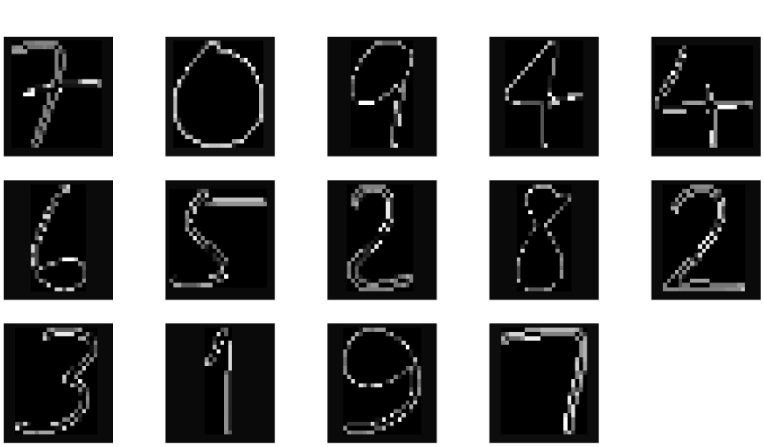


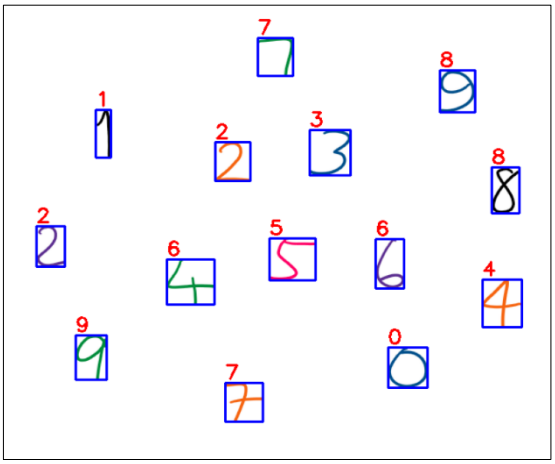
a) Original Image b) Identified digits from the original image

The detected digits are then resized and padded to obtain 28×28-pixel images, which are suitable for feeding into a prediction or classification model.

### 6.2. Shape Matching Model – To classify the digits extracted from the image

We proceed with the previously used dataset to maintain consistency and enable a clear comparison of outcomes. The images displayed below are the result of digit extraction, followed by resizing, and the image with the predicted labels.



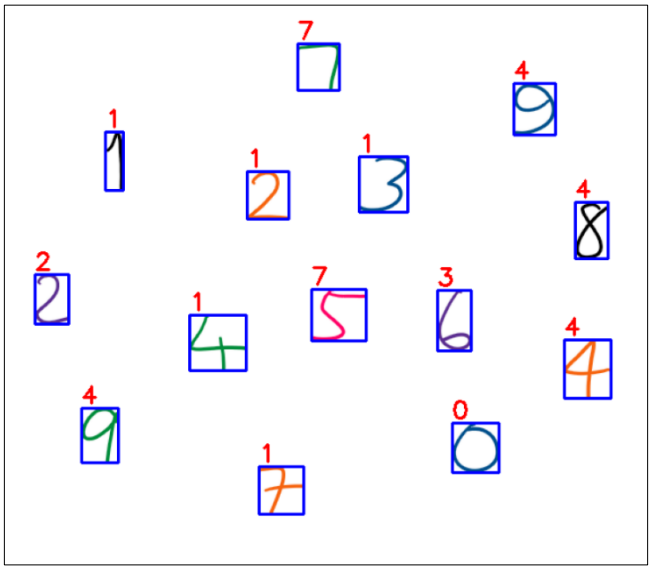


Time to compute the descriptors of the training set: 22.92 s.

