Handwritten Digit Classification Report

Rishabh Acharya[1] [*]
Ayush Pekamwar[1] [†]
Raj Nandan Singh[1] [‡]
Ankit Kumar[1] [§]
Pujit Jha[1] [¶]

[1]Indian Institute of Technology, Jodhpur
{b22cs090, b22ee084, b22ee052, b22cs076, b22cs091} @iitj.ac.in

Abstract

This report presents a methodical approach to enhancing classification accuracy on the MNIST dataset, and using it to classify handwritten digits into one of the ten decimal digits. Our approach incorporates a meticulous blend of feature extraction methods, dimensionality reduction techniques, and diverse classifiers, all strategically combined to optimize performance metrics while ensuring computational efficiency. In this project, we investigate the efficacy of two distinct feature extractors: Edge Detection and a bespoke Custom Feature Extractor, coupled with two well-established dimensionality reduction algorithms, namely Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA). Through rigorous experimentation and evaluation, incorporating advanced techniques like data augmentation to enrich training datasets, we systematically refine and validate our models based on a robust set of performance metrics.. Our findings culminate in the development of an optimized model that combines the strengths of the Custom Feature Extractor with a Random Forest classifier, further enhanced by augmented data samples. This optimized model achieved a stellar classification accuracy of 97.78%, demonstrating both the effectiveness of our approach and its applicability to real-world image classification tasks. Our findings underscore the critical role of thoughtful feature extraction, judicious model selection, and strategic data augmentation in not only improving classification outcomes but also in enhancing the robustness and generalizability of machine learning models across diverse datasets and real-world scenarios.

Keywords: MNIST, digit classification, feature extractor, data augmentation, predictions

[*]Roll Number: B22CS090
[†]Roll Number: B22EE084
[‡]Roll Number: B22EE052
[§]Roll Number: B22CS076
[¶]Roll Number: B22CS091

Contents

1 Introduction

1.1 Problem

Handwritten digit classification using the MNIST dataset for training. The aim of this project is to classify handwritten digits into one of 10 classes (each decimal digit).

1.2 Data Preprocessing

The MNIST dataset already contains separate training and testing data. We have 60,000 samples as a part of the training dataset and another 10,000 samples as a part of the testing dataset. Each training data point consists of 1-D array having 785 columns- the first column being the label and remaining 784 being the pixels. We resized the dataset to 2-D array (28x28) for better visualization on Matplotlib and to be able to use it on our custom feature extractor. The number of unique labels in both datasets was also visualized.

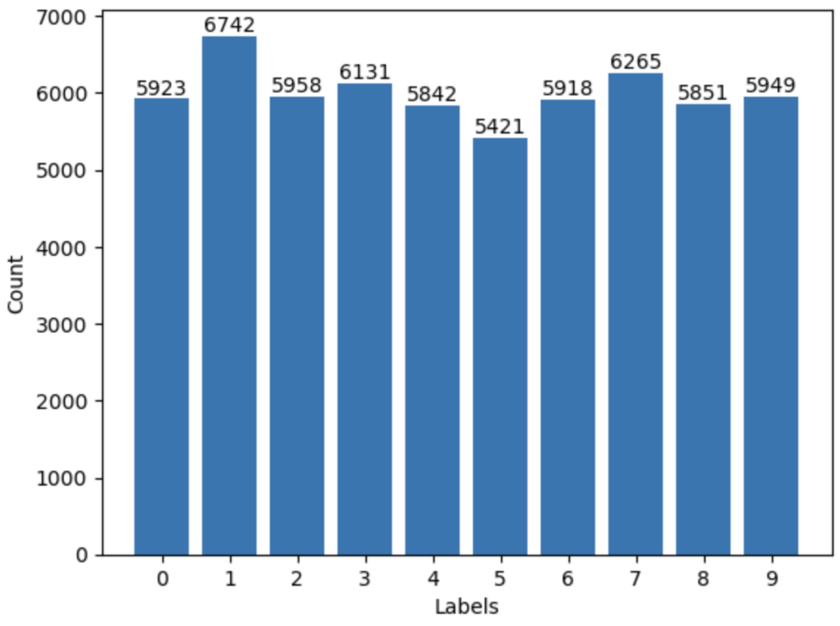


Figure 1: Unique Labels in Training Dataset

As the final step of preprocessing, we normalized the datasets.

1.3 Major Findings

- We have implemented a **custom feature extractor** (working explained later), and the accuracies using this extractor turned out to be better than those by pre-established techniques such as LDA and PCA.
- The implementation of data augmentation to generate a training dataset containing ten times the number of samples as in the original MNIST training dataset made our model much more robust and generalized, and generated a higher accuracy.

1.4 Structure of the Report

This report is structured as follows:

- **Section 2:** A discussion of various implementation ideas and the rationale behind choosing or rejecting specific approaches.
- **Section 3:** Presentation of accuracies achieved using different approaches, along with visualizations of the outputs. Additionally, we discuss training data augmentation, introduce the final selected model, and demonstrate its performance on a user-provided image.

- **Section 4:** A discussion of the interface designed for users to interact with our model, and its constraints.
- **Section 5:** A brief overview of the entire model and our methodologies.

2 Approaches Tried

2.1 Feature Extractors and Dimensionality Reduction Techniques

Using various feature extractors and dimensionality reduction techniques, we created the following 5 datasets:

2.1.1 Original dataset

This was the normalized training dataset with no additional feature extractors.

2.1.2 Dataset using PCA

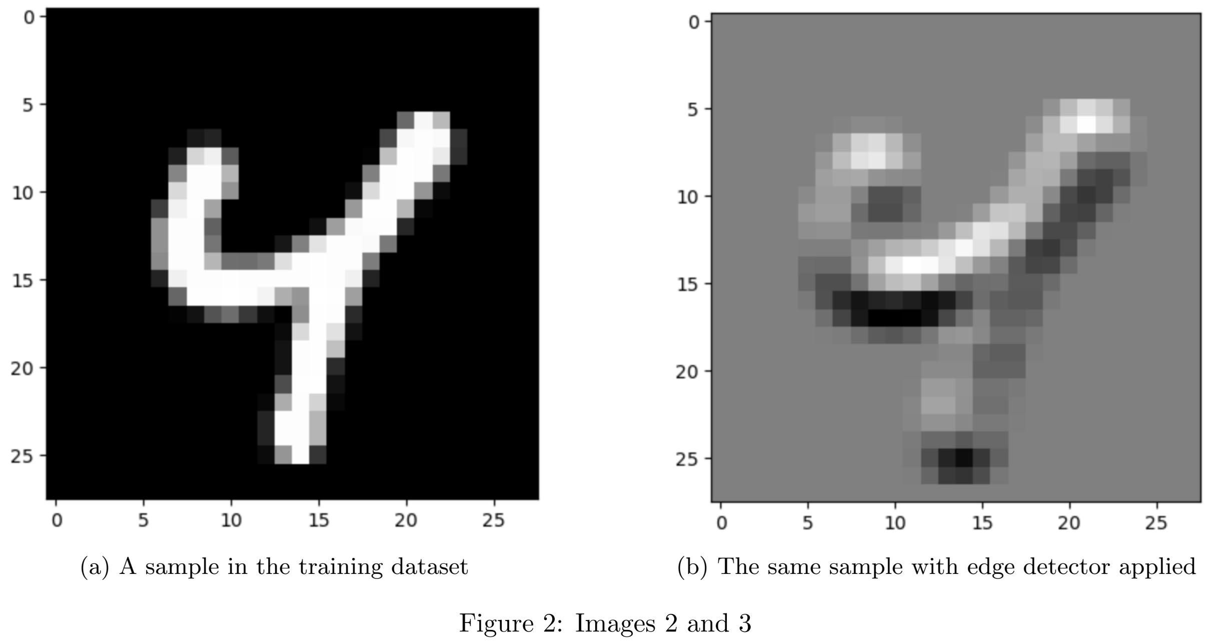
Using `n_components=0.98`, we transformed the training data into 459 components in order to achieve dimensionality reduction. We used 0.98 as the variance retention percentage was high (98%) enough to represent a reliable version of the original dataset, while not being computationally expensive.

2.1.3 Dataset using LDA

The original dataset was transformed using SK Learn's LDA.

2.1.4 Dataset using Edge Detector

Once the dataset was converted into a 28x28 array, the Prewitt operator (`prewitt_h` [18]) was used for horizontal edge detection in each image. The fundamental working of this operator, which is a single convolution operator, is that it highlights those pixels that have a strong horizontal gradient in intensity, emphasizing on transitions from dark to light or light to dark in the horizontal direction, making horizontal edges more prominent in the filtered image. For example, digits like '1', '7', and '9' typically have prominent horizontal strokes. By applying the Prewitt horizontal filter, these horizontal features are enhanced, thereby making them more prominent in the images.



