Bridging the Gap Between Visual Input and Textual Output

Nadia Bourne & Huu Tinh Nguyen

CS131: Processing Big Data - Tools and Techniques

05/21/25

Introduction

The topic of our project is **accurate letter prediction.** We want to, given the pixel measurements of a letter of the English alphabet, be able to accurately predict the intended letter.

This is an important facet of **optical character recognition** (OCR), a technology that uses the speed of computers to improve people's daily lives. OCR plays a crucial role in making text-based information more accessible, searchable, and useful by bridging the gap between the physical and digital worlds. Examples include scanning documents to PDFs instantly (useful for historical document preservation and everyday office tasks), serving as a key component in real-time translation by generating accurate English text quickly, and enhancing real-world text readers for the visually impaired. **Our goal is to contribute to the improvement of existing OCR technology.**

Objectives

- Develop and train multiple machine learning models for letter recognition
- Achieve 80%+ accuracy
- Evaluate models for potential real-time use by considering both runtime and accuracy

Literature Review

Optical Character Recognition, also known as OCR, has been around for a while with early efforts from people such as Frey and Slate's 1991 work using adaptive classifiers, which laid the groundwork for much, if not most, of today's research. Their paper, *Letter Recognition Using Holland-style Adaptive Classifiers* (1991), was one of the first to explore the idea of recognizing letters from pixel-based features with machine learning (ML) techniques. However, the key difference is that they worked with what were, at the time, much more limited computational resources and less mature ML tools.

This is also what makes our work different, its **modern context**: we revisited the same dataset used in their paper but re-evaluated it using **contemporary machine**

learning techniques and tools like Python, TensorFlow, and Scikit-Learn. We also integrated **command-line processing tools** as a core part of our preprocessing pipeline, which adds transparency and efficiency to the data handling process.

Furthermore, modern advancements in cross-validation, neural network training, and ensemble learning (like Random Forests) allow us to achieve much higher accuracy, up to **98%**, compared to the ~80% in their original paper. Our work demonstrates how technological progress in both algorithms and hardware has allowed us to push past previous limits and revisit older problems with new solutions.

Overall, our project exists because OCR is still widely used today in apps for the visually impaired, document digitization, and real-time translation. Our goal was to contribute to that ongoing progress, showing that even basic models can be extremely effective with proper tuning and clean data.

Understanding & Preparing the Dataset

Dataset Description

The dataset contains pixel measurements of a processed image of a capital letter of the English alphabet (A-Z). The rectangular image is referred to as 'box' in the feature descriptions. There are 17 features, and a total of 20000 instances.

The first feature is the label, which is categorial and contains the 26 letters of the alphabet. The remaining 16 are various pixel values derived from randomly distorted images of the letters in 20 different fonts (example images shown in Figure 1 below). The integer values given in the 16 features are continuous and scaled up from this data to fit in a range of 0-15.

Feature names and descriptions (taken from dataset):

1. lettr — capital letter (26 values from A to Z)
2. x-box — horizontal position of box (integer)
3. y-box — vertical position of box (integer)
4. width — width of box (integer)

5. high — height of box (integer)
6. onpix — total # on pixels (integer)
7. x-bar — mean x of on pixels in box (integer)
8. y-bar — mean y of on pixels in box (integer)
9. x2bar — mean x variance (integer)
10. y2bar — mean y variance (integer)
11. xybar — mean x y correlation (integer)
12. x2ybr — mean of x * x * y (integer)
13. xy2br — mean of x * y * y (integer)
14. x-ege — mean edge count left to right (integer)
15. xegvy — correlation of x-ege with y (integer)
16. y-ege — mean edge count bottom to top (integer)
17. yegvx — correlation of y-ege with x (integer)

Figure 1. Example of warped text. Screenshot from Frey and Slate, *Letter Recognition Using Holland-style Adaptive Classifiers* (1991).

Data Processing with Linux Command-Line

The first step we took was to **relabel** the label column. We wanted to replace every letter with an integer between 0 and 25, with 'A' being 0, 'B' being 1, etc. To do this, we wrote a function to transform the label string in the first column into an integer by using the ASCII values of capital letters. Since they are all consecutive (starting at 65 for 'A'), we iterated through the file and replaced the letters one after the other with the appropriate integer.

Code

```
relabel() {
        local infile=$1
        local outfile=$2
        awk -F, 'BEGIN {
                FS = OFS = ",";
                 for (i = 0; i < 26; i++) {
                        letter = sprintf("%c", i + 65);
                        map[letter] = i;
                }
        }
        {
                if ($1 in map) {
                        $1 = map[$1];
                }
        print;
        }' "$infile" > "$outfile"
}

relabel letter-recognition.data relabeled-lr.data
```

Letter-Labeled	Relabeled
T,2,8,3,5,1,8,13,0,6,6,10,8,0,8,0,8 I,5,12,3,7,2,10,5,5,4,13,3,9,2,8,4,10 D,4,11,6,8,6,10,6,2,6,10,3,7,3,7,3,9 N,7,11,6,6,3,5,9,4,6,4,4,10,6,10,2,8 G,2,1,3,1,1,8,6,6,6,6,5,9,1,7,5,10 S,4,11,5,8,3,8,8,6,9,5,6,6,0,8,9,7	19,2,8,3,5,1,8,13,0,6,6,10,8,0,8,0,8 8,5,12,3,7,2,10,5,5,4,13,3,9,2,8,4,10 3,4,11,6,8,6,10,6,2,6,10,3,7,3,7,3,9 13,7,11,6,6,3,5,9,4,6,4,4,10,6,10,2,8 6,2,1,3,1,1,8,6,6,6,6,5,9,1,7,5,10 18,4,11,5,8,3,8,8,6,9,5,6,6,0,8,9,7

(Note: Code has been spaced for readability.)