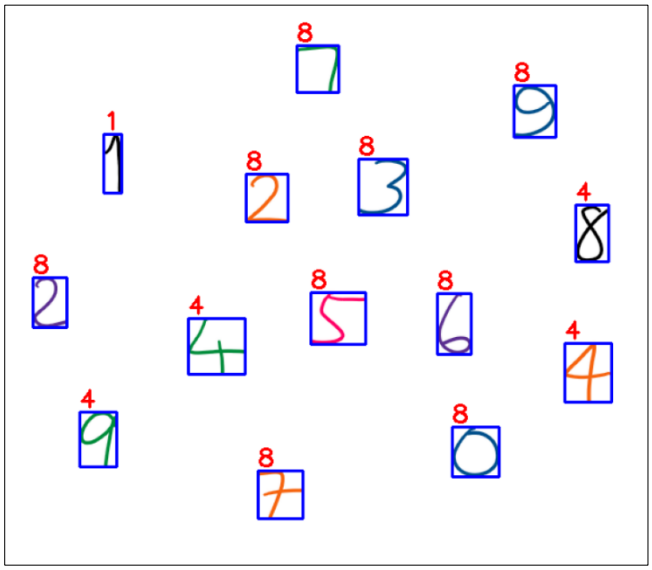
Time taken to predict the labels for the 14 digits: 135 s (For a single image, the prediction time is approximately 10 s, which is significantly higher compared to the 0.04 seconds taken per image when using the HOG-based method).

Accuracy: 12 out of 14 digits were classified correctly.

**6.3. Histogram of Oriented Gradients Model –** To classify the digits from the extracted image



a) KNN Classifier

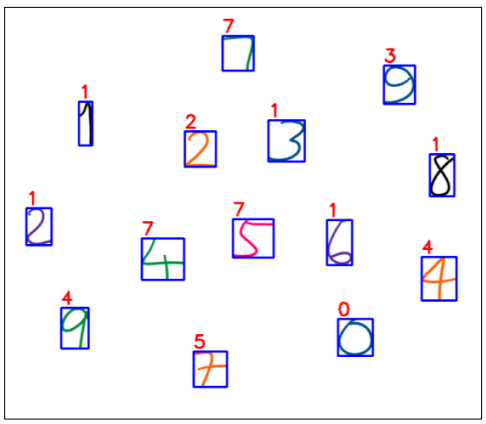


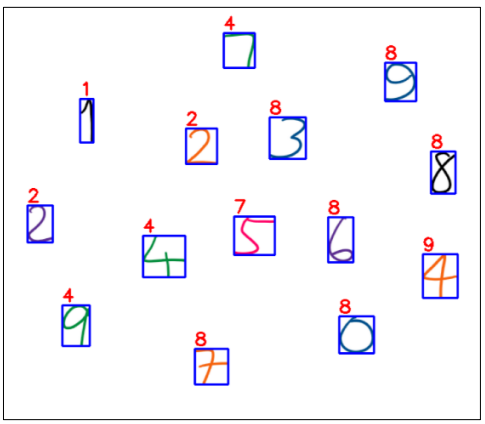
b) SVM Classifier

Training time: 57.14 s (KNN) and 87.41 s (SVM); Prediction time for 14 digits: 0.45 s (KNN) and 0.17 s (SVM)

Accuracy: Very poor, less than or equal to 5 digits were classified correctly.

**6.3.1. Modified Version –** KNN Classifier on left, SVM Classifier on right

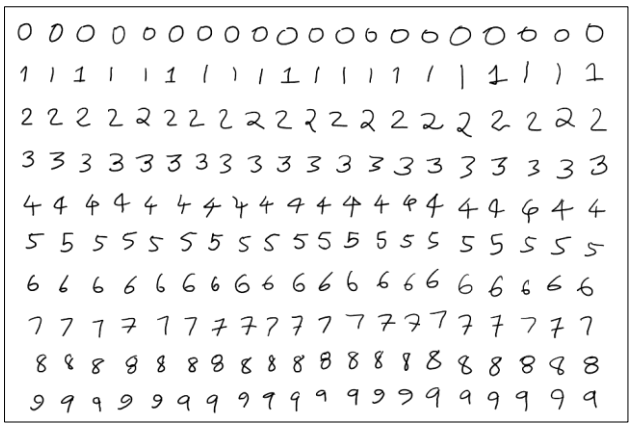




The adjustment made involved extracting contours from the edge-detected version of the image. However, this change did not lead to any significant improvement in prediction accuracy.