(a) A sample in the training dataset (b) The same sample with edge detector applied

Figure 2: Images 2 and 3

2.1.5 Custom Feature Extractor

Pixels on the edge and in the core of the digit were given different weights. For values ranging from 1 to 199, it assigned the value 0.75, while for 200 to 255, 1 was assigned. The basic idea was that higher intensity pixels tend to correspond to the main features of the digits, while lower intensity pixels may contain noise or background information.

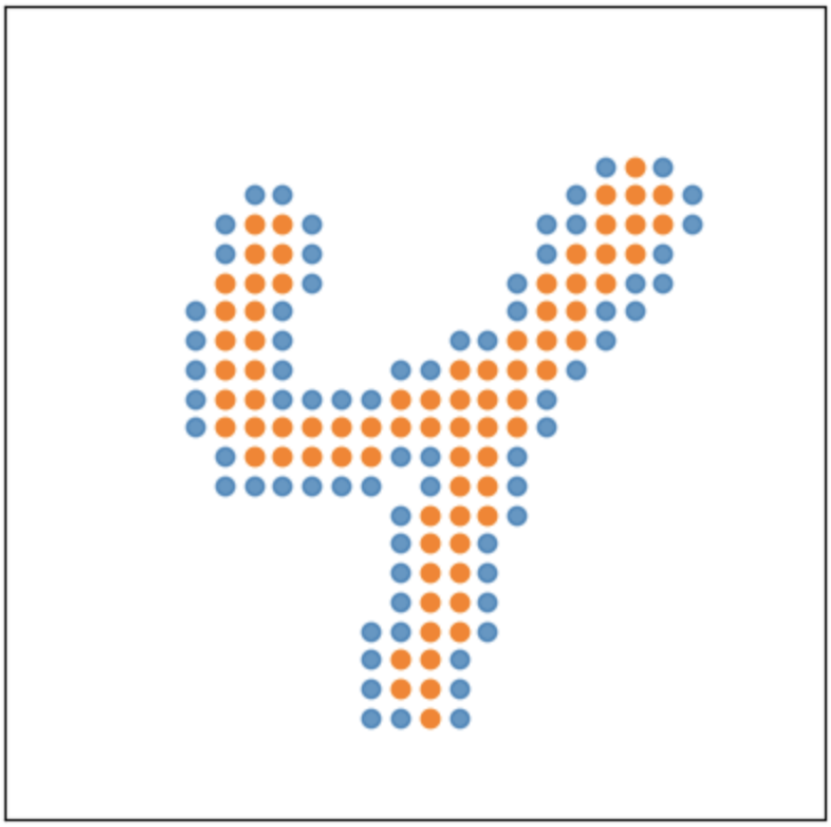


Figure 3: Visualization of the custom feature extractor

2.2 Approaches to obtain optimal model

2.2.1 KNN Classifier

The first classifier we tried was k-Nearest Neighbours [15] with k=3. We obtained the following test accuracies with it:

- Original dataset: 0.9441
- Edge detector dataset: 0.9468
- Custom feature extractor dataset: 0.9672

As we can see, we cannot reject any feature extractor using KNN as the values of obtained accuracies are nearly equal.

2.2.2 Decision Tree Classifier

- Our initial idea [4] behind using PCA was to pair it with a Decision Tree (DT) [9] classifier. This was because PCA decorrelates features and DTs may sometimes struggle with highly correlated features because they may redundantly split on similar information, leading to overfitting.
  As can be seen in figure 4, the correlation matrix has all the non-diagonal values as 0 because in PCA all the principal axes are mutually perpendicular.

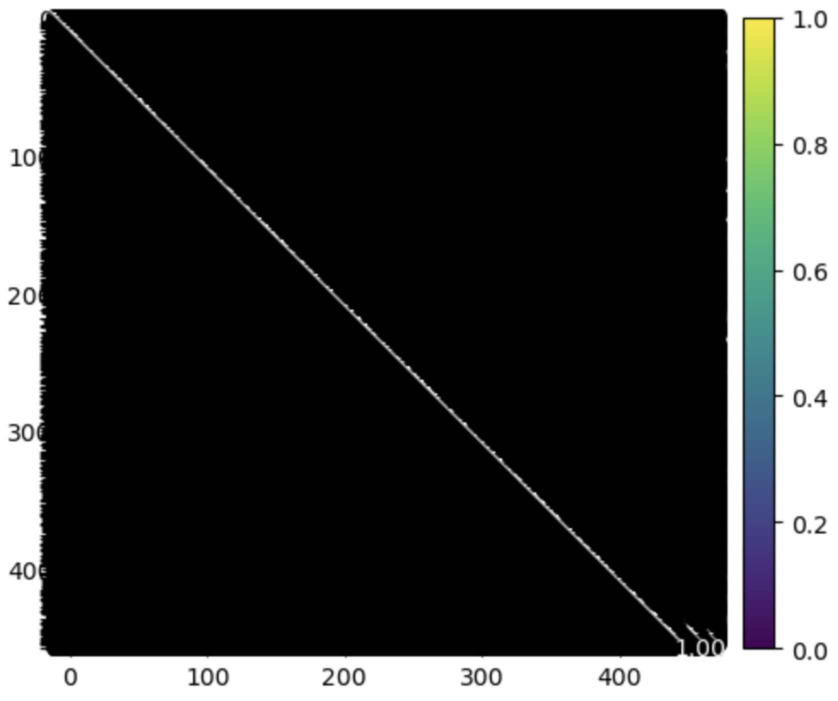


Figure 4: Correlation Matrix

However, this did not produce desirable results as we could not prevent overfitting (training accuracy=1.0). The accuracy on the test set was 0.8215.

- In order to improve on this number, we used grid search with a reduced depth of the DT (to prevent overfitting) to obtain best hyperparameters (ideal depth, which turned out to be 14, a significant change from the initial depth of 55). The test accuracy obtained in this case was 0.8333. This is only a slight improvement.

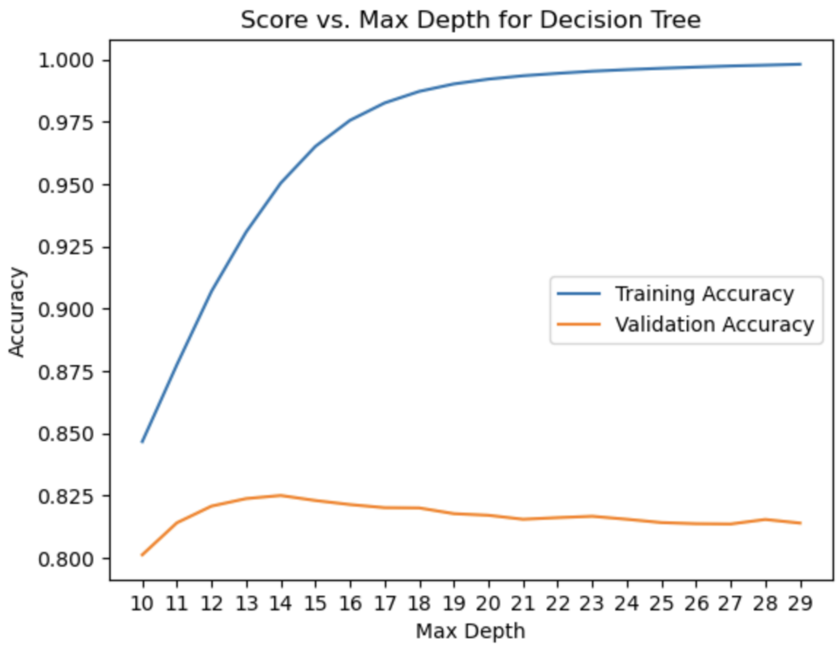


Figure 5: Score vs Max Depth for DT with PCA

- As we can see in figure 5, as the depth increases, there is clear overfitting because even though the training accuracy does rise, the validation accuracy stagnates. As a result, we rejected the combination of PCA+DT.

- To compare the performance of DT with and without PCA, we applied the classifier to our normalized dataset. The obtained accuracy was 0.882. This is not ideal either.
- Lastly, we used grid search on our custom dataset to obtain the ideal depth of the DT, which turned out to be 17, as can be seen in this figure.