Next, we checked the **class distribution** with:

cut -d, -f1 relabeled-lr.data | sort | uniq -c	
789 0 766 1 739 10 761 11 792 12 783 13 753 14 803 15 783 16 758 17 748 18 796 19 736 2	813 20 764 21 752 22 787 23 786 24 734 25 805 3 768 4 775 5 773 6 734 7 755 8 747 9

(Note: The right column is the label column [now integers instead of letters], and the left column is the number of data points per letter.)

The dataset is clearly quite balanced. With this in mind, we next checked for **duplicates** with:

```
sort relabeled-lr.data | uniq -d | wc -l
```

Our set had **845 unique duplicate rows**. We decided to **remove** them and save the deduplicated dataset separately, as removing a lot of rows might hinder our classification models by unbalancing the data. By saving it separately, we could run the models on both datasets and see which helped the models perform the best.

Code	
sort relabeled-lr.data | uniq > deduped-lr.data	
Relabeled	**Relabeled & Deduplicated**
19,2,8,3,5,1,8,13,0,6,6,10,8,0,8,0,8 8,5,12,3,7,2,10,5,5,4,13,3,9,2,8,4,10 3,4,11,6,8,6,10,6,2,6,10,3,7,3,7,3,9 13,7,11,6,6,3,5,9,4,6,4,4,10,6,10,2,8 6,2,1,3,1,1,8,6,6,6,6,5,9,1,7,5,10 18,4,11,5,8,3,8,8,6,9,5,6,6,0,8,9,7	0,1,0,2,0,0,7,3,2,0,7,2,8,2,6,1,8 0,1,0,2,0,0,7,4,2,0,7,2,8,1,6,1,8 0,1,0,2,0,0,7,4,2,0,7,2,8,1,7,1,8 0,1,0,2,0,0,7,4,2,0,7,2,8,2,6,1,8 0,1,0,2,0,0,7,4,2,0,7,2,8,2,7,1,8 0,1,0,2,0,0,8,3,2,0,7,2,8,2,6,1,8
Rows: 20000	**Rows:** 18668 (1332 rows removed)

(Note: The deduplicated dataset was sorted to remove the duplicates, which is why it does not have the same label column values.)

Finally, we later came back to double-check for any missing values with:

tail -n +2 relabeled-lr.data | grep -Ec '^,|,,|,$'
tail -n +2 deduped-lr.data | grep -Ec '^,|,,|,$'

Which confirmed there weren't any.

Models: Details & Implementation

Given the multiclass nature of the letter recognition task, we implemented three models: **Random Forest, Multinomial Logistic Regression,** and a **Multilayer Perceptron** (MLP). All three models used **5-fold cross-validation** to help ensure consistency and reliability by testing on multiple subsets of the data. This reduces the risk of overfitting and gives a better estimate of how the model will perform on unseen data. Our MLP model is implemented using TensorFlow, while the Random Forest and Multinomial Logistic Regression models use Scikit-Learn.

The dataset was divided using a **70:20:10 split**, with 70% used for training, 20% for validation, and 10% for testing. We wanted to try to match Frey and Slate's 80:20 train-test split to more directly compare approaches.

Our main metric was **accuracy**, because the true accuracy of the model is extremely important. The goal of the project is to identify the scanned text (given pixel data) as the correct letter with little to no error, given we want to avoid typos in scanned documents or translated text. However, accuracy alone can be misleading, especially if certain letters are less frequent or more difficult to classify. That's why we also want to calculate **precision, recall,** and the **F1-score**, to best illustrate the model's ability to correctly identify each letter.

Random Forest

We chose Random Forest for its strength in capturing nonlinear patterns and interactions between features. Below is the build function which utilizes GridSearchCV to automatically tune the listed hyperparameters and find what makes the best performing model.

Hyperparameters	Model

n_estimators [100, 500, 750]: *number of trees in the forest*
 	
max_depth [5, 10, 15]: *maximum depth of each tree*
	`def build_rf_GSCV():`
	`  model = GridSearchCV(`
	`  estimator=RandomForestClassifier(random_state=1),`
	`  param_grid={`
	`  'n_estimators': [100, 500, 750],`
	`  'max_depth': [5, 10, 15]`
	`  },`
	`  cv=5,`
	`  scoring={`
	`  'accuracy': 'accuracy',`
	`  'precision': make_scorer(precision_score, average='macro', zero_division=0),`
	`  'recall': make_scorer(recall_score, average='macro', zero_division=0),`
	`  'f1': make_scorer(f1_score, average='macro', zero_division=0)`
	`  },`
	`  refit='accuracy',`
	`  verbose=0,`
	`  return_train_score=True`
	`  )`
	`  return model`

Multinomial Logistic Regression

We included Multinomial Logistic Regression as a baseline model due to its simplicity and ability to handle multiclass classification, allowing us to compare its performance with more complex models. Like the Random Forest, we used GridSearchCV to automatically tune hyperparameters, as shown below.