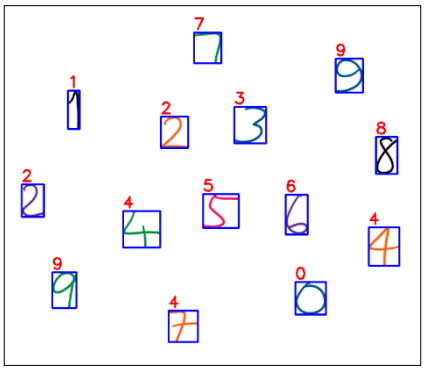
### 6.4. Modification of Training Data

Rather than relying on the MNIST dataset, we can construct our training set using a single image that contains handwritten digits from 0 to 9, each written with slight variations. These extracted digits from the image will serve as our training data. The train image is shown below.

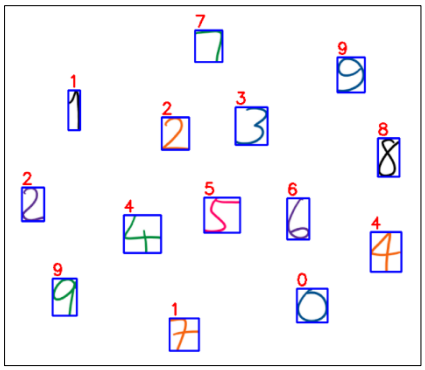


Since each digit is represented with 20 samples, the dataset maintains balance and avoids bias. Using these digits as the training set, we can now proceed to classify the digits present in the same image used in the earlier scenario.

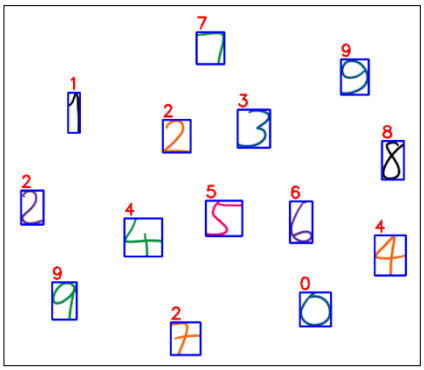
**6.4.1. Shape Matching Model, HOG (KNN), and HOG (SVM)** – Left to Right



