M1 Machine Learning Coursework Report

Steven Ma(ym432)

December 2024

Word Count: 2958

1 Introduction

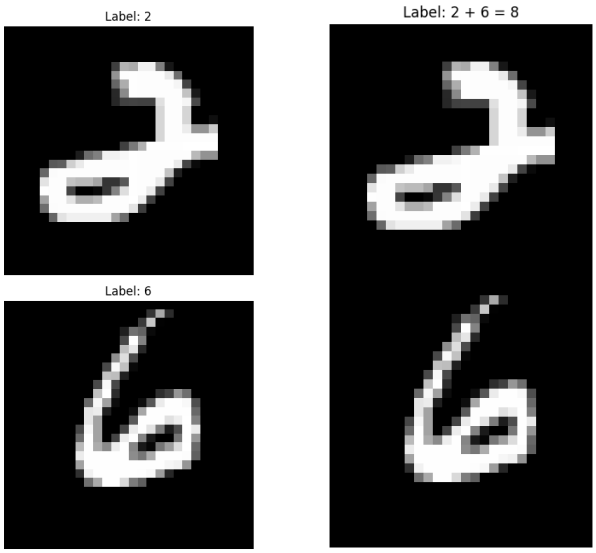
The MNIST dataset, a widely used benchmark in computer vision, consists of grayscale images of handwritten digits (0–9) and serves as a standard for evaluating machine learning algorithms. This project extends the MNIST dataset by constructing a new dataset where two digits are combined vertically, with their sum serving as the label. The study implements neural networks alongside classical machine learning algorithms, including Random Forest (RF) and Support Vector Machines (SVM), for classification tasks. It also explores the performance of weak classifiers under varying sample sizes and finishes by visualizing the t-distributed Stochastic Neighbor Embedding (t-SNE) embeddings.

2 Data Format and Preprocessing

2.1 Input format

The original MNIST dataset consists of 28×28 grayscale images of handwritten digits, with pixel values normalized between 0 and 1, representing 10 classes ranging from 0 to 9. For this task, the input data was created by vertically concatenating two MNIST samples to form new images of size 56×28 (see Figure 1), with the label being the sum of the two original labels. The combined dataset contains 19 classes ranging from 0 to 18, corresponding to the possible sums of two MNIST digits.

The combined image shown in Figure 1(b) is further flattened into a vector of size $56 \times 28 = 1568$ dimensions to serve as the input to the model. This is a classification task, where the label y is transformed into a one-hot encoded vector of dimension 19. For instance, if the original label y is 9, the resulting 19-dimensional vector will have its 9-th component set to 1, while all other components are set to 0.



(a) (b)

Figure 1: (a) shows two individual MNIST samples, each of shape 28×28. (b) shows their vertical concatenation to form a combined sample with a new label, shape is 56×28.

2.2 Dataset Generation

The dataset was initially created by repeatedly selecting two random images from the MNIST dataset and concatenating them vertically. A dataset generated in this manner has an imbalanced class distribution, with the label 9 having the highest probability of being generated, while labels 0 and 18 have the lowest probabilities. The imbalance is illustrated by left side of Figure 2.

This imbalance raised concerns that overrepresented labels, such as 9, might dominate the training process, while underrepresented labels, such as 0 and 18, might result in biased model performance (in Section 3.4 we will discuss why this was not the case).

To address this potential bias, a balanced dataset was created by generating a large number of samples and then selecting an equal number of samples for each class. The class distribution of the balanced dataset is illustrated on the right side of Figure 2. This ensured that each label in the balanced dataset had the same representation, mitigating the assumed imbalance in the training process.

To ensure a fair comparison of future model performance between the balanced and imbalanced datasets, the final datasets were constructed to have the same number of samples for training, validation, and testing, as follows:

- **Training set:** $3500 \times 19 = 66,500$ samples
- **Validation set:** $1000 \times 19 = 19,000$ samples
- **Test set:** $1000 \times 19 = 19,000$ samples

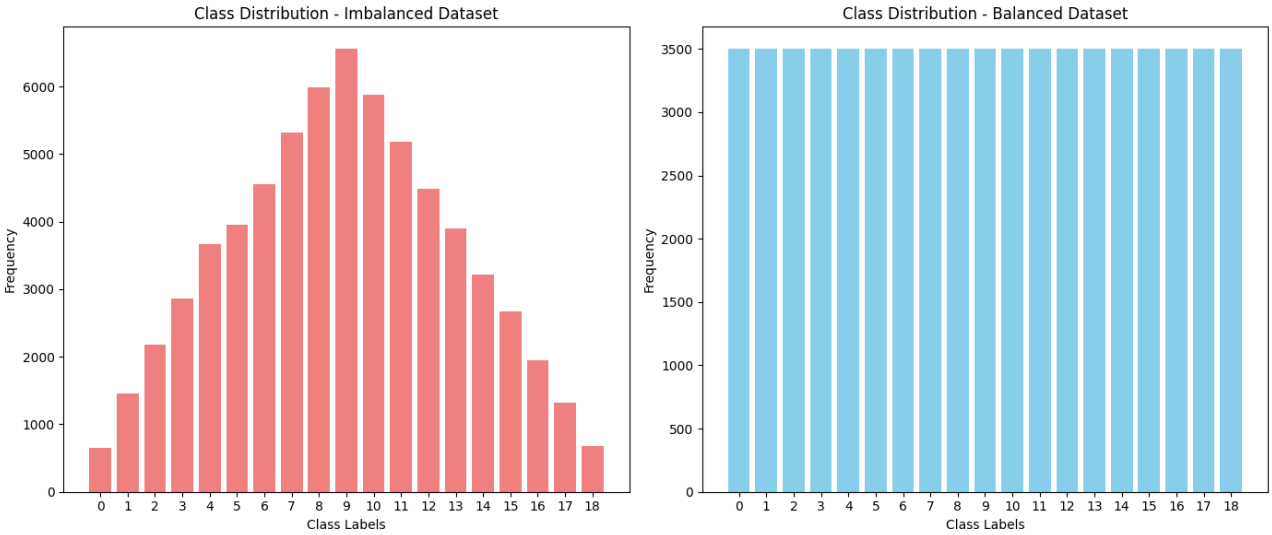


Figure 2: Class distributions of the training set (66,500 samples each). The left plot illustrates the imbalanced dataset, while the right plot shows the balanced dataset.