Hyperparameters	Model

C [0.01, 0.1, 1, 10]: *regularization strength. Smaller values lead to simpler models while larger values lead to more complex models*	`def build_mlr_GSCV():`
	`  model = GridSearchCV(`
	`  estimator=LogisticRegression(`
	`  solver='lbfgs',`
	`  max_iter=3000,`
	`  random_state=1`
	`  ),`
	`  param_grid={`
	`  'C': [0.01, 0.1, 1, 10]`
	`  },`
	`  cv=5,`
	`  scoring={`
	`  'accuracy': 'accuracy',`
	`  'precision': make_scorer(precision_score, average='macro', zero_division=0),`
	`  'recall': make_scorer(recall_score, average='macro', zero_division=0),`
	`  'f1': make_scorer(f1_score, average='macro', zero_division=0)`
	`  },`
	`  refit='accuracy',`
	`  verbose=0,`
	`  return_train_score=True`
	`  )`
	`  return model`

Multilayer Perceptron

Similar to Random Forest, we selected a Multilayer Perceptron for its ability to model nonlinear relationships, with the added advantage of being well-suited for complex and diverse datasets.

Hyperparameters	Model

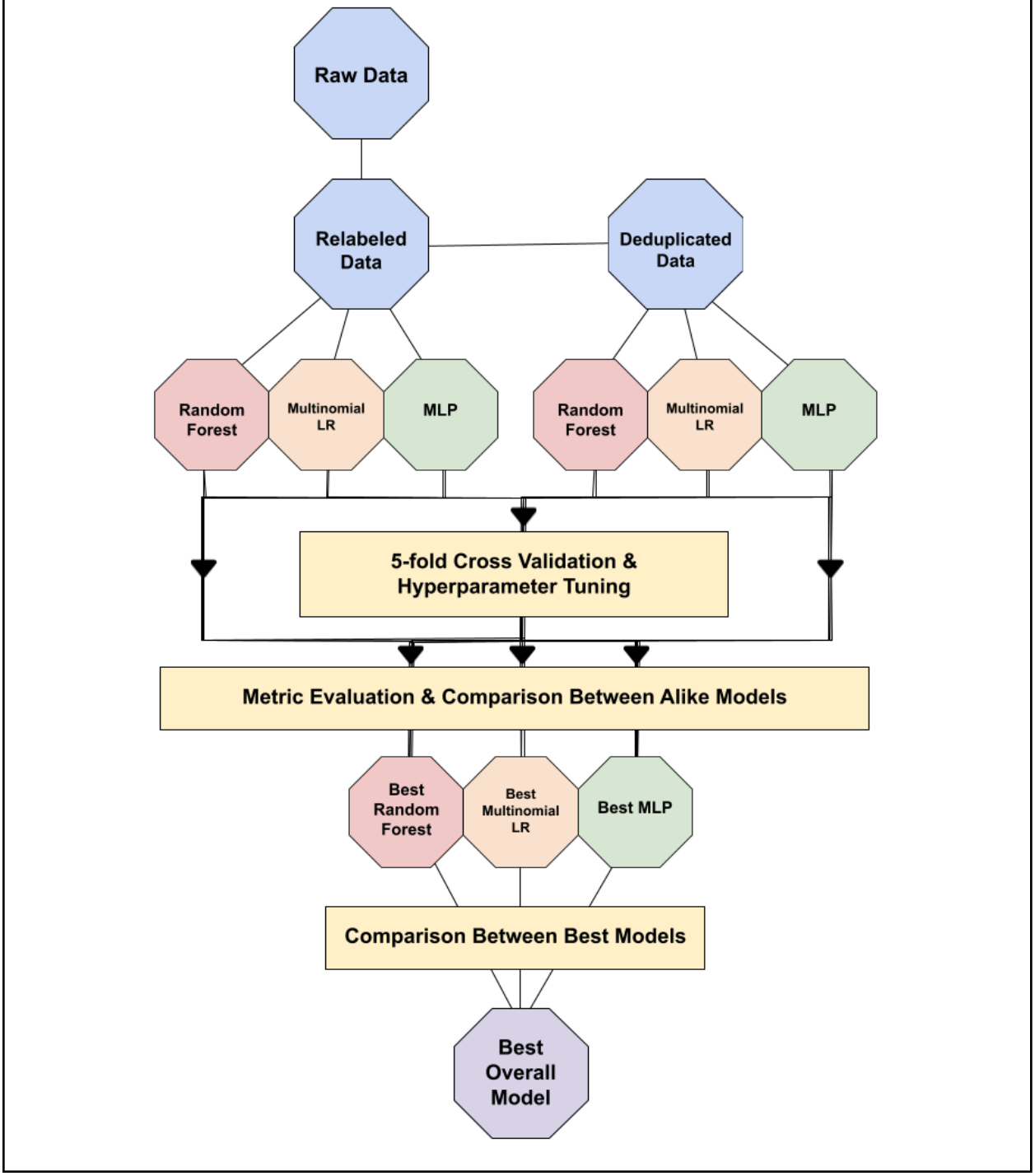
epochs [250,100,45]: *iterations through entire dataset*
 	
learning_rate [0.1, 0.01, 0.001]: *controlling weight updates*
	`def build_mlp(array):`
	`  model = Sequential([`
	`  Input(shape=(array.shape[1],)),`
	`  Dense(512, activation='relu'),`
	`  Dense(512, activation='relu'),`
	`  Dense(26, activation='softmax')`
	`  ])`
	`  model.compile(`
	`  optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),`
	`  loss='sparse_categorical_crossentropy',`
	`  metrics=['accuracy']`
	`  )`
	`  return model`