a) 13/14 accuracy



b) 13/14 accuracy

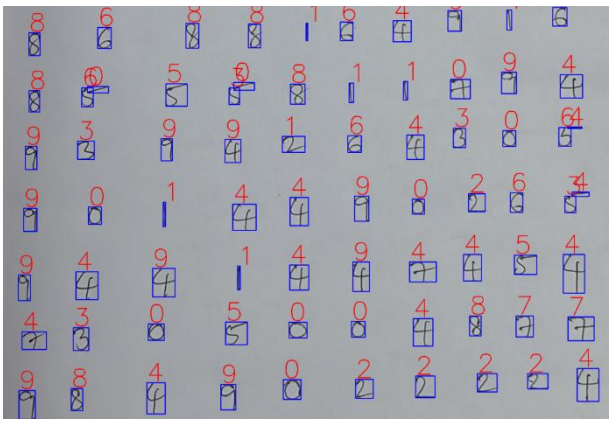


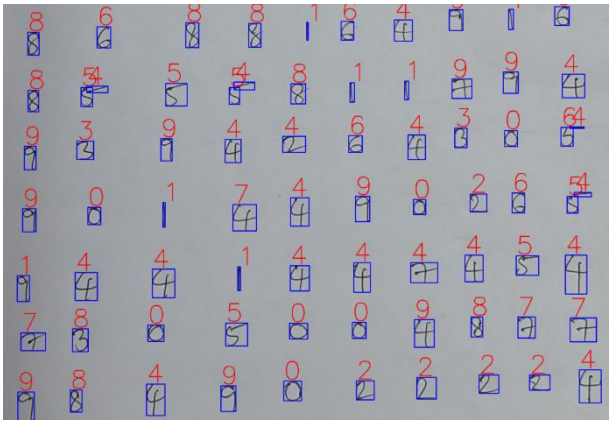
c) 13/14 accuracy

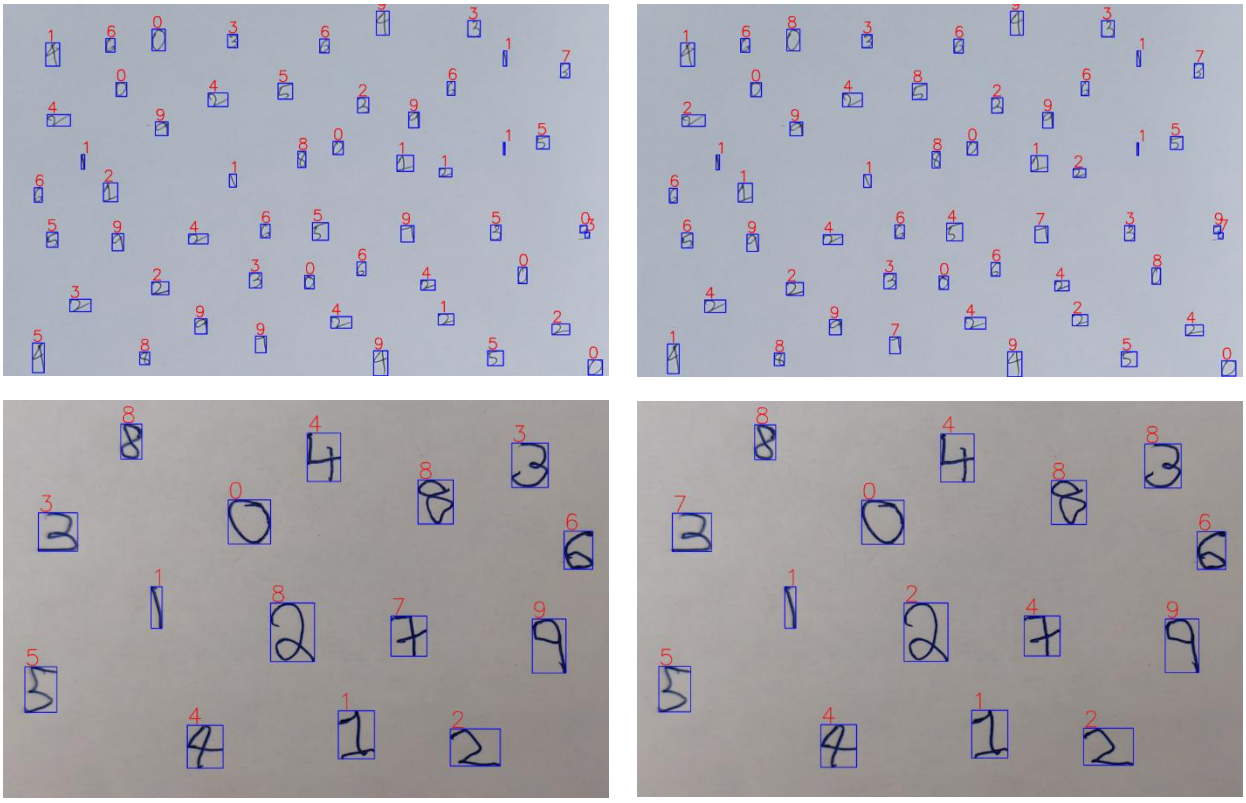
The complete process of training the model and generating predictions takes under 10 seconds in total.

*To evaluate the robustness of our models, we gathered handwritten digit images from our friends. The recognized digits and their corresponding predicted labels are displayed below.*

**6.4.2. Examples –** Shape Matching Model (Left) and HOG (SVM - Right)







While none of the examples demonstrated completely accurate classification, most digits were correctly identified. This variation in performance can largely be attributed to the nature of the training data, as model predictions are sensitive to how well the training set represents real-world variations. Therefore, it can be inferred that achieving 100% accuracy is highly unlikely, given the diversity in individual handwriting styles. Moreover, there exists a trade-off between model accuracy and prediction speed.

