Addition Classification using MNIST: Fully Connected Neural Networks, Ensemble Methods, and Weak Classifiers

Shahab Yousef-Nasiri (sy475)
Department of Physics, University of Cambridge
Word Count: 2998

Abstract

We trained a fully connected neural network (FCNN) on pairs of MNIST images to address the addition classification problem. The pipeline involved data preprocessing, including normalization, augmentation, and the creation of paired datasets. Hyperparameter tuning was conducted using Optuna, resulting in a final model accuracy of 94.3% on the test set, with optimal parameters selected for dropout rate, learning rate, number of neurons, and layers. Four experiments were performed to explore and investigate key aspects of the architecture: (1) determining the optimal combination of neurons and layers, (2) investigating the depth-to-width trade-off in FCNN architecture, (3) optimizing dropout rates, and (4) investigating the relationship between optimal learning rates and batch sizes through experimental results. Additionally, we compared the performance of the FCNN against support vector machines (SVMs) and random forest ensembles, which achieved lower accuracies of 19.0% and 74.0%, respectively. To further analyze model performance, two weak linear classifiers were trained—one on stacked images (56x28) and the other on individual images (28x28) sequentially. We evaluated their test performance with varying training sample sizes, with a maximum score of 19.2% and 83.4% for the two models, revealing over fitting issues in the 56x28 model for smaller datasets. Finally, we visualized the t-SNE distribution of class embeddings in the penultimate layer of the FCNN and compared it to the representation obtained by applying t-SNE directly to the input data, optimizing perplexity for interpretability.

Contents

1 Introduction

Fully Connected Neural Networks (FCNNs) are one of the foundational architectures in deep learning and are particularly well-suited for tasks requiring multi-class classification [1]. An FCNN consists of layers of neurons, where each neuron in one layer is connected to every neuron in the subsequent layer. This dense connectivity enables the network to learn complex non-linear mappings between input features and output classes.

1.1 Structure of FCNNs

The architecture of an FCNN includes an input layer, one or more hidden layers, and an output layer. Each neuron in a given layer applies a weighted linear transformation to its inputs, followed by a non-linear activation function. Mathematically, for a single layer:

$$z^{(l)} = W^{(l)} \cdot a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = \sigma(z^{(l)})$$

where $W^{(l)}$ and $b^{(l)}$ are the weights and biases for layer l, $a^{(l-1)}$ is the output of the previous layer, $\sigma(\cdot)$ is the activation function (e.g., ReLU, sigmoid, or tanh), and $z^{(l)}$ represents the pre-activation values for the neurons in layer l.

For multi-class classification tasks, the output layer typically uses the softmax function, which converts raw scores (logits) into probabilities for each class. If the output logits are denoted as z_k for class k, the softmax function is defined as:

$$p_k = \frac{\exp(z_k)}{\sum_{j=1}^{C} \exp(z_j)}$$

where C is the total number of classes. The predicted class is the one with the highest probability, i.e., $\hat{y} = \arg\max_k p_k$.

1.2 Loss Function for Multi-Class Classification

The learning process in an FCNN involves minimizing a loss function, which quantifies the difference between the predicted probabilities and the true labels. For multi-class classification, the categorical cross-entropy loss is used. It is defined as:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{C} y_{i,k} \log(p_{i,k})$$

where N is the number of samples, $y_{i,k}$ is a binary indicator (1 if the true class for sample i is k, 0 otherwise), and $p_{i,k}$ is the predicted probability for class k for sample i.

1.3 Optimization and Training

The optimization of network weights and biases was carried out using gradient-based methods, including stochastic gradient descent (SGD), RMSprop, and Adam [2, 3, 4]. Table 1 provides a concise summary of the optimizers used, along with their mathematical formulations.

Table 1: Summary of Optimizers Used in the Study

Optimizer	Update Formula
SGD (Stochastic Gradient Descent)	$\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$
RMSprop (Root Mean Square Propagation)	$\theta \leftarrow \theta - \frac{\eta}{\sqrt{v_t + \epsilon}} \frac{\partial \mathcal{L}}{\partial \theta}$, where $v_t = \beta v_{t-1} + (1-\beta)\left(\frac{\partial \mathcal{L}}{\partial \theta}\right)^2$
Adam (Adaptive Moment Estimation)	$\theta \leftarrow \theta - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$, where $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$, $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$

2 Methodology

2.1 Data Preprocessing and Augmentation

The MNIST dataset, consisting of grayscale images of handwritten digits, was used as the base dataset for creating paired images in the 56x28 format. Below, we detail the steps involved in preprocessing, augmentation, and the justifications for each.