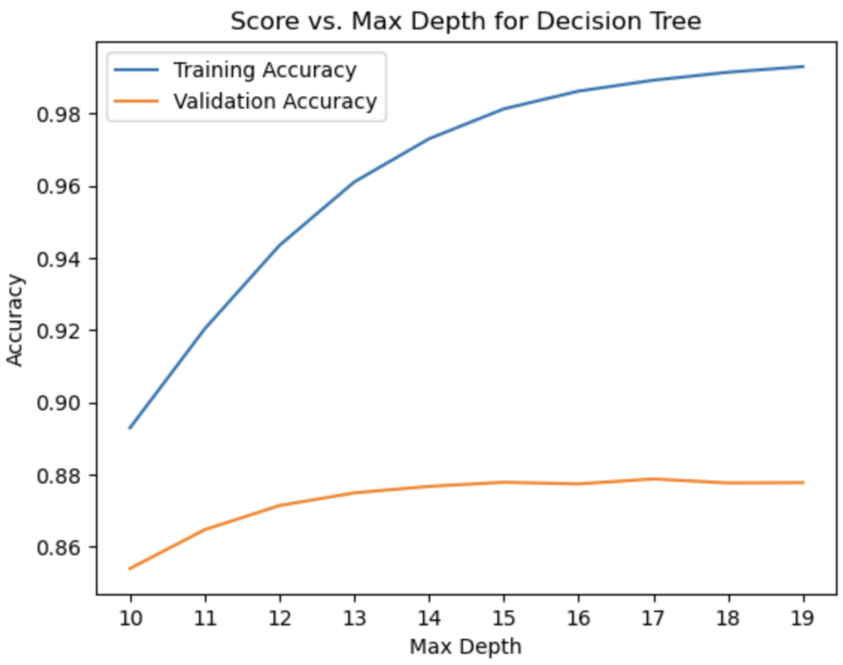
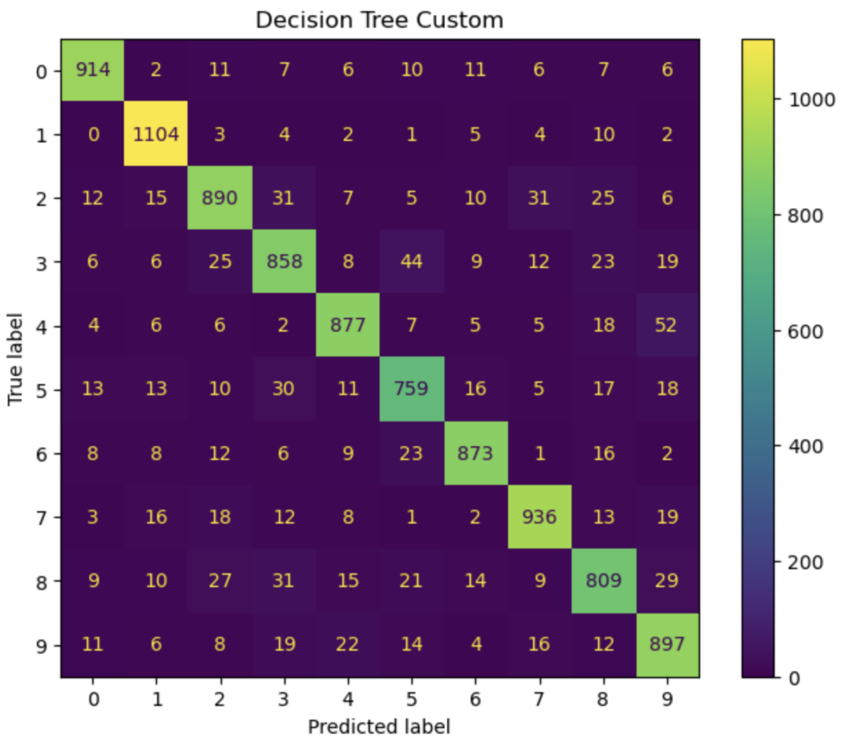


Figure 6: Score vs Max Depth for DT with Custom Feature Extractor

The accuracy of this combination was 0.8917, which, while better than other results, was still not nearly good enough.



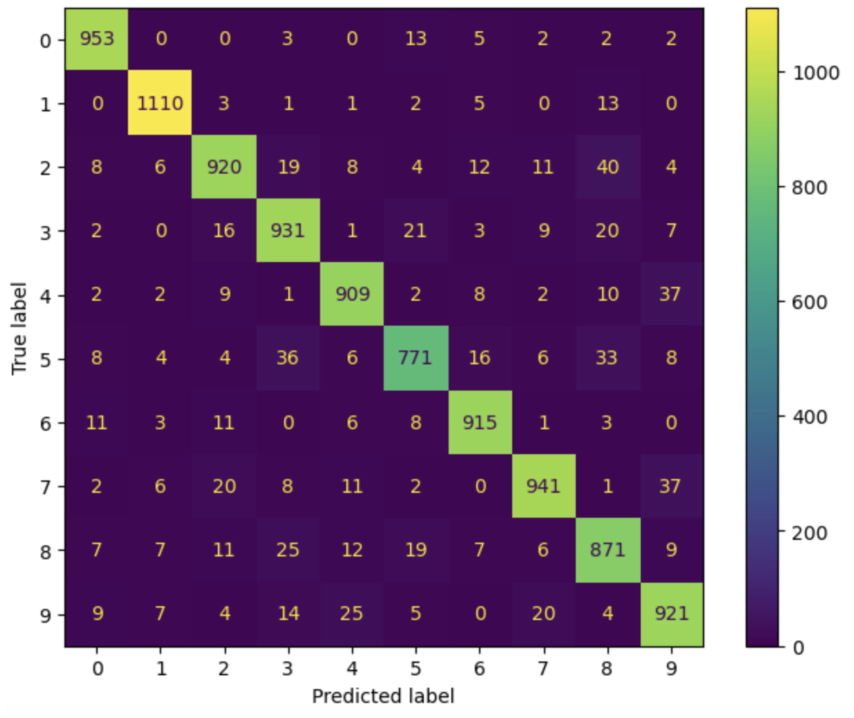
	0	1	2	3	4	5	6	7	8	9
0	914	2	11	7	6	10	11	6	7	6
1	0	1104	3	4	2	1	5	4	10	2
2	12	15	890	31	7	5	10	31	25	6
3	6	6	25	858	8	44	9	12	23	19
4	4	6	6	2	877	7	5	5	18	52
5	13	13	10	30	11	759	16	5	17	18
6	8	8	12	6	9	23	873	1	16	2
7	3	16	18	12	8	1	2	936	13	19
8	9	10	27	31	15	21	14	9	809	29
9	11	6	8	19	22	14	4	16	12	897

Figure 7: Confusion Matrix

Note: We have made similar confusion matrices for all classifiers, but only a few have been shown in this report to keep it concise.

2.2.3 Logistic Regression Classifier

- On the original normalized data, we got a testing accuracy of 0.9258.
- Edge detector feature extractor gave an abnormally low accuracy (0.1135) with logistic regression classifier [11] [17]. As a result, we rejected this feature extractor while moving forward.
- LR classifier gave a test accuracy of 0.925 on the PCA dataset. However, we rejected PCA feature extractor because:
  - The use of PCA on the DT classifier resulted in overfitting, thereby leading to undesirable accuracies. This showed that PCA is not consistent for all classifiers, and cannot be universally applied.
  - The time taken to train models on original dataset is not much longer as compared to that of PCA dataset.
- The LDA dataset was also rejected here as the dataset does not conform to the assumptions made by LDA (the features within classes are not normally distributed, and the classes do not have identical covariance matrices).
- With our custom feature extractor, LR produced a testing accuracy of 0.9242, which is higher than that by other feature extraction methods but the training time is also slightly higher.



	0	1	2	3	4	5	6	7	8	9
0	953	0	0	3	0	13	5	2	2	2
1	0	1110	3	1	1	2	5	0	13	0
2	8	6	920	19	8	4	12	11	40	4
3	2	0	16	931	1	21	3	9	20	7
4	2	2	9	1	909	2	8	2	10	37
5	8	4	4	36	6	771	16	6	33	8
6	11	3	11	0	6	8	915	1	3	0
7	2	6	20	8	11	2	0	941	1	37
8	7	7	11	25	12	19	7	6	871	9
9	9	7	4	14	25	5	0	20	4	921

Figure 8: Confusion Matrix

After using logistic regression and rejecting feature extractors based on the aforementioned reasons, we are left with these 2 datasets:

- Original dataset (just normalization)
- Dataset which uses our custom feature extractor

  These 2 datasets consistently provide great accuracy with all models. Hence, only these will be used with models which take longer time to train.