Let's use convolutional neural networks to recognize digits. Below is an overview of the model.

## 7. Deep Learning Model – Convolution Neural Networks

A deep CNN achieved 99.2% test accuracy on the MNIST dataset using an architecture with batch normalization and dropout regularization. Key innovations include:

- 4 convolution layers with batch normalization.
- Strategic dropout layers (0.25 – 0.4 rates).
- Early stopping with model checkpointing.

*The complete training data has been used to train the model and tested on the data with 10,000 samples.*

The epoch-wise training summary is shown in the table below.

| ***Epoch*** | ***Train Accuracy (%)*** | ***Validation Accuracy (%)*** | ***Time/Epoch (s)*** |
|---|---|---|---|
| 1 | 94.19 | 30.02 | 26.52 |
| 2 | 98.08 | 99.12 | 24.85 |
| 3 | 98.56 | 99.15 | 40.32 |
| 4 | 98.89 | 99.28 | 31.41 |
| 5 | 98.93 | 99.22 | 39.45 |
| 6 | 99.08 | 99.3 | 40.99 |
| 7 | 99.17 | 99.32 | 46.14 |
| 8 | 99.22 | 99.27 | 122.68 |
| 9 | 99.32 | 99.4 | 31.74 |

| | | | |
|---|---|---|---|
| 10 | 99.39 | 99.3 | 40.78 |
| 11 | 99.36 | 99.4 | 42.97 |
| 12 | 99.43 | 99.5 | 40.94 |
| 13 | 99.47 | 99.52 | 45.61 |
| 14 | 99.48 | 99.4 | 46.88 |
| 15 | 99.54 | 99.5 | 47.85 |
| 16 | 99.6 | 99.42 | 46.8 |
| 17 | 99.58 | 99.4 | 42.01 |

### 7.1. Model Architecture

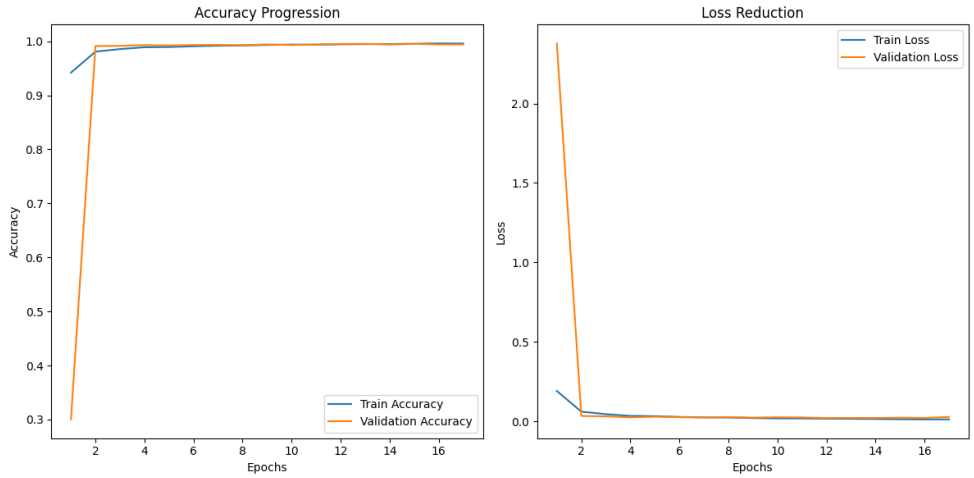
| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 26, 26, 32) | 320 |
| batch_normalization (BatchNormalization) | (None, 26, 26, 32) | 128 |
| conv2d_1 (Conv2D) | (None, 24, 24, 32) | 9,248 |
| batch_normalization_1 (BatchNormalization) | (None, 24, 24, 32) | 128 |
| max_pooling2d (MaxPooling2D) | (None, 12, 12, 32) | 0 |
| dropout (Dropout) | (None, 12, 12, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 10, 10, 64) | 18,496 |
| batch_normalization_2 (BatchNormalization) | (None, 10, 10, 64) | 256 |
| conv2d_3 (Conv2D) | (None, 8, 8, 64) | 36,928 |
| batch_normalization_3 (BatchNormalization) | (None, 8, 8, 64) | 256 |
| max_pooling2d_1 (MaxPooling2D) | (None, 4, 4, 64) | 0 |
| dropout_1 (Dropout) | (None, 4, 4, 64) | 0 |
| flatten (Flatten) | (None, 1024) | 0 |
| dense (Dense) | (None, 256) | 262,400 |
| batch_normalization_4 (BatchNormalization) (BatchNormalization) | (None, 256) | 1,024 |
| dropout_2 (Dropout) | (None, 256) | 0 |
| dense_1 (Dense) | (None, 10) | 2,570 |