2.1.1 Normalization

Each MNIST image is represented as a 28×28 grayscale image, where pixel intensities range from 0 to 255. For improved numerical stability and faster convergence during training, these pixel values were normalized to the range $[0, 1]$ by dividing each intensity by 255. This normalization reduces the risk of exploding/vanishing gradients during back-propagation and accelerates model convergence by allowing the optimizer to take appropriately sized steps in parameter space [5].

2.1.2 Pair Creation for 56x28 Images

To create the 56x28 paired dataset, two randomly selected 28×28 MNIST images were stacked vertically. Both the original pair order and the flipped order (i.e., swapping the positions of the two images) were included. The labels for the pairs were calculated as the sum of the individual image labels. Random pairing increases dataset diversity, which helps the model generalize to unseen data and including flipped pairs ensures the model does not develop positional biases based on the stacking order.

2.1.3 Data Augmentation

To artificially expand the dataset and introduce variability, several augmentation techniques were applied to the paired 56x28 images:

- **Random Rotation:** Images were rotated randomly by up to $\pm 30°$.
  - Rotation increases invariance to the orientation of handwritten digits.
- **Color Jitter:** Brightness and contrast of images were varied randomly.
  - Simulates real-world variations in scanning or lighting conditions.
- **Random Erasing:** Random sections of the image were occluded.
  - Mimics scenarios where parts of the digits are missing or obscured, forcing the model to infer the missing parts from the surrounding context.
- **Gaussian Noise:** Random Gaussian noise was added to pixel values.
  - Increases resilience to pixel-level variations, reducing overfitting.
- **Salt-and-Pepper Noise:** Random pixels were set to black (0) or white (1).
  - Simulates sharp, localized noise often encountered in real-world data.

Each augmentation was applied probabilistically, ensuring variability in the augmented dataset. The augmented images were then combined with the original paired images to form the final training dataset.

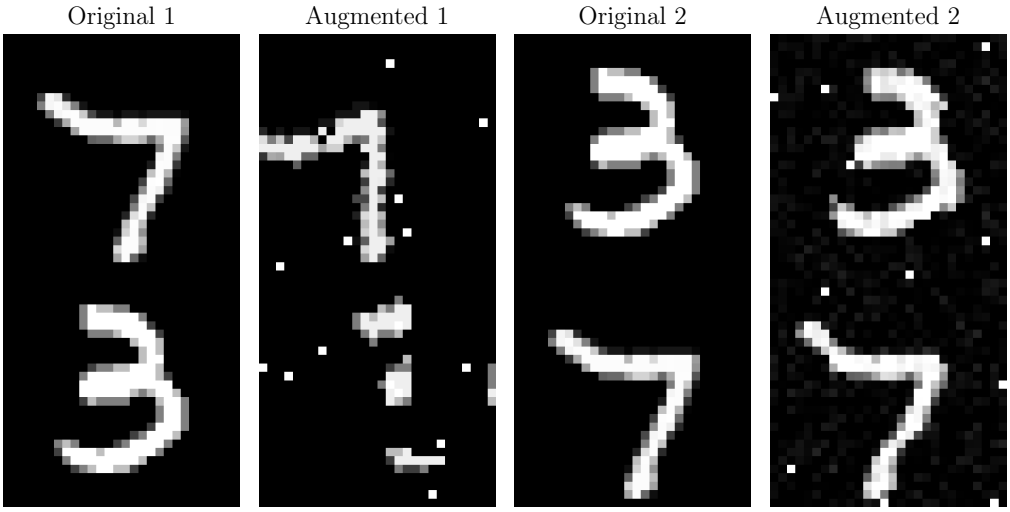


Figure 1: Example of data augmentation techniques applied to the 56x28 paired and flipped dataset.

2.1.4 Dataset Splitting

The final dataset consists of:

- **Training Set:** The training dataset comprises 60,000 unaltered paired and flipped images and an additional 60,000 augmented images of the same. This results in a total of 120,000 training samples.
- **Validation Set:** The validation dataset is formed by splitting 80% of the unaltered test set into 8,000 randomly paired and flipped images.
- **Test Set:** The remaining 20% of the unaltered test set, amounting to 2,000 randomly paired and flipped images, is used as the final test set.