2.2.4 Naive and Multinomial Bayes Classifiers

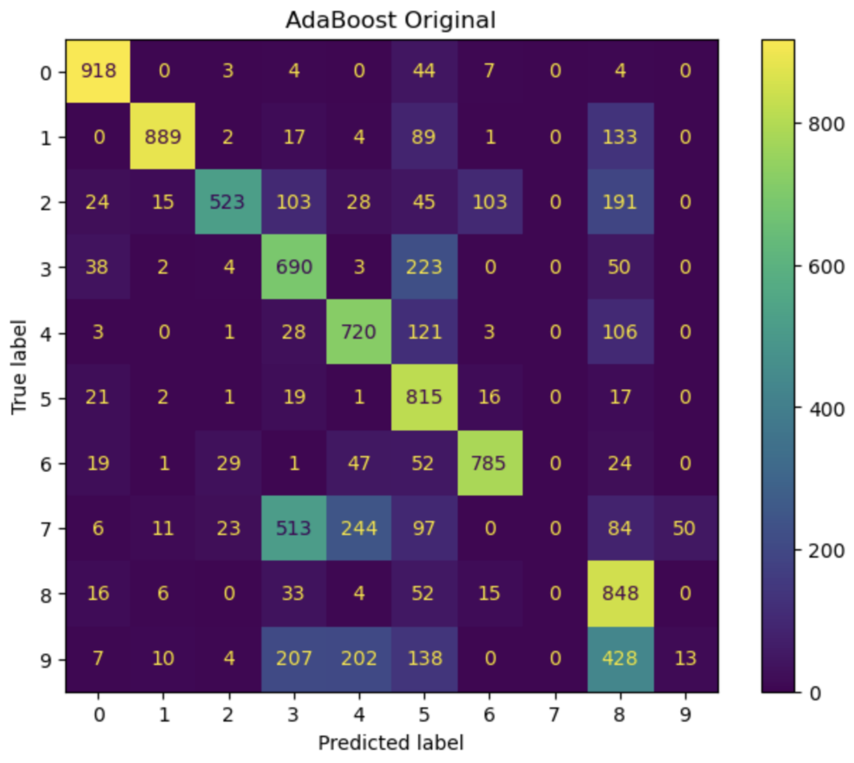
- Both Gaussian Naive Bayes [12] and Multinomial Naive Bayes Classifiers produce remarkably low accuracies. This is because the fundamental assumptions followed by these classifiers were violated in the dataset. These assumptions include:
  - Features within each class follow normal / multinomial distribution.
  - Features are conditionally independent.
- Accuracies of Gaussian Naive Bayes classifier on:
  - Original dataset: 0.5558
  - Custom dataset: 0.5484
- Accuracies of Multinomial Naive Bayes classifier on:
  - Original dataset: 0.8236
  - Custom dataset: 0.8365

2.2.5 Random Forest Classifier

- The Random Forest Classifier [13] was instantiated with the following parameters:
  - nestimators=200: This parameter sets the number of decision trees in the random forest ensemble to 200. Each tree in the forest is trained on a random subset of the training data and contributes to the final prediction through a voting or averaging mechanism.
  - bootstrap=True: When bootstrap is set to True, each decision tree in the random forest is trained on a bootstrap sample of the training data. This means that the algorithm randomly samples data points with replacement, creating multiple subsets for training each tree.
  - maxsamples=0.9: The maxsamples parameter specifies the maximum proportion of samples (data points) to be used for training each decision tree. Here, maxsamples=0.9 means that each tree is trained on a random subset of 90% of the training data, selected with replacement.
- Testing accuracy of this classifier on:
  - Original dataset: 0.9613
  - Custom dataset: 0.9718
- **Random Forest classifier has produced the best accuracy so far, with a peak of 97.18% when coupled with our custom feature extractor.**

2.2.6 AdaBoost Classifier

- AdaBoost (Adaptive Boosting) [7] is an ensemble learning algorithm designed for classification tasks. It operates by sequentially training a series of weak learners, such as decision trees, on the dataset. Each weak learner focuses on instances that previous learners classified incorrectly, adjusting their weights to prioritize these instances. The final prediction is made by combining the predictions of all weak learners, weighted by their individual accuracy. By iteratively emphasizing difficult-to-classify instances, AdaBoost aims to create a strong learner from a collection of weak learners, thereby improving classification accuracy.
- Testing accuracy of AdaBoost classifier on:
  - Original dataset: 0.6201
  - Custom dataset: 0.8542



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	918	0	3	4	0	44	7	0	4	0
1	0	889	2	17	4	89	1	0	133	0
2	24	15	523	103	28	45	103	0	191	0
3	38	2	4	690	3	223	0	0	50	0
4	3	0	1	28	720	121	3	0	106	0
5	21	2	1	19	1	815	16	0	17	0
6	19	1	29	1	47	52	785	0	24	0
7	6	11	23	513	244	97	0	0	84	50
8	16	6	0	33	4	52	15	0	848	0
9	7	10	4	207	202	138	0	0	428	13

Figure 9: Confusion Matrix

- The accuracies obtained by this classifier are significantly lower than what we have already achieved.

2.2.7 Histogram Gradient Boosting Classifier

- The Histogram Gradient Boosting Classifier [10] [8] is a variant of the Gradient Boosting [1] algorithm, optimized for large datasets. It operates by iteratively fitting decision trees to the residuals of the previous trees, gradually reducing prediction errors. Unlike traditional gradient boosting, which splits nodes based on the exact feature values, the histogram approach bins features into histograms, reducing computational complexity. The final prediction is a weighted sum of the predictions from all trees, with each tree correcting errors from previous trees. This algorithm is efficient for big data scenarios and can achieve high predictive accuracy.
- Testing accuracy of this classifier on:
  - Original dataset: 0.9452
  - Custom dataset: 0.9808
- **The accuracy of this classifier on the dataset produced by using our custom feature extractor is the highest we have achieved.**

2.2.8 Support Vector Machine (SVM) Classifier

- The RBF kernel [6] of SVM [14] classifier was used.
- Testing accuracy of RBF kernel on:
  - Original dataset: 0.9612
  - Custom dataset: 0.9791

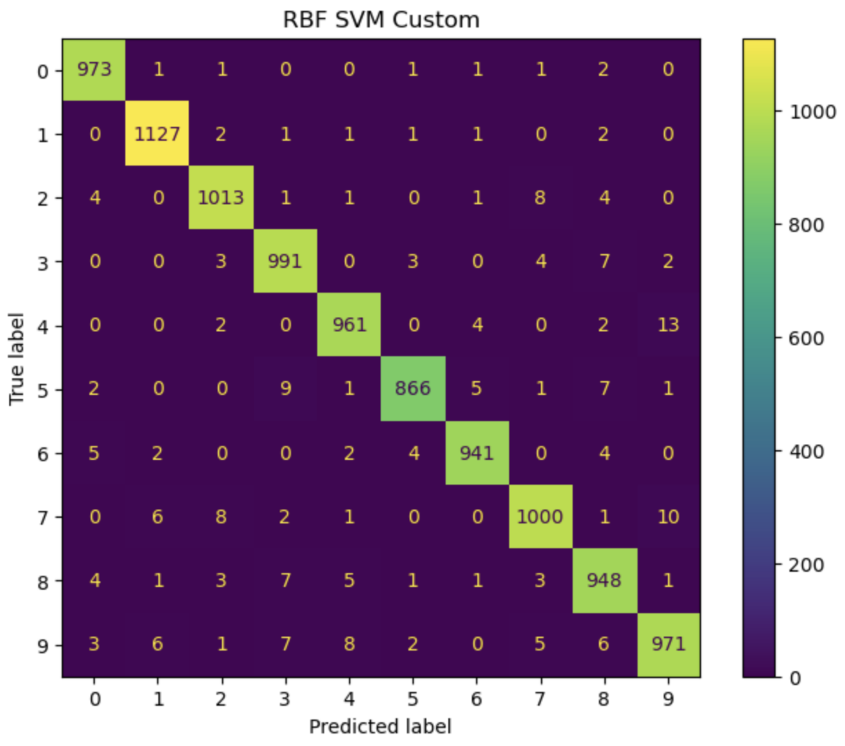


Figure 10: Confusion Matrix

- While these accuracies are appreciable, we have even better results at our disposal.

2.3 Optimal Models

- The following models produced acceptable testing accuracies:
  - Custom feature extractor + KNN: 0.9672
  - Custom feature extractor + Random Forest: 0.9718
  - Custom feature extractor + SVM Classifier (RBF Kernel): 0.9791
  - Custom feature extractor + Histogram Gradient Boost Classifier: 0.9808
- Among these, the model using SVM was rejected because the training time was too high (868.2s). Furthermore, if we are to use augmented data to increase the number of training samples, the time taken to train would significantly rise. This is not feasible for practical purposes.
- To finalize a model, we tried out ensemble voting and even augmenting the training set. This has been discussed in section 3 of this report.

3 Experiments and Results

3.1 Ensemble Voting

- Ensemble voting is a technique used in ensemble learning, where multiple individual models (e.g., classifiers or regressors) make predictions on the same dataset, and the final prediction is determined by combining or voting on the predictions of these individual models.
  The types of ensemble voting algorithms we used include:
  - Hard voting: Each individual model in the ensemble makes a prediction, and the final prediction is the class that receives the most votes (i.e., the majority of models predict that class).
  - Soft voting: Instead of making a binary decision based on a majority vote, soft voting takes into account the probability scores predicted by each model. The final prediction is often the class with the highest average probability across all models.
- We performed both of these on the final 3 models which we had obtained. The results were as follows:
  - Hard voting: 0.977
  - Soft voting: 0.9806
- As can be seen, while the final testing accuracies obtained by both types of ensemble voting were high enough to be accepted, they were not the highest values we had obtained. Hence, we discarded this concept.