Experiment Plan

We ran four versions of each model: two using the relabeled dataset, and two using the relabeled and deduplicated dataset, to evaluate whether removing duplicates would

impact performance metrics. For each dataset, one version used 5-fold cross-validation while the other did not, allowing us to explore whether the additional runtime from cross-validation had any noticeable effect on performance. After tuning hyperparameters, we compared the four versions of each model to one another, and then compared the best-performing version of each model to determine the overall best performer.

Project Pipeline



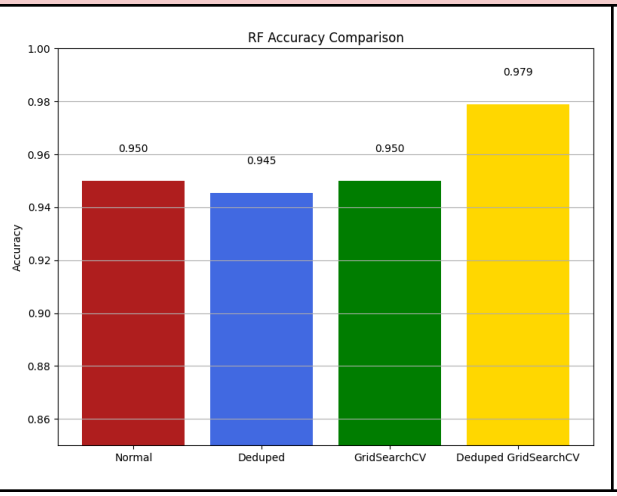
Results

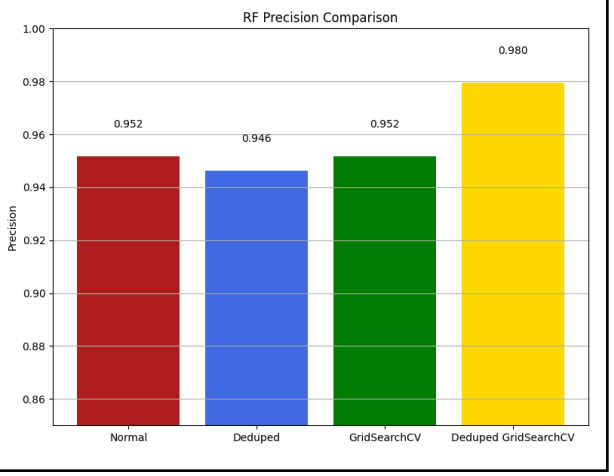
First we will compare the normal (relabeled) and deduplicated models of the same type, and then we will compare the best Random Forest, Multinomial Logistic Regression, and MLP models to each other to determine the best overall model.

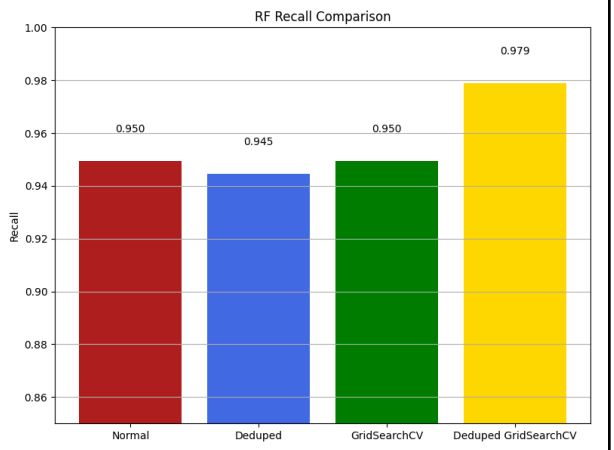
Brief Review of Evaluation Metrics:

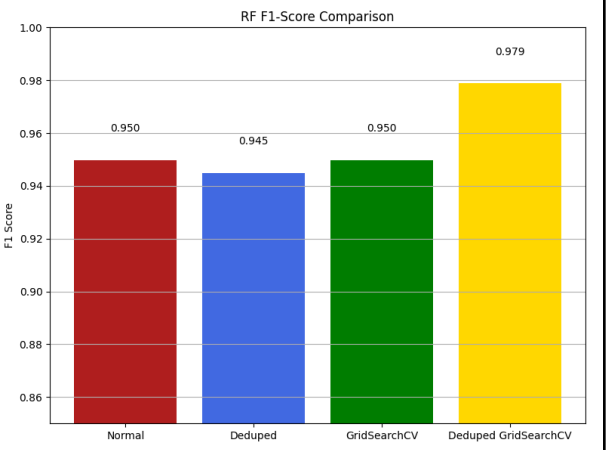
- Accuracy : Percentage of all predictions the model got right
- Precision : How often the model's positive predictions were actually correct.
- Recall : How often a 'correct' prediction was actually correct
- F1-score : Balance between precision and recall, most useful when classes are imbalanced

Random Forest Evaluation Metrics (on test set)









Best Model

Deduped w/ GridSearchCV : [n_estimators=750, max_depth=15]

Why this model?