2.1.5 Label Distribution Analysis

To ensure consistency and balance across the training, validation, and test sets, we analyzed the label distribution in each subset. As expected, a discrete triangular distribution was observed across all sets [6]. This distribution arises due to the nature of the paired label generation, where each pair's label is the sum of two MNIST digits. Mathematically, this follows the addition of two independent uniform distributions over $[0, 9]$, resulting in a triangular distribution with a peak around 9 and tails at 0 and 18, with a PMF:

$$P(Z = k) = \begin{cases} (k+1) & \text{for } 0 \leq k \leq 9, \\ (19-k) & \text{for } 10 \leq k \leq 18, \end{cases} \tag{1}$$

where Z is the sum of the two independent digit labels.

Importance of Label Distribution Analysis: Checking label distribution is crucial to:

- Ensure that the dataset subsets (training, validation, and test) represent the same underlying label distribution, preventing bias in evaluation.
- Confirm that no subset disproportionately lacks or overrepresents certain labels, which could affect model training and testing.

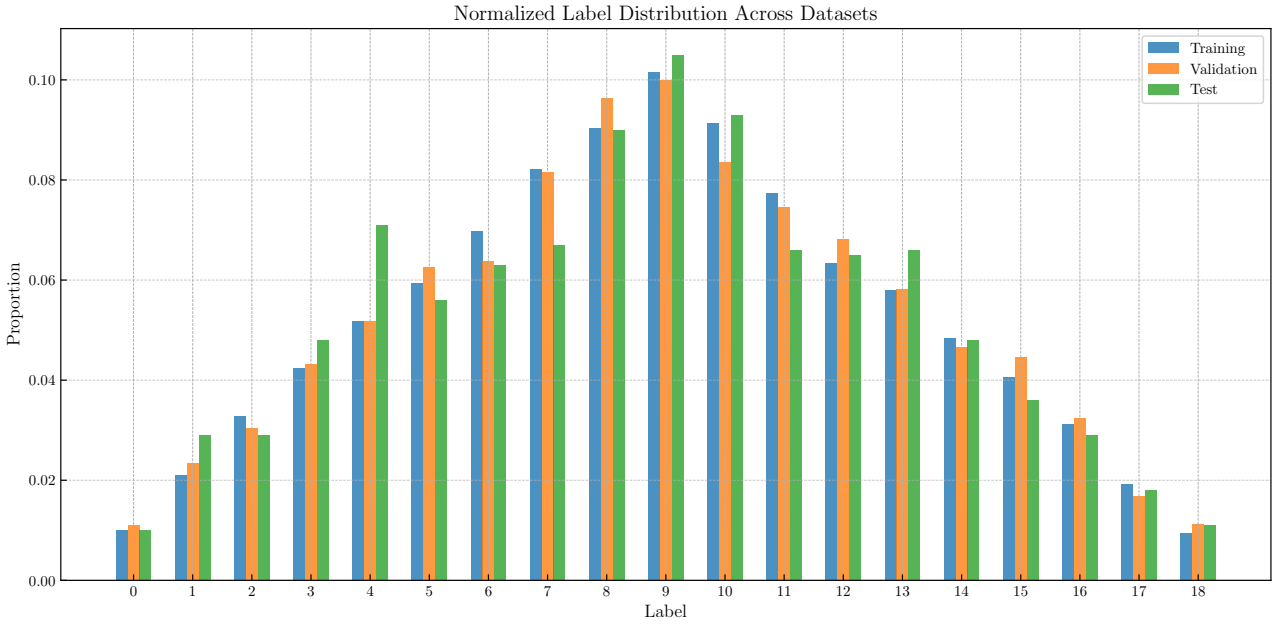


Figure 2: Label distribution across training, validation, and test sets, showing the discrete triangular distribution characteristic of digit pair sums.

2.2 Hyperparameter Optimization

This study employs **Optuna**, a state-of-the-art hyperparameter optimization framework, to efficiently explore the hyperparameter space [7]. Optuna leverages **Bayesian optimization**, which adapts its search strategy based on prior evaluations, significantly reducing computational overhead compared to exhaustive methods like grid or random search. Its adaptive mechanism identifies optimal configurations by focusing computational resources on the most promising regions of the hyperparameter space.