3 Fully Connected Neural Networks

3.1 Model Architecture and Training

This section focuses on utilizing a Fully Connected Neural Network (FCN) to predict the sum of the concatenated MNIST samples. The FCN consists of three main components:

- **Input Layer:** The input layer takes the concatenated and flattend MNIST sample as input, which has a dimension of 1568.
- **Hidden Layers:** The model includes multiple hidden layers with configurable hyperparameters such as the number of units, activation functions (`relu`, `sigmoid`, or `tanh`), and dropout rates (0.1, 0.15, or 0.2).
- **Output Layer:** The output layer has a dimension of 19, corresponding to the 19 possible classes (0 to 18). A `softmax` activation function is applied to convert output of previous layer into a probability distribution. The i-th component of the output vector represents the probability that the input sample belongs to class i.

The architecture of the FCN is illustrated in Figure 3 and the model uses the Adam optimizer, known for its efficiency and adaptability.

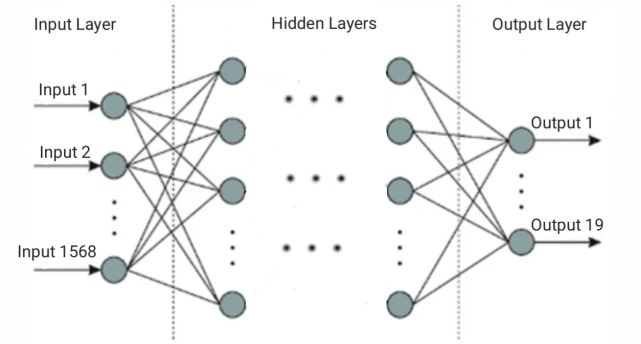


Figure 3: Architecture of the FCN.

3.1.1 Output Layer

In multiclass logistic regression, the goal is to predict the probability $\hat{y}_{ij}$ that each sample $\vec{x}_i$ belongs to class j. Here, $\hat{y}_{ij}$ represents the predicted probability of sample $\vec{x}_i$ being in class j, where $\vec{x}_i$ is the output from the previous layer of the network.

For each class j, a linear model is used to compute a score z_{ij} for that class:

$$z_{ij} = \vec{x}_i^\top \vec{w}_j + b_j, \tag{1}$$

where:

- $\vec{w}_j$ is the weight vector for class j.
- b_j is the bias term for class j.

Softmax Function:
To transform these raw scores z_{ij} into probabilities, we apply the softmax function.
We apply the softmax function to transform z_{ij} into probabilities in the range [0,1]. The softmax function is defined as:

$$\hat{y}_{ij} = \frac{e^{z_{ij}}}{\sum_{k=1}^{C} e^{z_{ik}}}, \tag{2}$$

Categorical Cross-Entropy Loss:
The model is trained using the categorical cross-entropy loss function, which is given by:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij}), \tag{3}$$

where:

- L is the total loss for the dataset.
- N is the total number of samples in the dataset.
- C is the number of classes (19 in this task).
- y_{ij} is the true label for sample i and class j, represented as a one-hot encoded vector ($y_{ij} = 1$ if sample i belongs to class j, otherwise $y_{ij} = 0$).

The predicted class is determined by selecting the index of the highest probability in the output vector $\vec{y}_i$.

3.1.2 Hyperparameter Tuning

The models were constructed using `tensorflow.keras`, and 20 sets of hyperparameters were selected through `keras_tuner.RandomSearch` from the search space defined in Table 1. The search process randomly sampled hyperparameter combinations and optimized the categorical cross-entropy loss on the validation set. These 20 hyperparameter configurations were then used to train models separately on both the balanced and imbalanced datasets, enabling a systematic evaluation of model performance under different data distributions.

Hyperparameter	**Choices**
Number of Hidden Layers	2, 3, 4
Units per Hidden Layer	32, 64, 128
Activation Function	`relu`, `sigmoid`, `tanh`
Dropout Rate	0.1, 0.15, 0.2
Learning Rate	10^{-3}, 10^{-4}, 10^{-5}

Table 1: Hyperparameters search space

The number of hidden layers, units per layer, activation functions, dropout rates, and learning rates directly impact model complexity, non-linear representation ability, regularization effectiveness, and optimization efficiency, making them critical factors for performance. Therefore, these were selected as the search space for hyperparameter optimization.

The training process was set to a default of 50 epochs. However, an early stopping mechanism was employed to prevent overfitting and minimize unnecessary overtraining. Specifically, the `EarlyStopping` callback monitored the validation loss and stopped training if no improvement was observed for 5 consecutive epochs. Additionally, the model's weights were reverted to those corresponding to the epoch with the lowest validation loss.

A total of 40 models were trained using the CPU version of TensorFlow 2.10 on an Intel Core i7-13700H processor. The entire process took approximately two and a half hours, with each model requiring an average training time of 3 to 4 minutes.

3.2 Model Performance

Hereafter, models trained on the balanced dataset are referred to as "balanced models," and those trained on the imbalanced dataset are referred to as "imbalanced models."

Among the 40 models, 14 (7 pairs of balanced and imbalanced models) achieved test accuracy above 85%, referred to as "good models." Each pair was trained using the same set of hyperparameters, highlighting 7 distinct hyperparameter configurations that consistently resulted in strong performance across both datasets. Most imbalanced models showed slightly lower performance on the imbalanced test set compared to the performance of balanced models on the balanced test set.