Our best Random Forest model was the one trained with **GridSearchCV** on the **deduplicated dataset**. This result was somewhat surprising, since Random Forests are generally robust to duplicate data due to their use of multiple randomized decision trees. However, the version trained on the deduplicated data with cross-validation consistently outperformed the others.

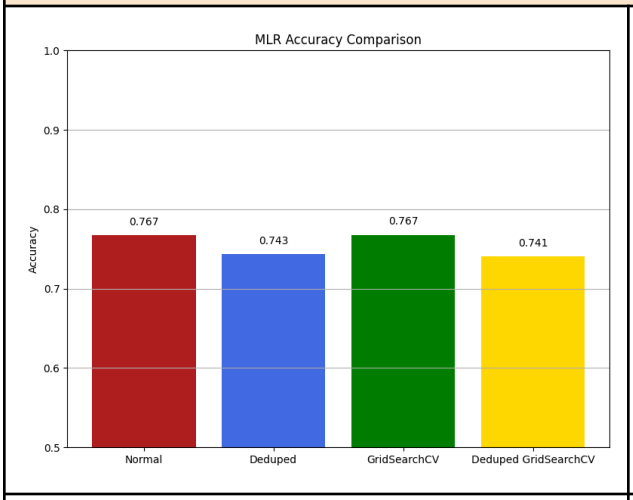
Cross-validation offers much greater reliability because models evaluated without it can achieve deceptively high or low scores depending on how the data is split, whereas using it ensures a more consistent performance across different subsets of the data. The models that used cross validation did not have significantly longer training times, which was another thing we wanted to keep in mind, given that the faster our models train and run, the greater potential they have for daily use.

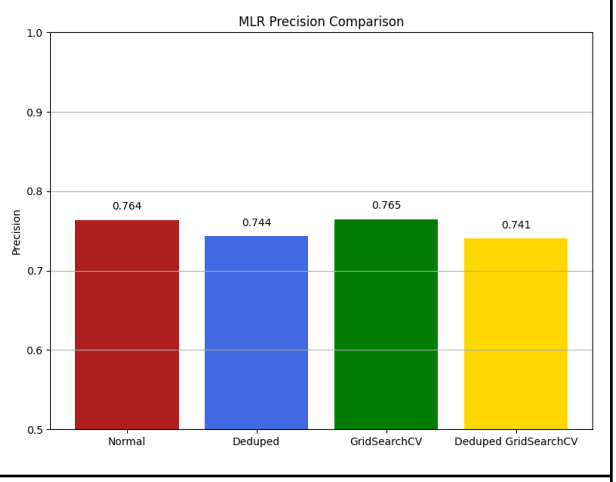
The final model has **750 trees** and a **max depth of 15**. It chose the highest number of trees and the maximum depth in the set of given values, most likely because of the complexity of a task like letter recognition. The more trees, and the more depth per tree, the more complex patterns the model can capture.

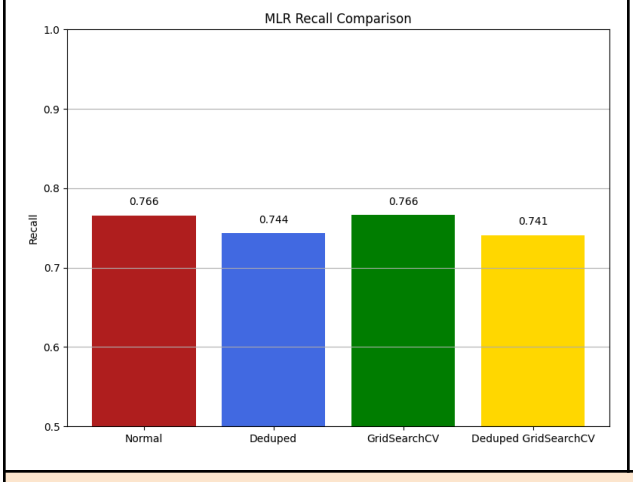
The model achieved around **98%** on accuracy, precision, recall, and F1-score, which indicates very strong overall performance. A high accuracy means the model consistently

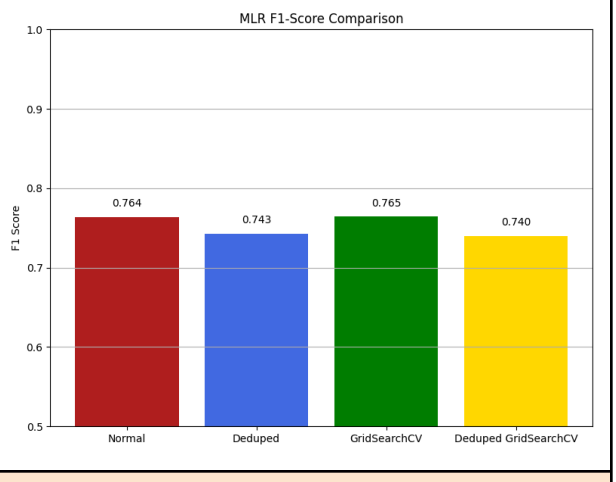
correctly predicted the right letter. A high precision means that when the model predicted a letter, it was usually the right one. A high recall means that model was good at correctly identifying true positives, and a high F1-score shows a strong balance between precision and recall. Having high scores on all these values shows that the model will likely generalize well to unseen letter data.