2.2.1 Bayesian Optimization

Bayesian optimization is a probabilistic method designed for optimizing expensive black-box functions [8]. It models the objective function $f(x)$ using a surrogate model, such as Gaussian processes, which approximates $f(x)$ based on prior evaluations. The optimization process iteratively selects the next set of hyperparameters x^* by maximizing an acquisition function:

$$x^* = \arg\max_x \alpha(x \mid \mathcal{D}), \tag{2}$$

where x^* is the next hyperparameter choice, $\alpha(x \mid \mathcal{D})$ is the acquisition function, and $\mathcal{D}$ represents the observed data (evaluations of $f(x)$). The acquisition function governs the trade-off between exploration (sampling unexplored regions of the hyperparameter space) and exploitation (refining promising regions).

The advantage of Bayesian optimization lies in its ability to balance computational efficiency and accuracy, making it particularly suitable for tuning neural networks with large hyperparameter spaces and costly evaluations.

2.2.2 Search Space and Dynamic Architecture

The hyperparameter space for the fully connected neural network (FCNN) included the following:

- **Dropout Rate:** Ranging from 0.1 to 0.5 to mitigate overfitting.
- **Learning Rate:** Spanning 1×10^{-4} to 1×10^{-1} to control the step size during gradient descent.

- **Batch Size:** Limited to 32 and 64 for computational efficiency.
- **Activation Functions:** ReLU, Tanh, and Sigmoid.
- **Optimizers:** Adam, SGD, and RMSprop.

Additionally, a **dynamic architecture** was implemented to explore variations in network depth and width. The number of layers was dynamically chosen between 2 and 8, with the number of neurons in the first layer ranging from 512 to 2048. To gradually reduce the width of the network, a reduction factor (α) between 0.5 and 0.9 was applied across successive layers:

$$n_i = n_{i-1} \cdot \alpha, \quad \text{where } n_i \text{ is the number of neurons in the } i\text{th layer.} \tag{3}$$

This approach allowed for flexible exploration of architectures, balancing model capacity and overfitting risks. Dropout layers were inserted between hidden layers to further regularize the network.

3 Fully Connected Neural Network

This section presents the performance of the best fully connected neural network (FCNN) model and analyzes the learned feature representations using t-SNE.

3.1 Model Results

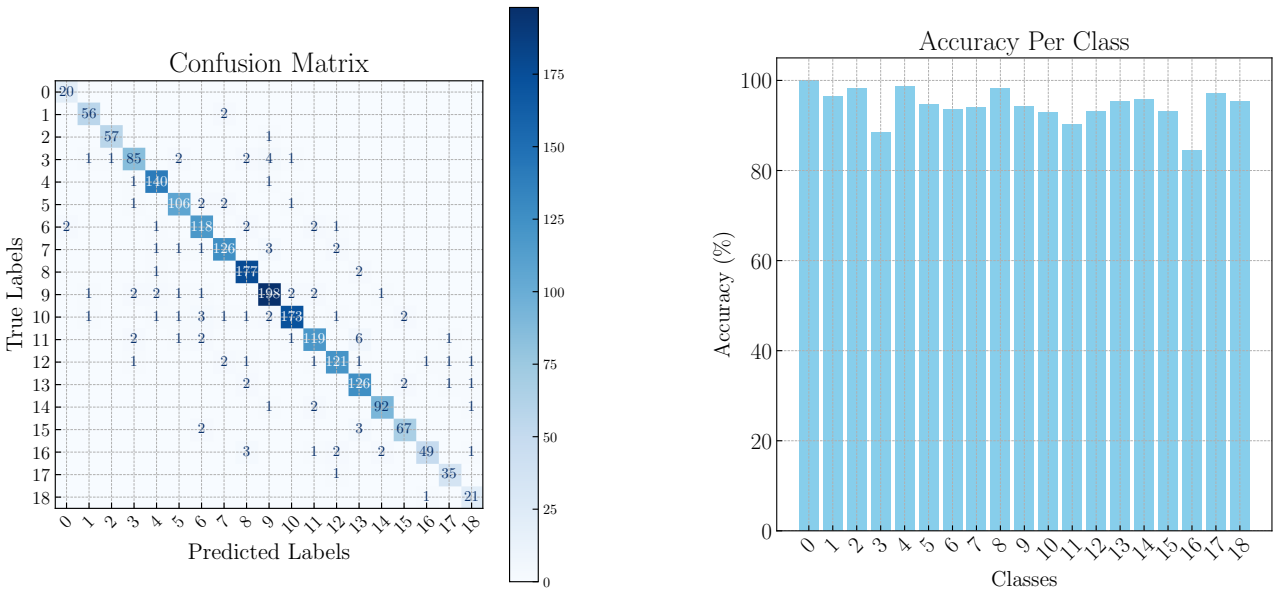
The optimal FCNN architecture was determined using Bayesian hyperparameter optimization. The selected hyperparameters and corresponding results are shown in Table 2. The optimization allowed 20 trials, each running for up to 25 epochs with early stopping applied (patience = 3).