Model and Test	**Mean Accuracy**	**Best Accuracy**
Balanced Model on Balanced Test	0.6809	0.8979
Balanced Model on Imbalanced Test	0.6130	0.8835
Imbalanced Model on Balanced Test	0.6521	0.8912
Imbalanced Model on Imbalanced Test	0.6419	0.8897

Table 2: Mean and best accuracies of balanced and imbalanced models on balanced and imbalanced test sets.

Best Model Performance and Architecture:

The best models achieved test accuracy between 89% and 90% on their respective test sets. Notably, the best balanced model and the best imbalanced model used the same set of hyperparameters. The architecture for these models is as follows:

- **Input Layer:** Input shape: 1568.
- **Hidden Layer 1:** 128 units, `sigmoid` activation, dropout rate: 0.2.
- **Hidden Layer 2:** 64 units, `relu` activation, dropout rate: 0.1.
- **Output Layer:** 19 units, `softmax` activation.

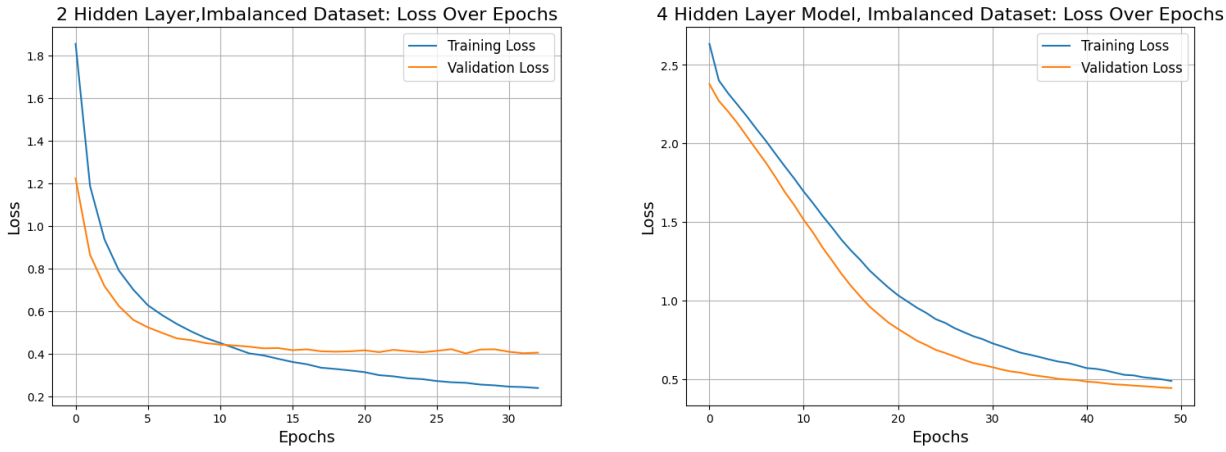
3.2.1 Good Models

Key Characteristics:

- **Learning Rate:** Moderate learning rates (e.g., 10^{-3}) enabled faster parameter updates and better convergence.
- **Network Structure:** Simpler architectures with two hidden layers performed well, particularly when the first layer had a large number of units and the second layer appropriately reduced this number.
- **Activation Functions and Dropout:** Classic activation functions such as `relu` or `sigmoid`, paired with moderate dropout rates (0.1–0.2), facilitated effective learning.

Training Dynamics:

- **Shallow Models:** Models with two hidden layers showed rapid improvement in validation accuracy, often reaching 80% within the first 10 epochs. However, due to limited complexity, their validation loss stopped decreasing, triggering early stopping between epochs 15 and 40(see Figure 4a). These models completed training in under one and half minutes .
- **Deeper Models:** Models with three to four hidden layers improved validation accuracy more gradually and continued to reduce validation loss without triggering early stopping. These models typically required around three and a half minutes to complete training. The validation loss still exhibited a downward trend at this point, indicating that the model had not fully converged, as shown in Figure 4b.



(a) (b)

Figure 4: Loss curves for typical good shallow and deep, imbalanced models.

3.3 Balanced and Imbalanced Models

Refer back to the assumptions made in Section 2.1, where it was hypothesized that models trained on the balanced dataset would perform better and more robustly across both test sets. This was proven not to be true.

Referring to Table 2, the following observations were made:

The performance of balanced and imbalanced models across different test sets is summarized below:

Balanced Model Performance:

- Mean accuracy on the balanced test set: 0.6809.
- Mean accuracy on the imbalanced test set: 0.6130 (6.8% drop).
- Best balanced model achieved:
  - 0.8979 on the balanced test set.
  - 0.8835 on the imbalanced test set (1.4% drop).

Imbalanced Model Performance:

- Mean accuracy on the balanced test set: 0.6521.
- Mean accuracy on the imbalanced test set: 0.6419.
- Best imbalanced model achieved:
  - 0.8912 on the balanced test set.
  - 0.8897 on the imbalanced test set (0.1% difference).

The balanced model showed a consistent performance drop on the imbalanced test set, with all 20 models experiencing an average decline of 6.8%. In contrast, the imbalanced model demonstrated widespread performance improvement on the balanced test set, with 16 out of 20 models achieving an average gain of 1% (see Figure 5). This highlights the robustness of the imbalanced model across varying test set distributions.