3.2 Data (Image) Augmentation

- Data augmentation [5] refers to the process of artificially expanding a dataset for training machine learning models by applying various transformations to existing images.
  These are the primary objectives of image augmentation:
  - Increasing the size of the dataset: Data augmentation effectively increases the size of the training dataset by generating new images featuring variations. This often results in enhanced model performance and generalization.
  - Enhancing robustness: The inclusion of augmented images in the training data introduces diversity and variability, thereby fortifying the model against the fluctuations and noise that are inherent in real-world images.
  - Preventing overfitting: Augmentation can function as a regularization technique to mitigate overfitting by introducing variability and noise into the training data. This introduces complexity and discourages the model from retaining minute details of the training images, thereby preventing overfitting.
  - Improving generalization: Augmenting the diversity of the training dataset enables the model to acquire a greater number of generalized features and patterns, thereby enhancing its capability to process novel, unseen images during the inference process.
- Our approach to image augmentation:
  - Every image will be randomly rotated in either clockwise or counterclockwise direction within an angle of 15 degrees from the original vertical axis.
  - This rotation will be coupled with a translation in one of 8 directions (up, down, left, right, left along main diagonal, right along main diagonal, left along counter diagonal, right along counter diagonal), or no translation at all. This leads to 9 possible combinations.

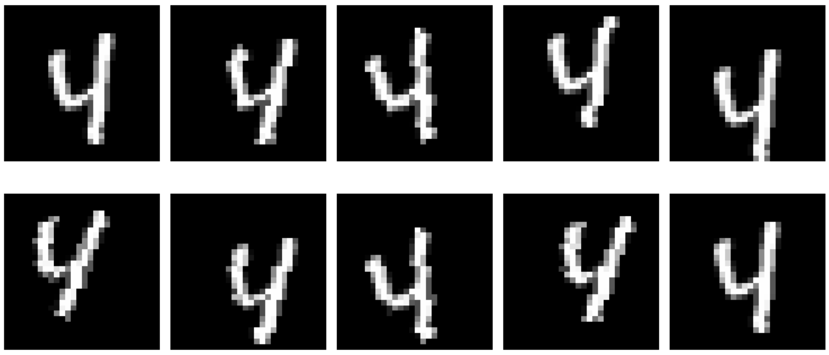


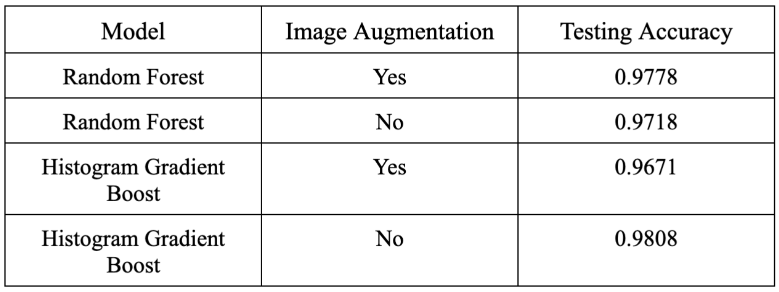
Figure 11: Variations observed in a sample after augmentation

  - Given that the size of the training dataset is 60,000 samples, this image augmentation procedure produced 5,40,000 additional samples, resulting in a total of 6,00,000 training samples.

- We had arrived at 3 optimal models at the end of section 2. Post image augmentation, we rejected the model which used KNN classifier as the memory requirement for it would be tremendously high in this new training dataset which had 10 times the number of samples as the original dataset.

- We loaded the remaining 2 models on this new training dataset and computed the testing accuracies, which turned out to be as follows:

  - Random forest model: 0.9778
  - Histogram gradient boosting model: 0.9671



Model	Image Augmentation	Testing Accuracy
Random Forest	Yes	0.9778
Random Forest	No	0.9718
Histogram Gradient Boost	Yes	0.9671
Histogram Gradient Boost	No	0.9808

Figure 12: Comparative Table

- Figure 12 summarizes the testing accuracies on the 4 models we have at hand currently.
  As we can see, the accuracy increases for the Random Forest model on performing image augmentation but it decreases for the Histogram Gradient Boost model.

- Between the 2 models producing the highest accuracies, we preferred the Random Forest model because its accuracy is post image augmentation, unlike in the case of the Histogram Gradient Boost model. This would mean that the former is more suitable for generalized inputs and is the more robust model.

Final Result: The best model we have obtained involves using our custom feature extractor along with the Random Forest classifier and performing image augmentation on the training dataset. This model produces a testing accuracy of 97.78%.

3.3 Predictions on Real Images

3.3.1 Preprocessing

- The RGB image was first converted to greyscale using the luminance method [2], which takes into account the human eye's sensitivity to different color channels.
  The formula used was: Grayscale=0.2989×Red+0.5870×Green+0.1140×Blue
- Next, the negative of the entire image was taken. This involved subtracting the value of each pixel from the maximum value across all pixels.
- Once this was done, we performed the min-max scaling in a manner such that the pixel with least intensity gets assigned the value "0" and the one with the maximum intensity gets assigned the value "255".