Table 2: Selected hyperparameters for the best FCNN model.

Hyperparameter	**Value**
Number of Layers	5
Initial Neurons	557
Reduction Factor	0.623
Dropout Rate	0.167
Learning Rate (LR)	0.00092
Batch Size	64
Activation Function	ReLU
Optimizer	RMSprop
Computed Hidden Sizes	[557, 346, 215, 134, 83]

The model achieved a final test accuracy of **94.30%**, demonstrating excellent performance on the classification task. Figure 4 shows the training and validation loss curves for the above model. The curves reveal minimal over fitting and effective convergence of the model, as indicated by the plateauing of the validation loss.

Further analysis of the model's predictions is presented in Figure 3, which illustrates the confusion matrix and per-class accuracy. The confusion matrix (Figure 3a) provides insights into the misclassification patterns, while the per-class accuracy (Figure 3b) highlights the variability in model performance across different addition results. Both performance metrics indicate a strong alignment between the model's predictions and the true labels, demonstrating that the model performs well in distinguishing between classes with minimal misclassification.



(a) Confusion Matrix

(b) Per-Class Accuracy

Figure 3: Comparison of Confusion Matrix and Per-Class Accuracy for Model Evaluation.

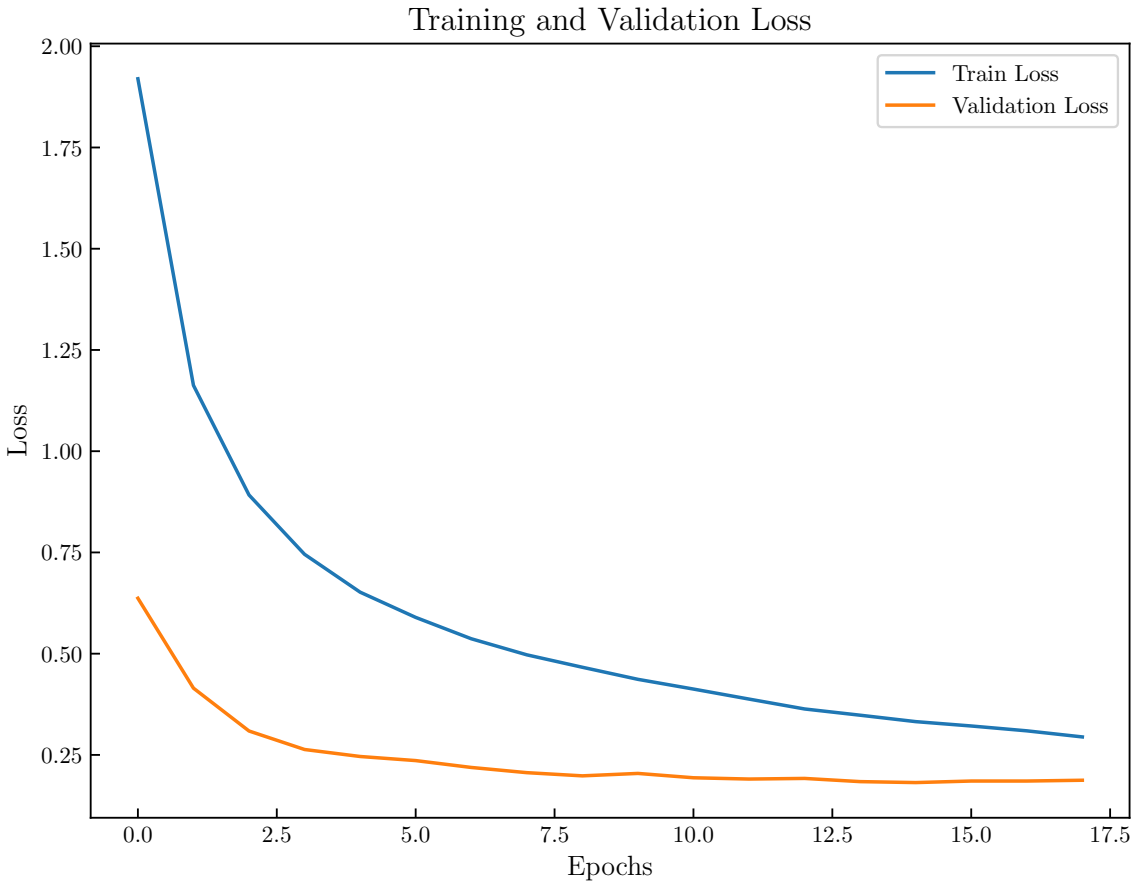


Figure 4: Training and validation loss curves for the best FCNN model.

3.2 t-SNE and Silhouette Optimization

To evaluate the quality of feature representations learned by the FCNN, t-distributed Stochastic Neighbor Embedding (t-SNE) was used to project the high-dimensional embeddings into a two-dimensional

space for visualization [9].