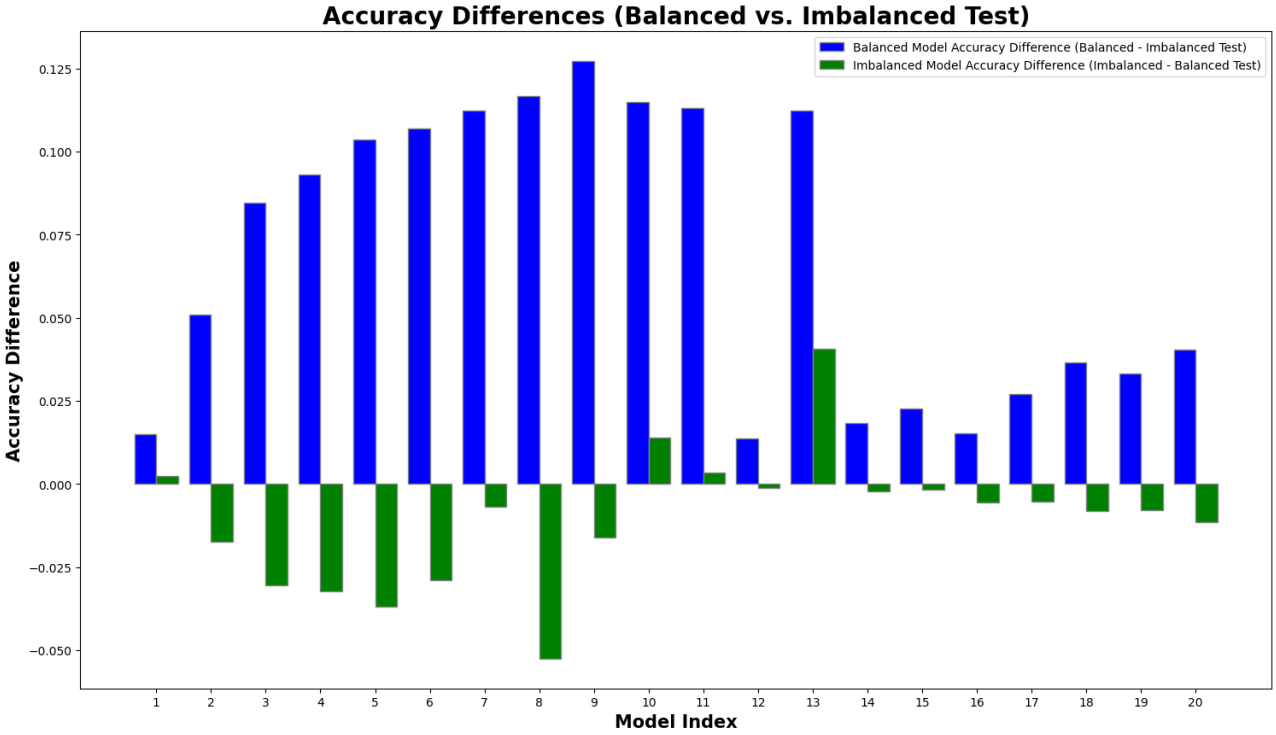


Figure 5: Accuracy differences between balanced and imbalanced test sets for all 20 models.

All subsequent models will use the setup where imbalanced data is used for training to expose the model to the natural label distribution, while balanced data is used for testing to ensure unbiased evaluation.

3.4 Impact of Data Distribution and Class Composition on Model Performance

Figure 6 visualizes the classification report for the balanced model on the balanced test set. Using F1-score as the reference metric, a clear trend can be observed: the F1-score decreases from class 0 to class 9, reaching its lowest point, and then gradually increases, achieving another peak at class 18. This is a general trend across balanced models tested on the balanced test set.

This phenomenon indicates that labels closer to class 9 are inherently more challenging for the model to learn, whereas labels near 0 and 18 are comparatively easier to classify. Referring to Table 3, it is evident that labels near class 9 have the largest number of unique combinations. In the balanced dataset, enforcing equal sample counts across all classes reduces the diversity of constituent combinations for each label. For instance, while label 0 (formed only by the combination (0,0)) includes 3500 samples, label 9 (with combinations like (0,9), (1,8), (2,7), etc.) also has only 3500 samples, despite being constructed from 10 unique combinations.

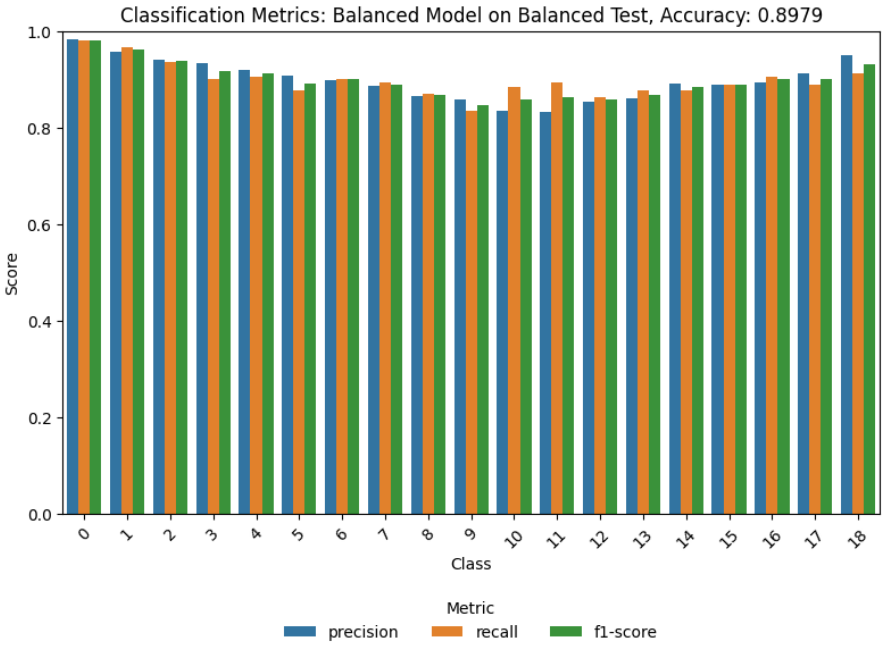


Figure 6: Classification report for the balanced model on the balanced test set.

In contrast, the imbalanced dataset naturally preserves the full range of diversity in class compositions. Since it includes 19 classes formed from 100 unique combinations, each combination appears in proportion to its natural occurrence, allowing the model to encounter a more representative data distribution. This broader exposure may explain why imbalanced models are more robust across varying test set distributions.

However, this explanation does not fully account for why imbalanced models perform significantly worse on classes 0 and 18 on the balanced test set compared to the imbalanced test set (see Figure 7). One possibility is that the dominance of class 9 in the imbalanced training set leads the model to prioritize learning the class label distribution over the input-output mappings. Alternatively, the limited number of samples for extreme classes (e.g., 0 and 18) in the imbalanced dataset could also contribute to this behavior. Further investigation into these discrepancies is warranted but falls outside the scope of this work.