Multinomial Logistic Regression Evaluation Metrics (on test set)









Best Model

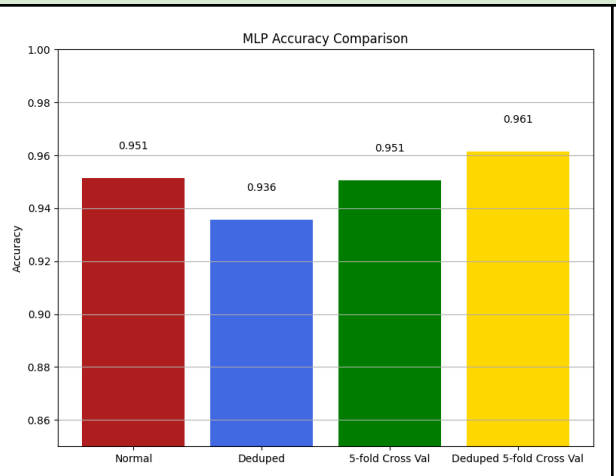
Normal w/ GridSearchCV : [C=10]

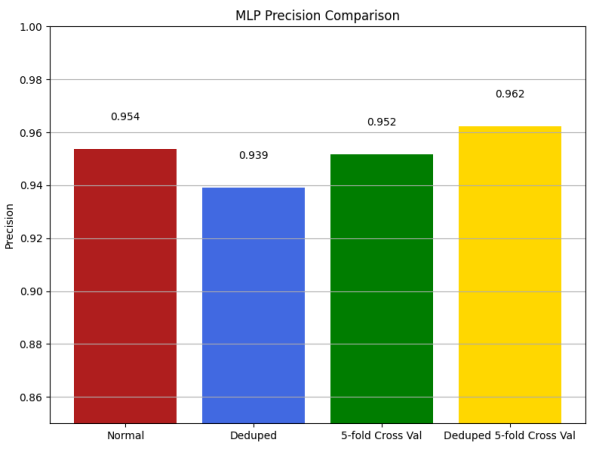
Why this model?

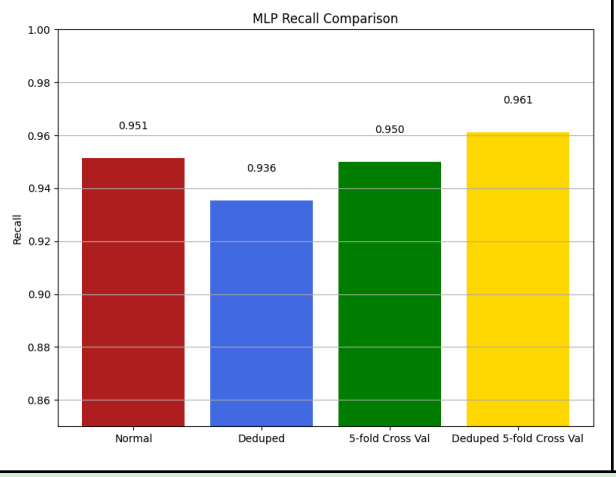
The best Multinomial Logistic Regression model was the one that trained with **GridSearchCV** on the **normal dataset**. This is likely because it's a linear model, and the deduplication process resulted in a more imbalanced dataset. It had much lower overall scores than the Random Forest model, at around **76.5%**.

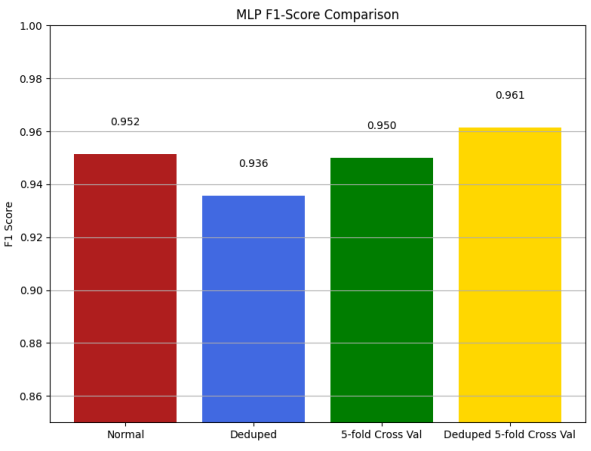
It had a **C value** of **10**, the highest among the tested options. Similar to the Random Forest model, this is likely because of the complexity of the task. In logistic regression, a higher C value means less regularization, allowing the model to fit the training data more closely. This could have been helpful in accurately recognizing similar looking letters.