t-SNE is a dimensionality reduction technique designed to preserve the local structure of high-dimensional data in a low-dimensional space. Given a set of high-dimensional data points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, the algorithm defines conditional probabilities $p_{j|i}$ that represent the similarity between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ in the high-dimensional space. These probabilities are defined using a Gaussian kernel:

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

where σ_i is a perplexity parameter controlling the effective number of neighbors. In the low-dimensional space, similar probabilities q_{ij} are computed using a Student's t-distribution:

$$q_{ij} = \frac{\left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2\right)^{-1}}. \tag{5}$$

The algorithm minimizes the Kullback-Leibler (KL) divergence between the high-dimensional p_{ij} and low-dimensional q_{ij} distributions:

$$\mathrm{KL}(P \| Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{6}$$

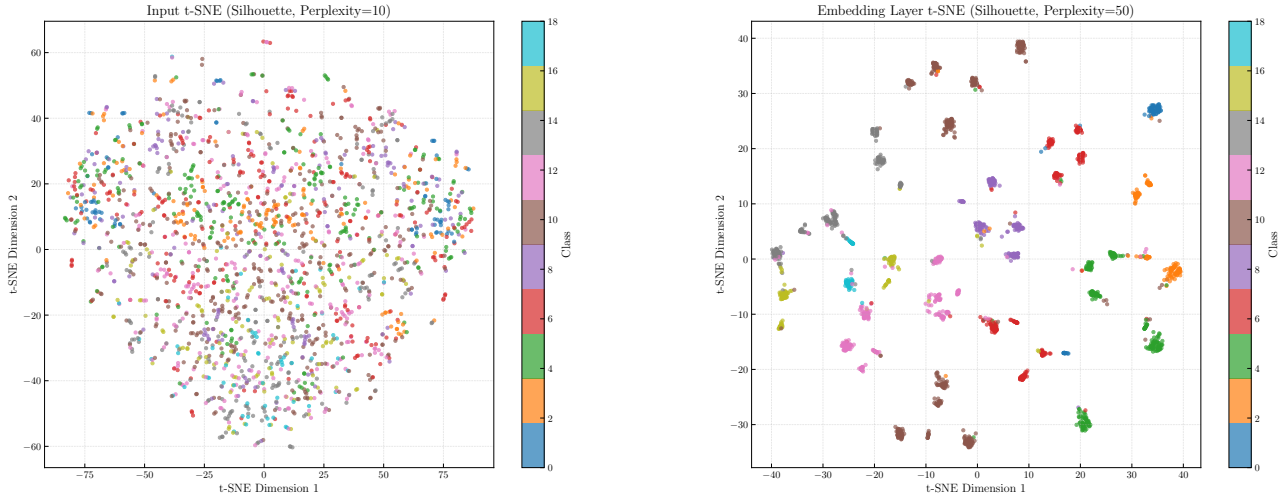
t-SNE visualizations were generated for both the input layer (raw data representation) and the embedding layer (learned feature representation).

To ensure the quality of the t-SNE embeddings, the silhouette score was used as a quantitative measure [10]. The silhouette score for a point i is defined as:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \tag{7}$$

where $a(i)$ is the mean intra-cluster distance (distance to points in the same cluster), and $b(i)$ is the mean nearest-cluster distance (distance to points in the nearest other cluster). The silhouette score ranges from -1 (poor separation) to 1 (perfect separation).