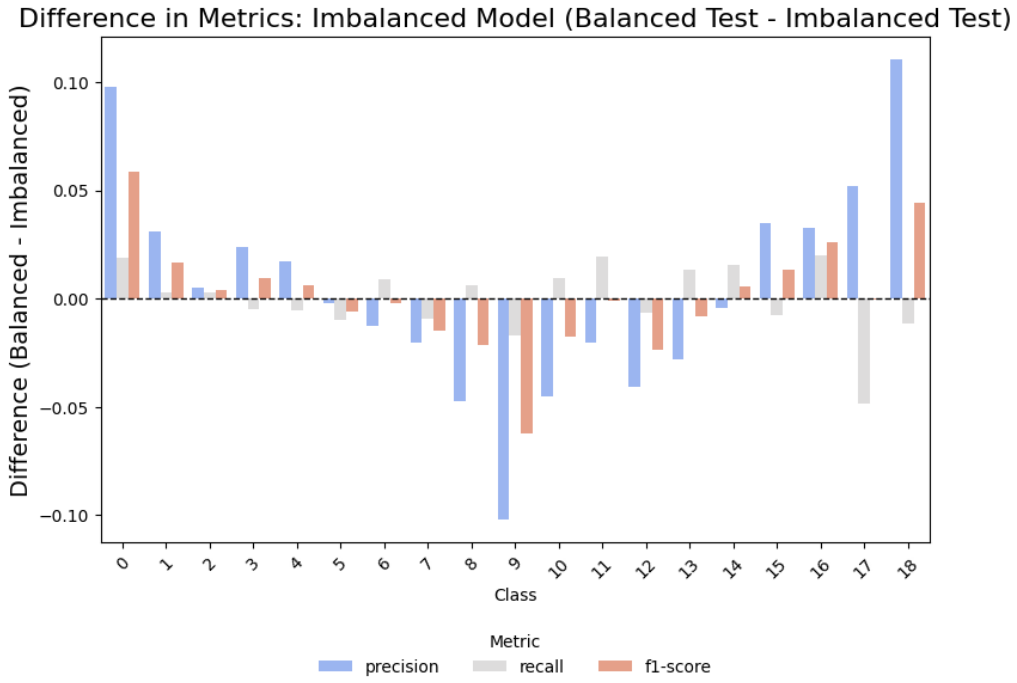


Figure 7: Accuracy differences for imbalanced models on balanced versus imbalanced test sets.

New Label	Original Label Pairs	Number of Combinations
0	(0,0)	1
1	(0,1), (1,0)	2
2	(0,2), (1,1), (2,0)	3
3	(0,3), (1,2), (2,1), (3,0)	4
4	(0,4), (1,3), (2,2), (3,1), (4,0)	5
5	(0,5), (1,4), (2,3), (3,2), (4,1), (5,0)	6
6	(0,6), (1,5), (2,4), (3,3), (4,2), (5,1), (6,0)	7
7	(0,7), (1,6), (2,5), (3,4), (4,3), (5,2), (6,1), (7,0)	8
8	(0,8), (1,7), (2,6), (3,5), (4,4), (5,3), (6,2), (7,1), (8,0)	9
9	(0,9), (1,8), (2,7), (3,6), (4,5), (5,4), (6,3), (7,2), (8,1), (9,0)	10
10	(1,9), (2,8), (3,7), (4,6), (5,5), (6,4), (7,3), (8,2), (9,1)	9
11	(2,9), (3,8), (4,7), (5,6), (6,5), (7,4), (8,3), (9,2)	8
12	(3,9), (4,8), (5,7), (6,6), (7,5), (8,4), (9,3)	7
13	(4,9), (5,8), (6,7), (7,6), (8,5), (9,4)	6
14	(5,9), (6,8), (7,7), (8,6), (9,5)	5
15	(6,9), (7,8), (8,7), (9,6)	4
16	(7,9), (8,8), (9,7)	3
17	(8,9), (9,8)	2
18	(9,9)	1

Table 3: Mapping of New Labels to Original Label Pairs and Number of Combinations to Generate Each Label.

4 Other Algorithms

4.1 Random Forest Classifier

Random Forest (RF) is an ensemble learning method that improves classification performance by constructing multiple decision trees and averaging their predictions. The model was implemented using `scikit-learn`'s `RandomForestClassifier`.

Before training, all input data were standardized using `StandardScaler`, which scales each feature to have a mean of 0 and a standard deviation of 1. This step eliminates the influence of feature magnitudes on the model.

Among the hyperparameters of RF, the number of trees is the most critical and was chosen for hyperparameter tuning. Five values were selected: [50, 100, 200, 300, 500], while other hyperparameters were fixed to:

Hyperparameter	Value
`max_depth`	`None`
`min_samples_split`	2
`min_samples_leaf`	1
`max_features`	`sqrt`
`class_weight`	`balanced`

Table 4: Hyperparameters for RF.

As the number of trees increased, the model's validation accuracy also improved, as shown in Table 5. However, the incremental gains in accuracy diminished as the number of trees increased, indicating the model was progressively converging toward its optimal performance.

Number of Trees	Validation Accuracy
50	0.7115
100	0.7465
200	0.7661
300	0.7734
500	0.7802

Table 5: Accuracy for RF on balanced test set.

Figure 8 shows the RF model's classification report on the validation set, with an F1-score trend similar to Figure 6, where class 9 has the lowest F1-score, and F1-scores decrease as classes approach class 9.