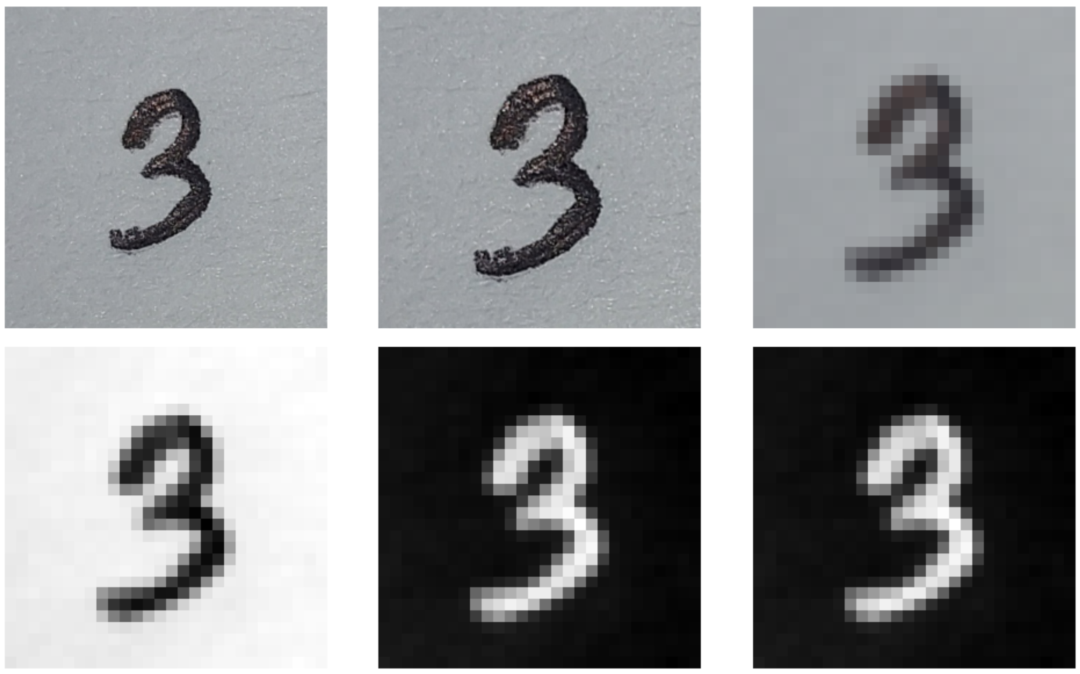


Figure 13

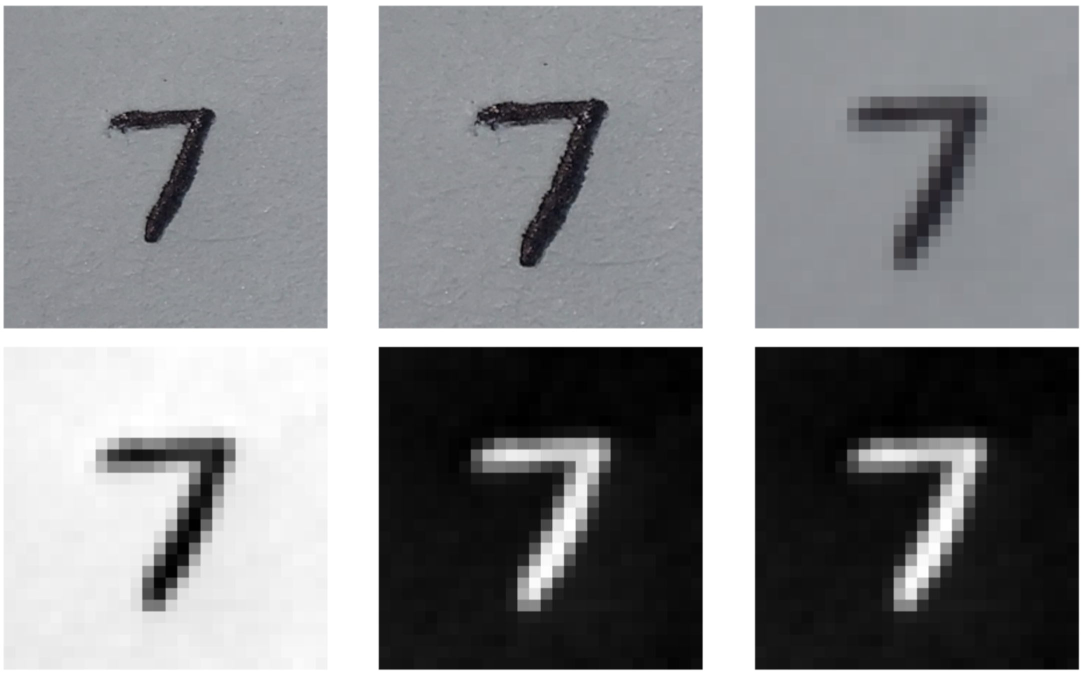


Figure 14

Each step of preprocessing is highlighted in image 13 and image 14

3.3.2 Feature Extraction

- Our custom feature extractor was deployed on the preprocessed images.

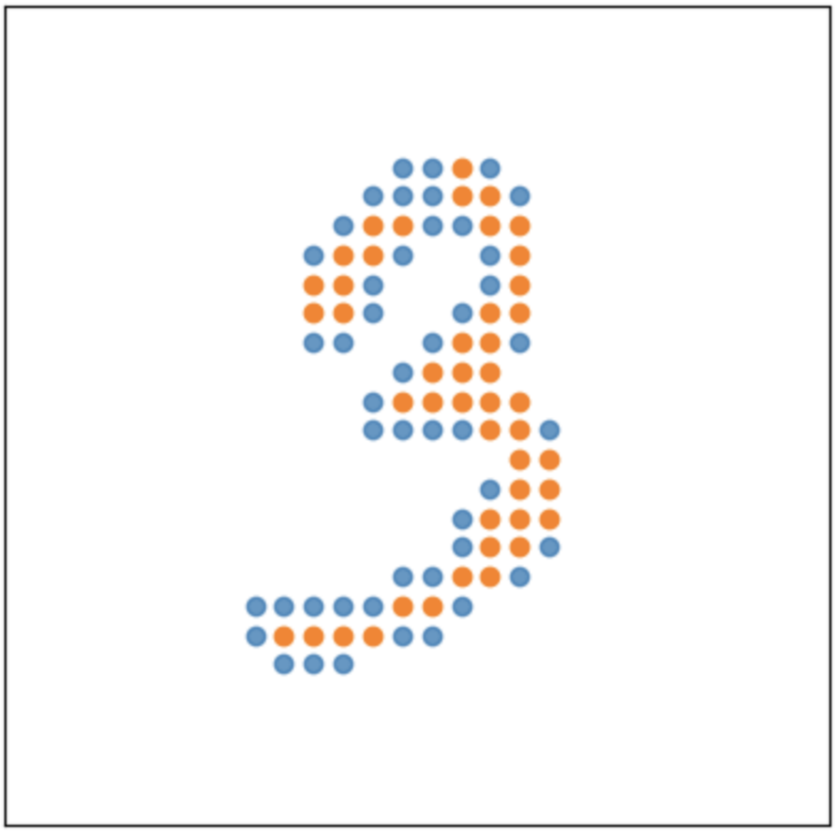


Figure 15

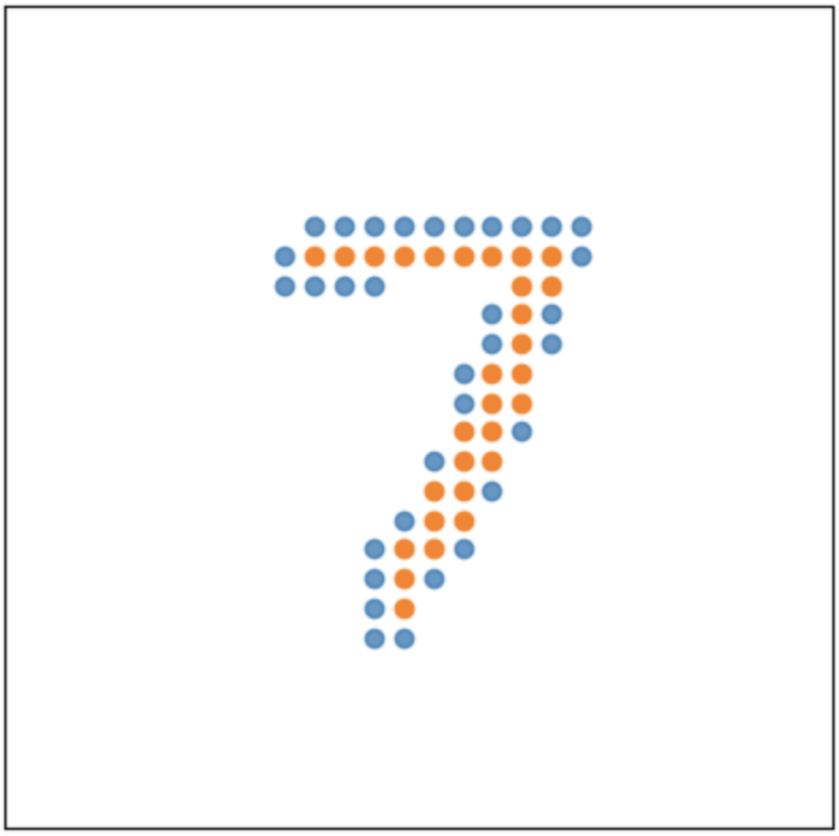


Figure 16

A visualization of the working of our feature extractor is shown in figures 15 and 16.

3.4 Predictions

- Lastly, our final model was used to make predictions on these images.
- The value with the highest probability is going to be predicted by the model.

	Figure 15	Figure 16
0	0.0166667	0.0
1	0.05666667	0.01333333
2	0.04	0.01
3	0.32	0.02666667
4	0.03666667	0.00666667
5	0.20666667	0.0
6	0.01333333	0.0
7	0.08	0.9333333
8	0.07666667	0.0
9	0.15333333	0.01

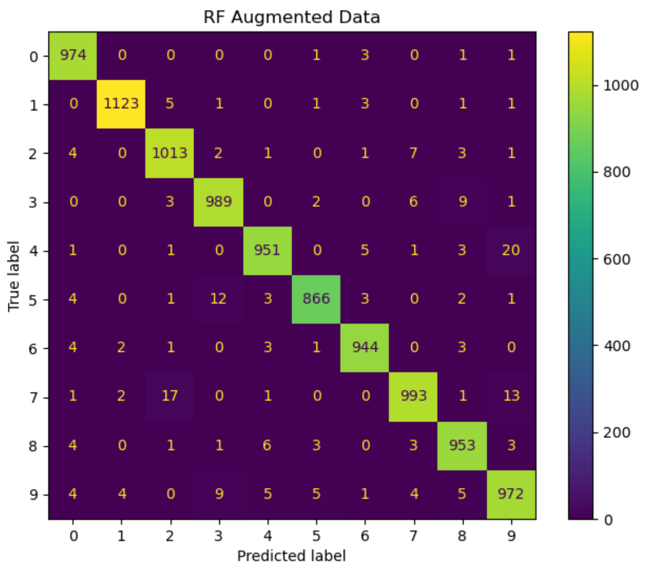
Figure 17: Probability of Each Digit

Figure 17 shows the probability of each digit when the model is deployed on figures 15 and 16.

- **The digit with the highest probability is predicted. Hence, the predictions are ”3” and ”7” respectively. Both predictions are correct.**

3.5 Failure Case Analysis

- We tried to analyze where our model is misclassifying images. To do so, we observed the model’s confusion matrix, which is shown in figure 18



	0	1	2	3	4	5	6	7	8	9
0	974	0	0	0	0	1	3	0	1	1
1	0	1123	5	1	0	1	3	0	1	1
2	4	0	1013	2	1	0	1	7	3	1
3	0	0	3	989	0	2	0	6	9	1
4	1	0	1	0	951	0	5	1	3	20
5	4	0	1	12	3	866	3	0	2	1
6	4	2	1	0	3	1	944	0	3	0
7	1	2	17	0	1	0	0	993	1	13
8	4	0	1	1	6	3	0	3	953	3
9	4	4	0	9	5	5	1	4	5	972

Figure 18: Confusion matrix of best model

- Based on the confusion matrix, we plotted the images of those categories where larger misclassifications were occurring.