The silhouette score was optimized by tuning the t-SNE perplexity parameter, which controls the balance between local and global structure preservation. A perplexity value of 10 and 50 yielded the best results for the input and embedding layer respectively, as shown by the visual clarity of clusters in figure 5.



(a) Input Layer t-SNE Visualization

(b) Embedding Layer t-SNE Visualization

Figure 5: Comparison of t-SNE Visualizations for Input Layer and Embedding Layer. Each point represents an addition result, and colors indicate different classes.

4 Investigating FCNN Architectures and Training Dynamics

This section investigates the relationships between architectural choices, hyperparameters, and training dynamics in Fully Connected Neural Networks (FCNNs). Through a series of experiments, we explore the effects of key design parameters on network performance and training behavior. Rather than focusing solely on identifying optimal configurations (which was addressed in 2.2), the aim here is to uncover general trends, interactions between architectural parameters, and their influence on training efficiency, generalization, and model behavior. A default configuration model was used for all experiments, where parameters not in question were fixed as follows: hidden sizes [512, 256], activation function: ReLU, dropout rate: 0.1, learning rate: 0.005, and batch size: 64.

4.1 Impact of Neurons and Layers on Performance

The effect of the number of layers and neurons per layer on performance was analyzed to identify optimal regions of layer and neuron combinations. Figure 6 illustrates that validation loss decreases as depth (number of layers) increases, approaching its minimum at 3-5 layers with roughly 300-600 neurons per layer. Beyond this range, deeper and wider networks exhibit overfitting. Balanced architectures perform best, with moderate depth allowing the network to capture hierarchical patterns effectively and moderate width providing sufficient capacity to model complex features without overparameterization. In contrast, shallow networks (1-2 layers) underfit the data, failing to learn its complexity, while excessively deep or wide networks overfit by memorizing noise in the training data.