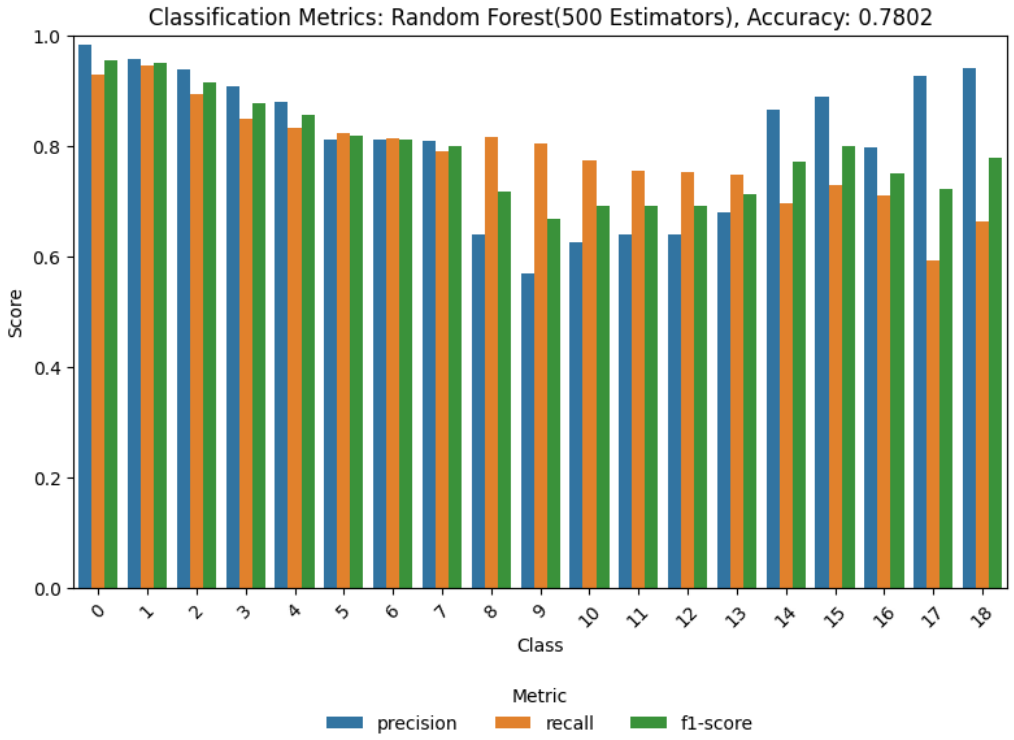


Figure 8: Classification report for RF on balanced test set.

Random Forest supports multithreaded computation. In this experiment, 6 threads were utilized, and training the model with 500 trees required only about 80 seconds, demonstrating the computational efficiency of Random Forest.

4.2 Support Vector Classifier

SVC (Support Vector Classifier) extends SVM to multi-class problems using strategies such as One-vs-One (OvO). OvO trains classifiers for each pair of classes, effectively simplifying classification by focusing on two classes at a time. This strategy is more suitable for the highly imbalanced 19-class label distribution in this task, as it reduces the class imbalance issues inherent in One-vs-Rest (OvR), where one class is compared against all others [1].

The model was implemented using `scikit-learn`'s `SVC`. In the OvO approach, a classifier is trained for each pair of classes, requiring $\frac{n(n-1)}{2}$ classifiers for an n-class problem[1]. For this 19-class task, the OvO strategy necessitates training $\frac{19\times18}{2} = 171$ classifiers, results in long computational time.

Hyperparameter tuning focused on the kernel function, since it determines how the input is transformed into a higher-dimensional space to be separated by a linear hyperplane. Other hyperparameters considered were:

Hyperparameter	Default Value	Applicable Kernels
`C`	1.0	All Kernels
γ	`scale`	Radial Basis Function(RBF), Polynomial, Sigmoid
`class_weight`	`balanced`	All Kernels

Table 6: Other hyperparameters for SVC.

Due to the long training time, only 20% of the dataset (13,300 samples) was used for hyperparameter tuning. While this is not ideal, long computational time necessitated this approach. The train accuracies for different kernels are as follows:

Kernel	Validation Accuracy
Linear	0.3119
Polynomial (deg=3)	0.3755
RBF	0.5301
Sigmoid	0.3413

Table 7: train accuracies for different kernels.

The RBF kernel, which performed best during hyperparameter tuning, was selected and trained on the full dataset (66,500 samples), it achieved a suboptimal accuracy of 0.7113 on the balanced test set.

The classification report (Figure 9) revealed a similar trend observed in earlier models: class 10 had the lowest F1-score, and F1-scores decreased as the classes approached class 9 and 10.

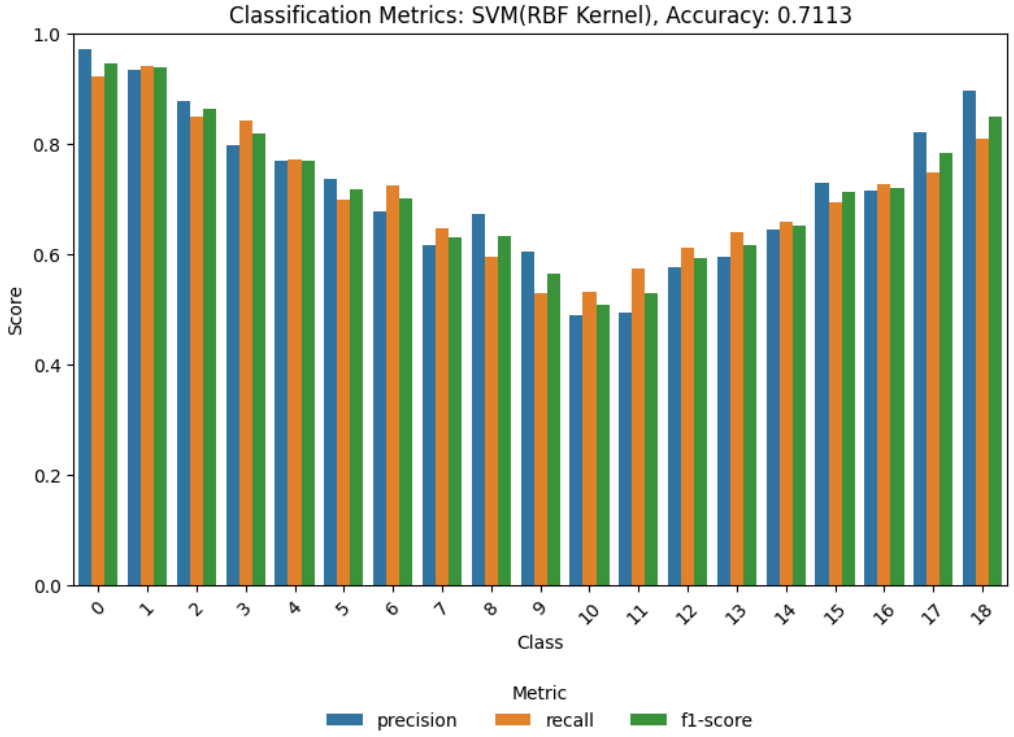


Figure 9: Classification report for the SVM model on the balanced test set.