Multilayer Perceptron Evaluation Metrics (on test set)

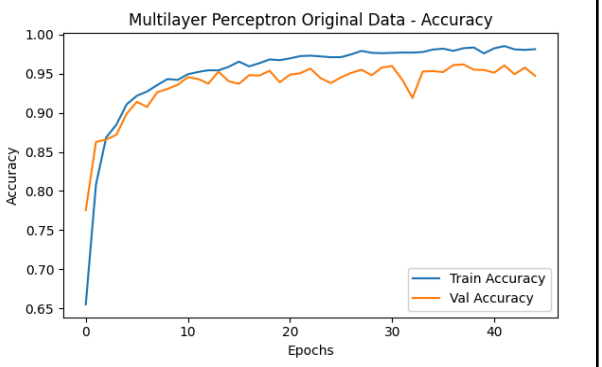


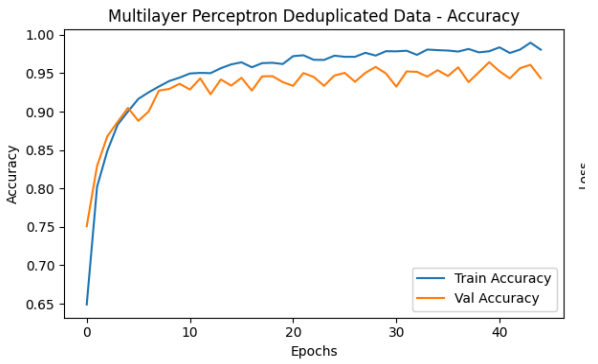


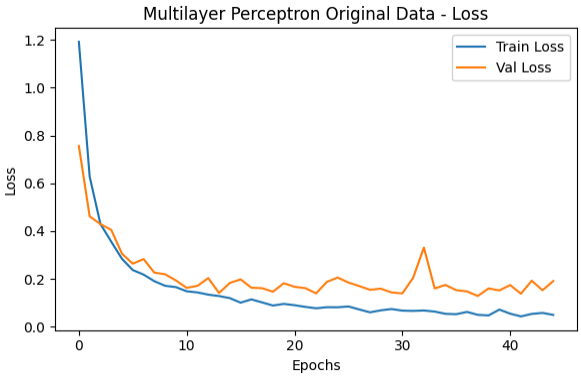


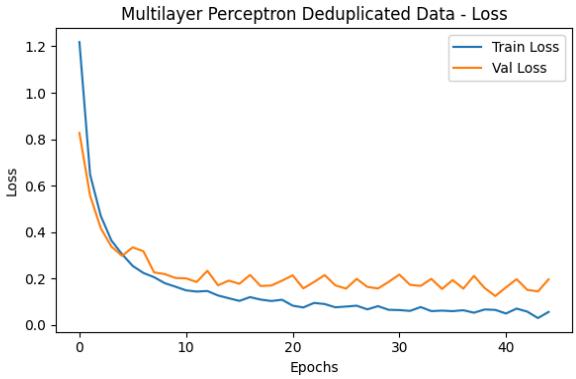


Training Graphs (w/o CV)









Best Model
Deduped w/ 5-fold Cross-Validation : [epochs=45, learning_rate=0.001]

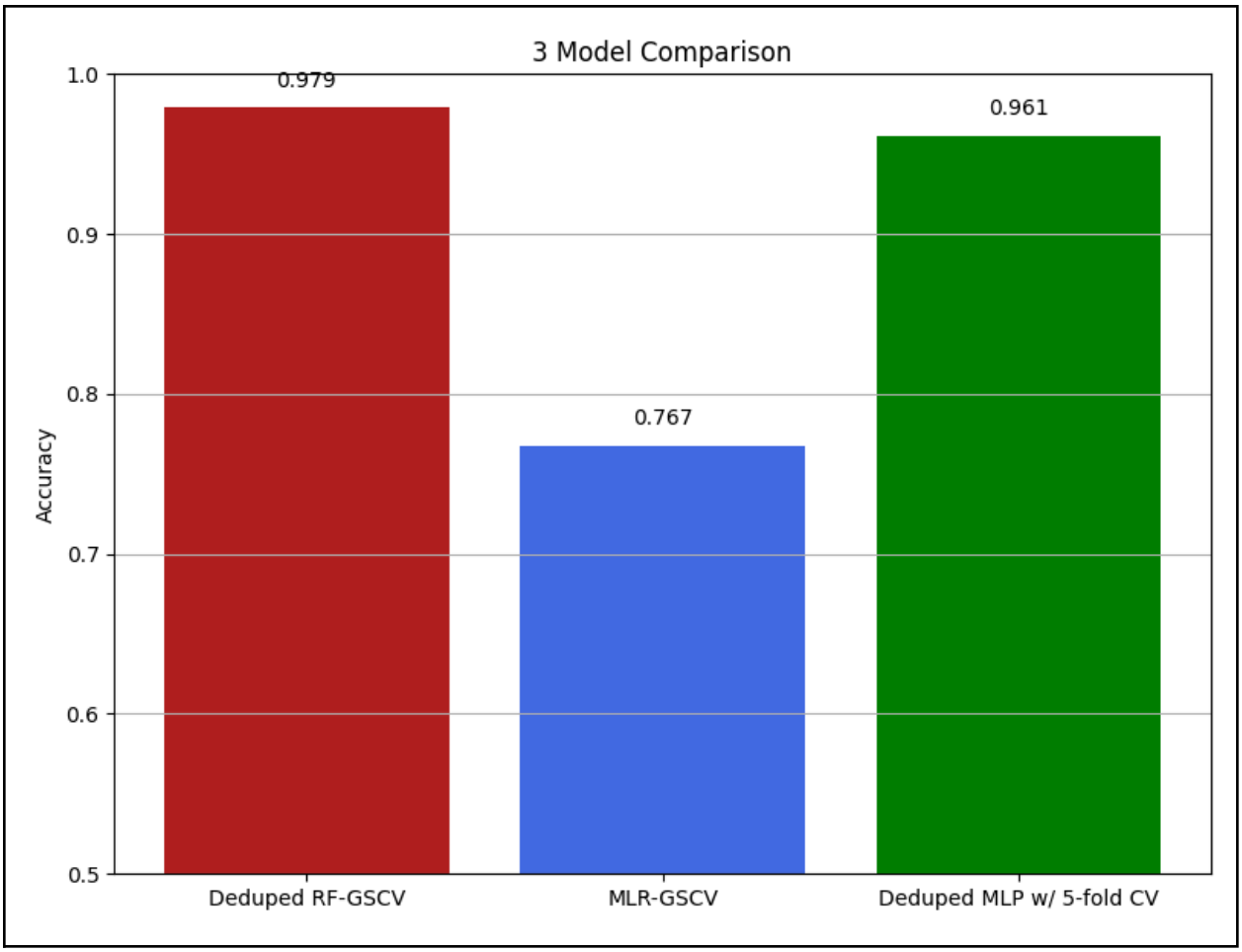
Why this model?