- Our model could not distinguish between 5 and 3 in the images shown in figure 19, and ended up predicting them to be 3, which was incorrect. A possible explanation for this is that the images were very blurry and ambiguous, even to the naked eye (eg: the fourth and fifth images from the left in the first row look more like a ”3” even to humans) .

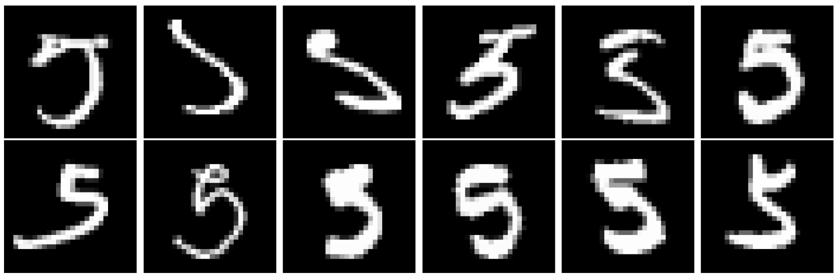


Figure 19: Samples in which 5 was misclassified as 3

- In the samples shown in figure 20, our model gave the output as 2 while it was supposed to be 7. This is perhaps due to the horizontal line in the samples, which makes it look like 2 if we exclude the lower part of the sample. This might have been confusing to the classifier.

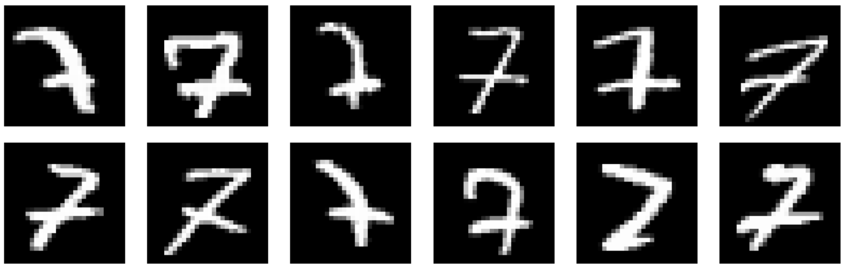
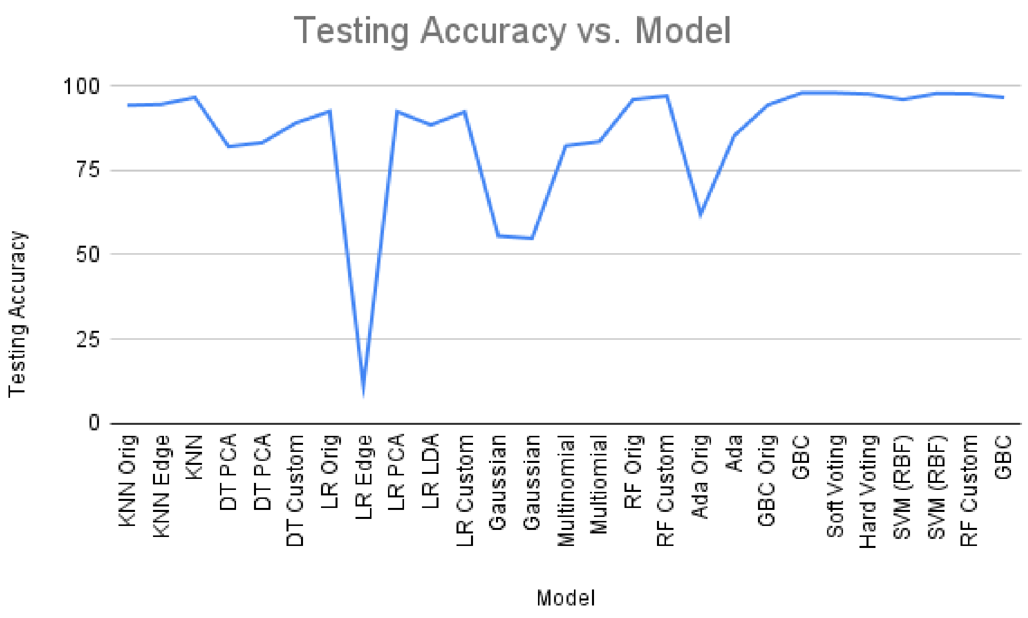


Figure 20: Samples in which 7 was misclassified as 2

- To conclude, our model primarily misclassified only those images which were unclear and ambiguous due to different handwriting styles.

3.6 Graphs and Tables



-

Figure 21: Testing Accuracy vs Model

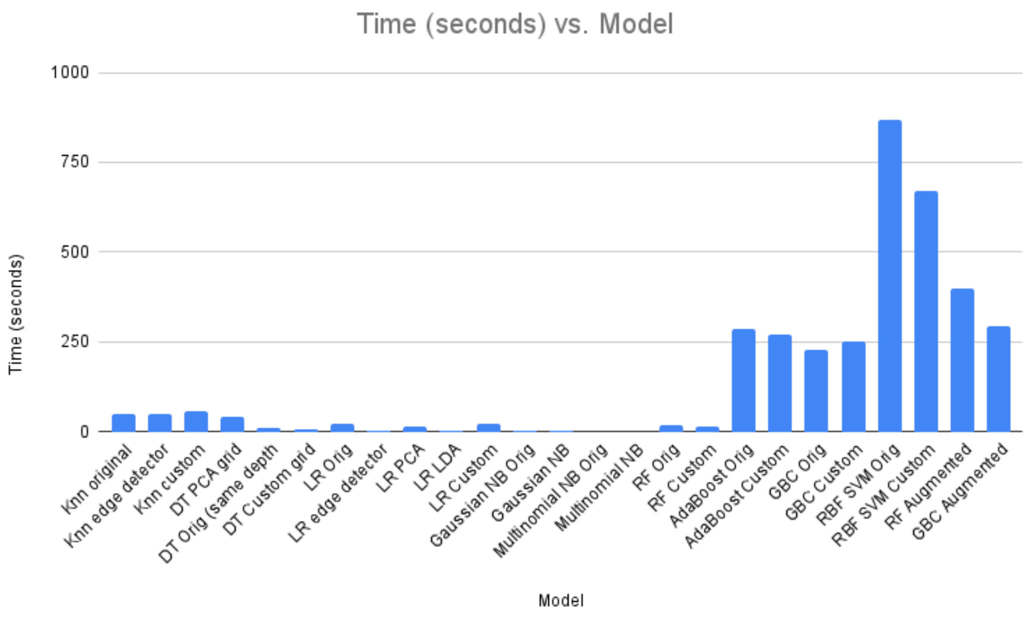
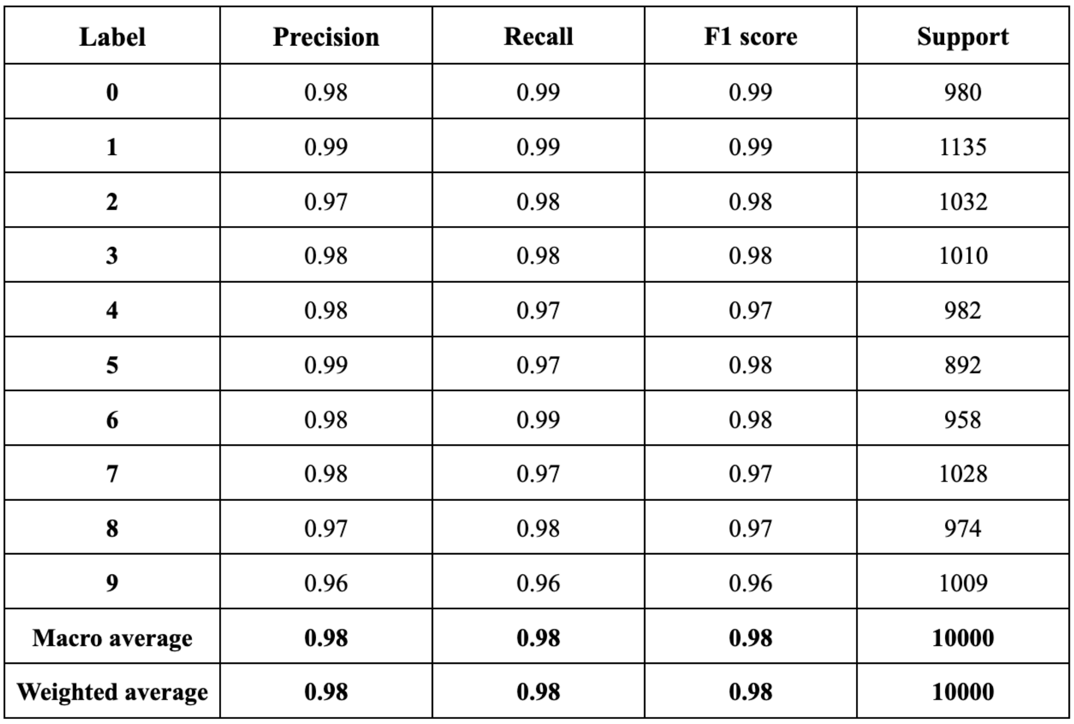
- 