Limitations of SVC:

- **Computational Time:** SVC does not support multithreading, requiring 171 classifiers (for the 19-class OvO strategy) to be trained sequentially. With a time complexity more than $O(n^2)$ (n is the number of data points)[2], this resulted in a total training time of 66 minutes.
- **Suboptimal Performance:** Despite the long training time, the SVC model achieved only 0.7113 accuracy on the balanced test set, indicating limited effectiveness for this task.

5 Weak Linear Classifier: Logistic Regression

The logistic regression models used here are based on the model described in Section 3.1.1, with the only difference being that in Section 3.1.1, $\vec{x}_i$ represents the output of the previous layer, while here $\vec{x}_i$ directly corresponds to the input data.

Two methods were implemented:

- **Method 1:** The input consists of two concatenated MNIST images (dimension 1568), and the logistic regression model performs 19-class classification.
- **Method 2:** A 10-class classifier was trained on the upper and lower halves of the images, with the final label obtained by summing the results.

Both methods were evaluated with sample sizes {50, 100, 500, 1000, 2000, 5000, 10000, 20000}. Method 1 required approximately 70 seconds, while Method 2 completed in just 16 seconds.

The accuracy results for both methods are summarized in Table 8 and visualized in Figure 10. Method 2 consistently outperformed Method 1 across all sample sizes, achieving significantly higher accuracy even with smaller training sets.

Sample Size	Method 1 Accuracy	Method 2 Accuracy
50	0.0934	0.4712
100	0.1087	0.6013
500	0.1430	0.7431
1000	0.1483	0.7672
2000	0.1602	0.7901
5000	0.1661	0.7894
10000	0.2032	0.7923
20000	0.2373	0.8118

Table 8: Comparison of Accuracy for Method 1 and Method 2 with Different Sample Sizes.

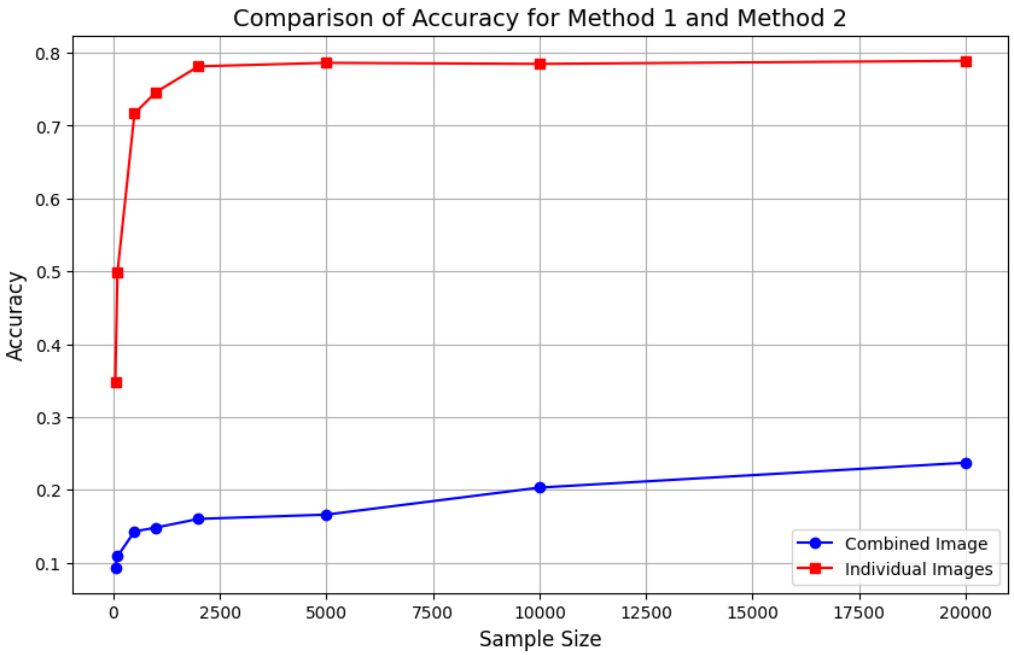


Figure 10: Comparison of Accuracy for Method 1 and Method 2.

6 t-SNE

t-SNE is a technique for visualizing high-dimensional data in lower dimensions. It preserves local data structure by ensuring that similarities between points in high-dimensional space correspond to similarities in low-

dimensional space[3].

6.1 Theoretical Foundations of t-SNE

For two points $\vec{x}_i$ and $\vec{x}_j$, t-SNE defines the conditional probability $p_{j|i}$ as:

$$p_{j|i} = \frac{\exp\left(-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|\vec{x}_i - \vec{x}_k\|^2}{2\sigma_i^2}\right)}. \tag{4}$$

Here, σ_i is a parameter for the Gaussian kernel. The symmetrized joint probability p_{ij} is given by:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}. \tag{5}$$

In the low-dimensional space, the similarity between $\vec{y}_i$ and $\vec{y}_j$ is modeled using a Student-t distribution:

$$q_{ij} = \frac{\left(1 + \|\vec{y}_i - \vec{y}_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|\vec{y}_k - \vec{y}_l\|^2\right)^{-1}}. \tag{6}$$

The goal of t-SNE is to minimize the Kullback-Leibler (KL) divergence between high- and low-dimensional similarities:

$$C = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{7}$$

Perplexity controls the balance between local and global structure, defined as:

$$Perplexity = 2^{H(P_i)}, \tag{8}$$

where $H(P_i)$ is the entropy of $p_{j|i}$:

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}. \tag{9}$$

The optimization process of t-SNE involves two main steps:

- Optimizing σ_i via binary search to match the target perplexity for each point's P_i.
- Refining the low-dimensional coordinates $\vec{y}_i$ using gradient descent to minimize KL divergence.