### 7.2. Model Training and Hyperparameters

The following hyperparameters have been used to train the CNN model.

- Batch Size = 128, Epochs = 20, Validation Split = 10%.
- Optimizer = Adam, Loss Function = Categorical Cross Entropy.

### 7.2.1. Learning Curves




## 7.3. Model Evaluation

The accuracy and loss values have been shown in the table below for training, validating, and testing data.

| ***Metric*** | ***Training*** | ***Validation*** | ***Testing*** |
|---|---|---|---|
| Accuracy | 99.5% | 99.1% | 99.2% |
| Loss | 0.015 | 0.028 | 0.027 |

# 8. Interactive Digit Recognition Application

This project presents an interactive digit recognition system that combines a deep learning model with a user-friendly graphical interface. The application allows users to draw digits (or letters) on a virtual canvas and receive instant predictions using a pre-trained Convolutional Neural Network (CNN). The system demonstrates the practical integration of computer vision, deep learning, and user interface design.

## 8.1. System Overview

The application consists of two main components:

- ***Graphical User Interface (GUI)*:** Built using Tkinter, the GUI provides a digital canvas for drawing, as well as buttons for prediction and clearing the canvas. The interface is intuitive and responsive, designed to make the digit recognition process accessible to users of all backgrounds.
- ***Digit Recognition Model*:** The backend uses a robust CNN model, previously trained on the MNIST dataset, which is capable of recognizing handwritten digits with high accuracy. The model processes the user's drawing in real time and provides immediate feedback.

## 8.2. Workflow and User Experience

1. ***Drawing*:** The user draws a digit or letter on a 280 x 280 canvas using the mouse.
2. ***Prediction*:** Upon clicking the "Predict" button, the application processes the drawing, resizes and normalizes it to match the input requirements of the CNN, and then performs inference.
3. ***Result Display*:** The predicted digit (or letter) is displayed clearly below the canvas.
4. ***Reset*:** The "Clear" button allows the user to erase the canvas and start a new drawing.

The entire process from drawing to prediction is seamless, with predictions typically returned in under 100 milliseconds, ensuring a real-time interactive experience.

## 8.3. Image Processing and Model Inference

To ensure the drawn image is suitable for the CNN, the application performs several preprocessing steps:

- Captures the canvas content and converts it to a grayscale image.

- Inverts the image so that the digit appears white on a black background, matching the MNIST training data format.
- Resizes the image to 28 x 28 pixels using high-quality resampling to preserve stroke clarity.
- Normalizes pixel values to the 0-1 range.
- Reshapes the data to fit the model's expected input dimensions.

The processed image is fed into the CNN, which outputs a probability distribution over possible classes. The class with the highest probability is selected as the prediction.

*The prediction speed is typically under 100ms per single image, and the accuracy is more than 99%.*

### 8.4. Limitations and Future Work

- Like most handwriting recognition systems, this system might make mistakes if the writing is unclear.
- The concept can also be extended to text recognition using the EMNIST dataset, as demonstrated in the following section. In other words, the approach generalizes beyond digit recognition to full character-level recognition.