Like the Random Forest model, the MLP performed best with the **deduplicated dataset** and **5-fold cross-validation**. It had accuracy, precision, recall, and f1-scores around **96%**, which is slightly worse than the Random Forest but not by very much. This model took much, much longer to run than the Random Forest and the Multinomial Logistic Regression models, perhaps because of the large number of neurons per layer (512).

The final model used a **learning rate** of **0.001** and trained for **45 epochs**. We initially started with 250 epochs and gradually reduced this to 100 and then 45 based on training stability. When we tried fewer epochs, the spikes in the loss and accuracy graphs became too drastic, which is a sign of underfitting, so we stopped at 45. Similarly, we tested learning rates of 0.1, 0.01, and 0.001. The higher learning rates caused the loss and

accuracy to fluctuate too much. Among the three learning rates tested, 0.001 was the smallest, resulting in smaller and more controlled weight updates across epochs.

Overall Accuracy Comparison Between Models		
Random Forest	Training	Test
Normal	0.9942	0.9500
Deduped	0.9941	0.9454
Normal w/ GridSearchCV	0.9458	0.9500
Deduped w/ GridSearchCV	0.9385	0.9790
Multinomial LR		
Normal	0.7806	0.7670
Deduped	0.7666	0.7434
Normal w/ GridSearchCV	0.7733	0.7675
Deduped w/ GridSearchCV	0.7576	0.7408
Multilayer Perceptron		
Normal	0.9525	0.9515
Deduped	0.9356	0.9357
Normal w/ 5-fold CV	0.9423	0.9505
Deduped w/ 5-fold CV	0.9432	0.9614
Comparison Between the 3 Best Models		



The Best Overall Model

The best overall model was the **deduplicated Random Forest with GridSearchCV**, using 750 estimators and a max depth of 15. It consistently achieved the highest performance across all key metrics, scoring around 98% on accuracy, precision, recall, and F1-score. What really set it apart, however, was its efficiency and stability. Despite its complexity, its training time remained short relative to the MLP, and cross-validation helped ensure reliability across different data folds.

Another key takeaway was how much better all models performed when trained on the deduplicated dataset with cross-validation. While Random Forest is generally robust, the presence of duplicate rows could have caused skewed splits in non-CV runs. By removing those duplicates and using cross-validation, the data was less biased and the evaluation more fair.

Areas for improvement

For the MLP model, we didn't implement automated hyperparameter tuning for the number of hidden layers or neurons per layer. Incorporating something like a grid search could've helped us discover even more optimal architectures.

In addition, all our best-performing models maxed out the tested hyperparameter ranges (e.g., 750 trees, max depth of 15, C = 10), which suggests we could see better results by expanding the grid search range even further.

Conclusion

This project set out to predict handwritten capital letters using pixel data and evaluate multiple machine learning models for accuracy and efficiency. After preprocessing the dataset using Linux command-line tools and evaluating three model types (Random Forest, Multinomial Logistic Regression, and Multilayer Perceptron), we found that Random Forest trained on the deduplicated dataset with GridSearchCV gave the strongest results, achieving nearly 98% across all key evaluation metrics.

By working on this project, we demonstrated how traditional OCR problems can still benefit from modern tooling and thoughtful pipeline design. Our work proves that even with the same dataset, model performance can vary a lot depending how the data is prepared and how the models are validated. The consistent success of cross-validation also reinforces how important it is when working with real-world data, where a lucky (or unlucky) data split can otherwise skew the outcome.

In the future, we hope to explore automated tuning of neural network structures, run additional tests with larger models (more trees, deeper networks), and potentially explore other model types all together. With OCR being an essential part of accessibility, translation, and document scanning, improving letter-recognition has real-world impact that extends well beyond this project.

References

Frey, P.W., Slate, D.J. Letter recognition using Holland-style adaptive classifiers. *Mach Learn* 6, 161–182 (1991). https://doi.org/10.1007/BF00114162