6.2 Application of t-SNE

Perplexities of $\{5, 30, 50, 100\}$ were tested to find the best configuration for t-SNE.

t-SNE on Embedding Layer Outputs: The input in this case is a 64-dimensional representation obtained from the penultimate layer of the best imbalanced FCN model. Model's mbedding outputs were extracted for 5000 randomly sampled inputs. The optimal perplexity was 50. As shown in Figure 11(a), the visualization reveals well-separated clusters for different labels, indicating that the model effectively captures class-specific features.

t-SNE on Input Data: Similarly, 5000 random samples were selected from the raw input data, and t-SNE was applied across the same perplexity range, with best preplexity being 100. The results, shown in Figure 11(b), reveal overlapping clusters, highlighting the difficulty of distinguishing classes based solely on the raw input data without feature extraction.

Perplexity	KL Divergence (Embedding)	KL Divergence (Input)
5	1.0992	2.3429
30	0.6830	2.5294
50	0.6630	2.4204
100	0.7178	2.2548

Table 9: KL Divergence for t-SNE with different perplexities on embedding and input data.

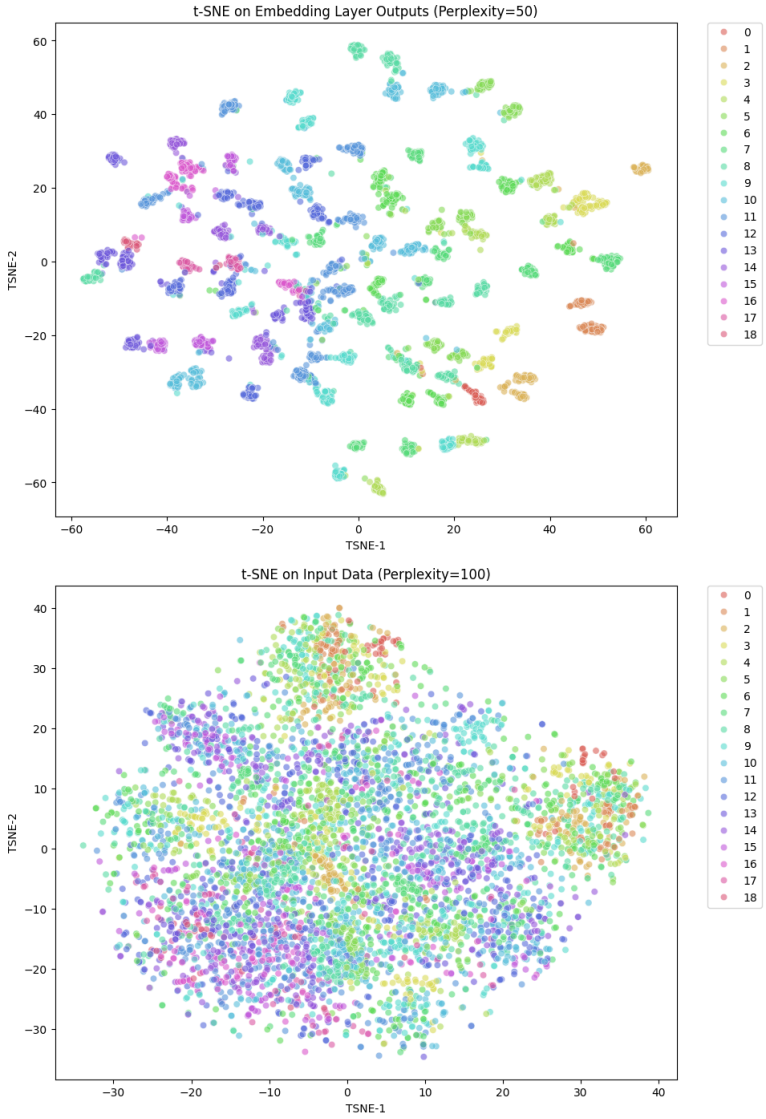


Figure 11: (a) t-SNE on Embedding Layer Outputs and (b) t-SNE on Raw Input Data.

7 Conclusion

In conclusion, the task of classifying two concatenated MNIST images with an FCN model proved challenging, with the best accuracy falling below 90%. This suggests that FCNs are not ideal for this task, and a Convolutional Neural Network (CNN) might offer better performance, as CNNs excel at image data. The performance of balanced vs. imbalanced models showed that imbalanced data leads to more robust results across varying test sets. Additionally, the weak linear classifier (logistic regression) performed much worse on concatenated images than on individual ones,emphasizing the challenges faced by simple models when dealing with more complex input configurations. Lastly, t-SNE visualizations indicated that feature extraction significantly improved cluster performance, as raw input data was harder to classify effectively.

References

[1] Liu, W., Tsang, I.W., Müller, K.-R. *An Easy-to-Hard Learning Paradigm for Multiple Classes and Multiple Labels*. Journal of Machine Learning Research, 18, 1-38, 2017.

[2] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and

Duchesnay, E., *Scikit-learn: Machine Learning in Python*, Journal of Machine Learning Research, 12, 2825-2830, 2011.

[3] van der Maaten, L. and Hinton, G., *Visualizing Data using t-SNE*, Journal of Machine Learning Research, 9, 2579-2605, 2008.

8 Appendix

Based on the original text and instructions provided by the author, texts in this report was generated by ChatGPT 4o. The tool also assisted in formatting and refining the LaTeX code.