## 9. Letter and Digit Recognition using EMNIST Data

This section involves training a Convolutional Neural Network (CNN) on the EMNIST dataset to recognize handwritten letters and digits. The trained model is then used to detect and predict multiple characters present in an image. This has practical applications in optical character recognition (OCR), document digitization, and automated data entry systems. In other words, the model serves as a foundational component for systems that require accurate interpretation of handwritten or scanned textual content.

### 9.1. Dataset Description

The EMNIST By Class dataset consists of 814,255-character images, encompassing digits (0–9), uppercase letters (A–Z), and lowercase letters (a–z), resulting in a total of 62 distinct classes. Each image is a 28×28 grayscale bitmap, formatted similarly to the original MNIST dataset.

### 9.2. Model Architecture

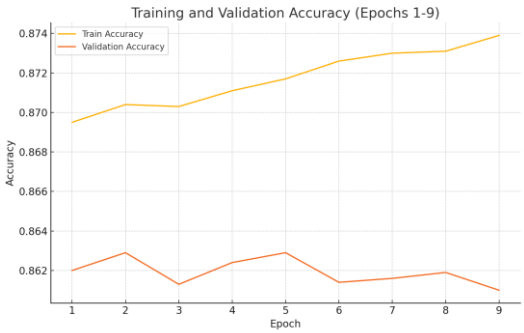
The CNN model consists of three convolutional layers followed by max-pooling, a flattening layer, and two dense layers. The final output layer has 62 nodes with SoftMax activation. The model layers are shown in the table below.

| ***Layer Type*** | ***Details*** | ***Activation*** |
|---|---|---|
| Conv2D | 32 filters, 3 x 3 kernel | ReLU |
| MaxPooling2D | 2 x 2 pool size | - |
| Conv2D | 64 filters, 3 x 3 kernel | ReLU |
| MaxPooling2D | 2 x 2 pool size | - |
| Conv2D | 64 filters, 3 x 3 kernel | ReLU |
| Flatten | - | - |
| Dense | 64 units | ReLU |
| Dense | 62 units (output) | SoftMax |

### 9.3. Training Process

The model was trained on normalized image data using the Adam optimizer and sparse categorical cross-entropy loss. Training was conducted for 20 epochs with early stopping and model checkpointing.

The figure shown below represents training and validation accuracy (Epochs 1–9) from training logs.




### 9.4. Application – Multiple Letter Recognition in an Image

The trained CNN model was evaluated on a synthetic test image containing all uppercase and lowercase English letters written in a neat handwritten style. The test image was segmented into individual character regions using contour detection and sorted spatially. Each segment was passed through the trained EMNIST classifier for prediction.

Test image with handwritten uppercase and lowercase alphabets.

Model output with bounding boxes and predicted characters.

Some misclassifications are visible, e.g., 'G' recognized as 'B' and 'j' as '$'. This highlights challenges in distinguishing visually similar characters, especially in lowercase, where the letter is not continuous.

*The model achieved approximately 87.4% training accuracy and 86.2% validation accuracy by epoch 9.*

## 10. Summary of the Project

The project initially focuses on single-digit recognition using traditional methods such as Shape Matching and Histogram of Oriented Gradients (HOG). It is then extended to multiple-digit recognition by extracting individual digits from a given image. The training dataset was augmented to include 20 samples for each digit to enhance accuracy. Finally, leveraging state-of-the-art Deep Learning techniques, specifically Convolutional Neural Networks (CNNs), an application was developed to recognize digits efficiently and quickly. This approach was subsequently extended to letter recognition as well.

## 11. Contributions

**Saketh Sai Ram:** Complete Shape Matching (3), Code for HOG (4.1 – 4.2), Multi-Digit Classification (6).

**Purushottam:** HOG Model Testing (4.3 – 4.4), Complete CNN (7), Application (8), Letter Recognition Model (9).

## 12. Reference Papers


- "Shape Matching and Object Recognition Using Shape Contexts" by S. Belongie, J. Malik, and J. Puzicha.
- "Histogram of Oriented Gradients for Human Detection" by N. Dalal and B. Triggs.