Figure 22: Training Time (s) vs Model

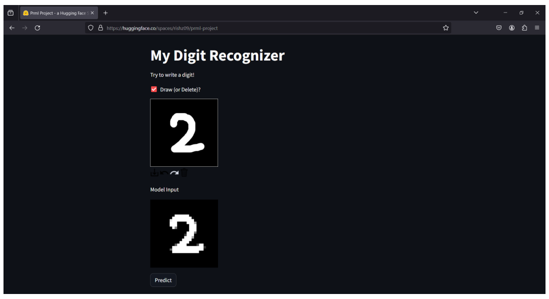
- 

Label	Precision	Recall	F1 score	Support
0	0.98	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.97	0.98	0.98	1032
3	0.98	0.98	0.98	1010
4	0.98	0.97	0.97	982
5	0.99	0.97	0.98	892
6	0.98	0.99	0.98	958
7	0.98	0.97	0.97	1028
8	0.97	0.98	0.97	974
9	0.96	0.96	0.96	1009
Macro average	**0.98**	**0.98**	**0.98**	**10000**
Weighted average	**0.98**	**0.98**	**0.98**	**10000**

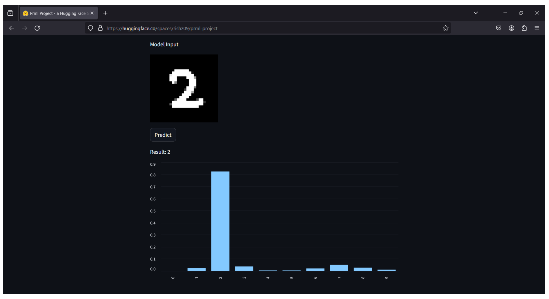
Figure 23: Classification report of best model

4 Interface

- To start off, we designed the UI using StreamLit [16]. We added a sketchpad where the user is supposed to give the input.
- The dimensions of sketchpad available to the user is 192x192. We resized the image to 28x28 to be compatible with our model, and then converted the image to grayscale. Following this, we employed our custom feature extractor as the final part of preprocessing.
- We loaded the model and used it to get the predictions.
- Finally, we used StreamLit's bar chart function to get a visualization of the probabilities of each digit.
- Once this was done, we hosted our website on Hugging Face [3].



(a) A sample in the training dataset



(b) The same sample with edge detector applied

Figure 24: Website hosted on Hugging Face

- Error case:

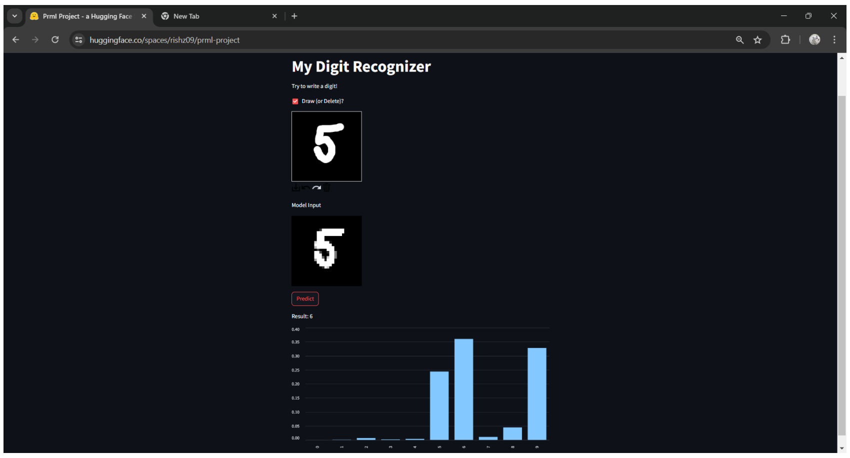


Figure 25: Misclassification

As can be seen in figure 25, 5 has been predicted to be 6 by the model due to our ambiguous drawing.

- A video showing the working of our website hosted on Hugging Face can be viewed here.

5 Summary

Brief summary of the project:

- Samples from the MNIST dataset were pre-processed initially.
- Using 4 feature extractors (LDA, PCA, Edge Detection and Custom Feature Extractor), we created 5 datasets.
- We checked the performance of various classifiers on each dataset, and then rejected different feature extractors based on either the testing accuracy scores or the fact that the hypothesis for some of the classifiers was violated in the MNIST dataset.
- Finally, we selected 4 models (combination of a feature extractor and a classifier) that were giving the best accuracies. One of these was rejected due to very high time consumption.
- To determine one best model, we tried ensemble voting (hard voting and soft voting). However, we did not obtain an improvement in the accuracy.
- Next, we performed data augmentation, and produced 5,40,000 additional training samples. Due to a very large training sample size, we were able to reject another model for its extremely high memory consumption was undesirable.
- We tested the remaining 2 models on this augmented training dataset, and compared the accuracies produced of both models with and without data augmentation. Keeping in mind the fact that augmentation makes the model more robust and generalized, we reached a final model.
- **Best model obtained: Custom feature extractor+ Random Forest classifier+ Data Augmentation. This model produced an accuracy of 97.78%.**
- Lastly, we tried this model on real images and displayed the predictions. Failure case analysis was also performed.

References

[1] DataCamp. Guide to the Gradient Boosting Algorithm. https://www.datacamp.com/tutorial/guide-to-the-gradient-boosting-algorithm, Accessed 2024.

[2] Dynamsoft. Image Processing 101: Color Space Conversion. https://www.dynamsoft.com/blog/insights/image-processing/image-processing-101-color-space-conversion/, Accessed 2024.

[3] Hugging Face. Hugging Face. https://huggingface.co/, Accessed 2024.

[4] Doruk Kilitcioglu. PCA Decision Tree. https://dorukkilitcioglu.com/2018/08/11/pca-decision-tree.html, Accessed 2024.

[5] Towards Data Science. Improving Accuracy on MNIST using Data Augmentation. https://towardsdatascience.com/improving-accuracy-on-mnist-using-data-augmentation-b5c38eb5a903, Accessed 2024.

[6] Towards Data Science. Radial Basis Function (RBF) Kernel: The Go-To Kernel. https://towardsdatascience.com/radial-basis-function-rbf-kernel-the-go-to-kernel-acf0d22c798a, Accessed 2024.

[7] scikit learn. sklearn.ensemble.AdaBoostClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html, Accessed 2024.

[8] scikit learn. Comparison of Random Forest and Hist Gradient Boosting. https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_hist_grad_boosting_comparison.html, Accessed 2024.

[9] scikit learn. sklearn.tree.DecisionTreeClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html, Accessed 2024.

[10] scikit learn. sklearn.ensemble.HistGradientBoostingClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html, Accessed 2024.

[11] scikit learn. sklearn.linear_model.LogisticRegression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html, Accessed 2024.

[12] scikit learn. sklearn Naive Bayes. https://scikit-learn.org/stable/modules/naive_bayes.html, Accessed 2024.

[13] scikit learn. sklearn.ensemble.RandomForestClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, Accessed 2024.

[14] scikit learn. sklearn.svm.SVC. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html, Accessed 2024.

[15] scikit-learn Contributors. Scikit-learn: Kneighborsclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html, 2022. Accessed on April 19, 2024.

[16] Streamlit. Streamlit. https://streamlit.io/, Accessed 2024.

[17] UCSD. Log Softmax. https://mathweb.ucsd.edu/~s1gangul/materials/log_softmax_mnist_cse251b.pdf, Accessed 2024.

[18] Vaibhav. Sobel Filter, Prewitt Filter, and the Laplacian Filter. https://medium.com/@vaibhav1403/sobel-filter-prewitt-filter-and-the-laplacian-filter-d3b8995c13, Accessed 2024.

A Contribution of each member

1. **Rishabh Acharya**: Ideation and implementation of custom feature extractor; Data Augmentation; Training best models; Designing project page
2. **Ayush Pekamwar**: Training with Decision Trees and Naive Bayes models; Designing the website which works as an interface for showing working; Making the presentation
3. **Raj Nandan Singh**: Training with Histogram Gradient Boosting and RBF SVM; Report documentation
4. **Ankit Kumar**: Brief ideation; Training with Random Forest and AdaBoost; Making the presentation
5. **Pujit Jha**: Making new datasets with different feature extractors; Training with KNN and Logistic Regression classifiers; Preparing the report; Video narrator