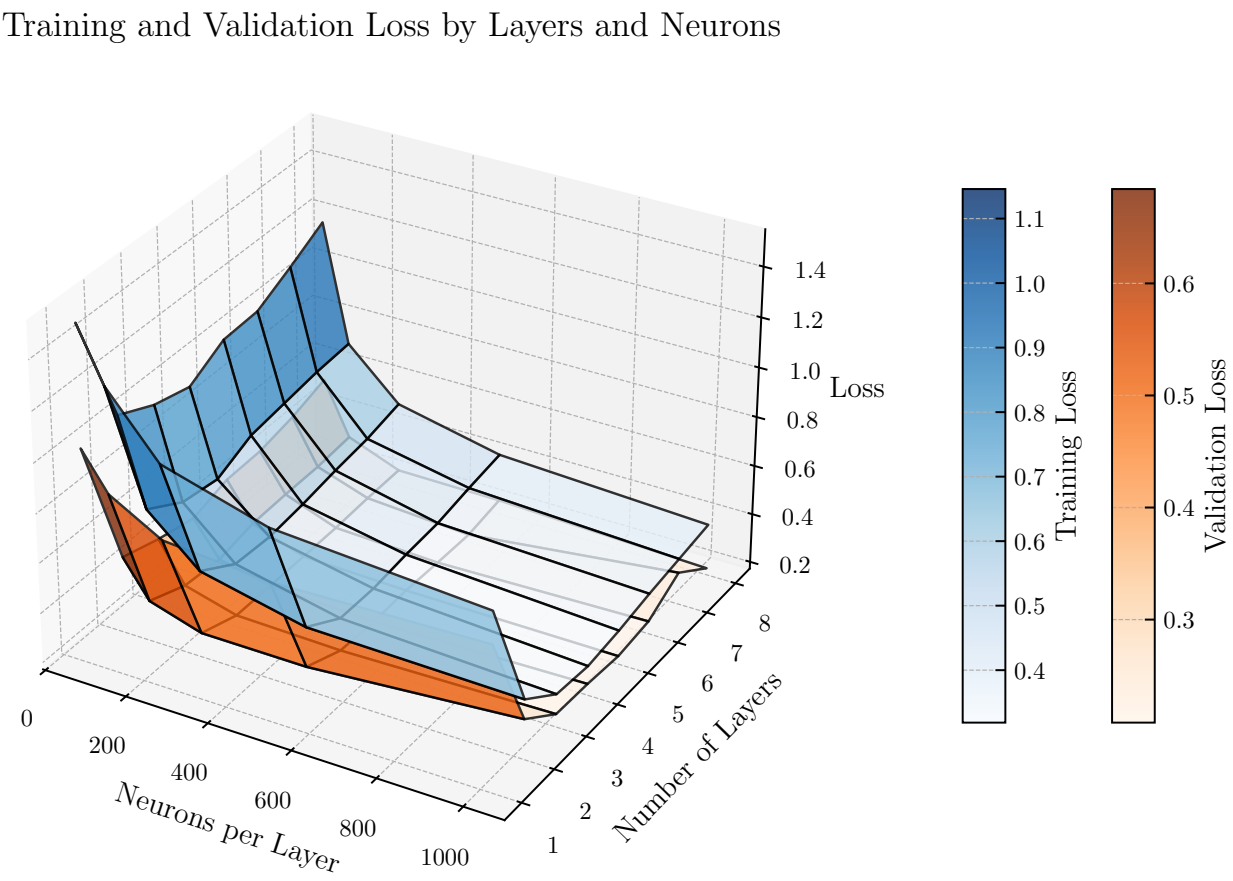


Figure 6: Validation accuracy as a function of the number of neurons and layers in the network.

4.2 Depth-to-Width Trade-off

For depth-width configurations in FCNNs, gradual reductions in layer widths consistently yielded better performance across all architectures. For 2-layer networks, [512, 128] and [512,256] achieved the lowest validation losses (0.2632) and (0.2761, outperforming abrupt changes like [1024, 64] (0.2999) or

overly wide configurations like [1024, 512] (0.4248), which likely led to inefficiencies or overfitting. In 3-layer networks, [1024, 512, 256] performed best (0.2034), with gradual reductions remaining superior to abrupt bottlenecks like [1024, 256, 64] (0.2365). For 4-layer networks, [1024, 512, 256, 128] (0.2053), confirmed that geometric reduction patterns (e.g., halving widths) generalized well as depth increased. However, the marginal gains from deeper architectures diminished with the 5 layer network [1024, 512, 256, 128, 64] (0.2060) suggesting diminishing returns with added layers.

Table 3: Validation Loss by Layer Structures

Validation Loss by Layer Structures			
2 Layers		**3 Layers**	
Hidden Sizes	**Validation Loss**	**Hidden Sizes**	**Validation Loss**
[512, 128]	0.2632	[1024, 512, 256]	0.2034
[512, 256]	0.2761	[1024, 256, 128]	0.2068
[1024, 64]	0.2999	[512, 256, 128]	0.2174
[256, 128]	0.3069	[1024, 512, 128]	0.2195
[1024, 128]	0.3136	[1024, 128, 64]	0.2203
[256, 64]	0.3158	[512, 128, 64]	0.2222
[512, 64]	0.3217	[1024, 512, 64]	0.2312
[1024, 256]	0.3666	[1024, 256, 64]	0.2365
[128, 64]	0.3753	[256, 128, 64]	0.2464
[1024, 512]	0.4248	[512, 256, 64]	0.2740
4 Layers		**5 Layers**	
Hidden Sizes	**Validation Loss**	**Hidden Sizes**	**Validation Loss**
[1024, 512, 256, 128]	0.2053	[1024, 512, 256, 128, 64]	0.2060
[1024, 512, 256, 64]	0.2056		
[512, 256, 128, 64]	0.2089		
[1024, 512, 128, 64]	0.2128		
[512, 256, 128, 64]	0.2166		

4.3 Dropout Rate and Overfitting Control

The effect of dropout rates on validation loss was analyzed for networks using ReLU and Tanh activation functions, as shown in Figure 7.

For ReLU networks, validation loss remains consistently low, between 0.2 and 0.3 across all dropout rates. This is due to:

- **Sparsity alignment:** ReLU's natural sparsity from zeroing out negative values complements the dropout mechanism, minimizing reliance on specific neurons.
- **Stable gradients:** ReLU avoids gradient saturation, ensuring gradients remain stable and effective even as dropout increases.

In contrast, for Tanh networks, validation loss rises sharply from 0.6 at 0% dropout to over 1.0 at 50%

dropout, explained by:

- **Gradient saturation:** Tanh saturates for large inputs, leading to vanishing gradients, which dropout further exacerbates.
- **Disrupted complexity:** Tanh's reliance on nonlinear interactions is destabilized by neuron deactivation at higher dropout rates.

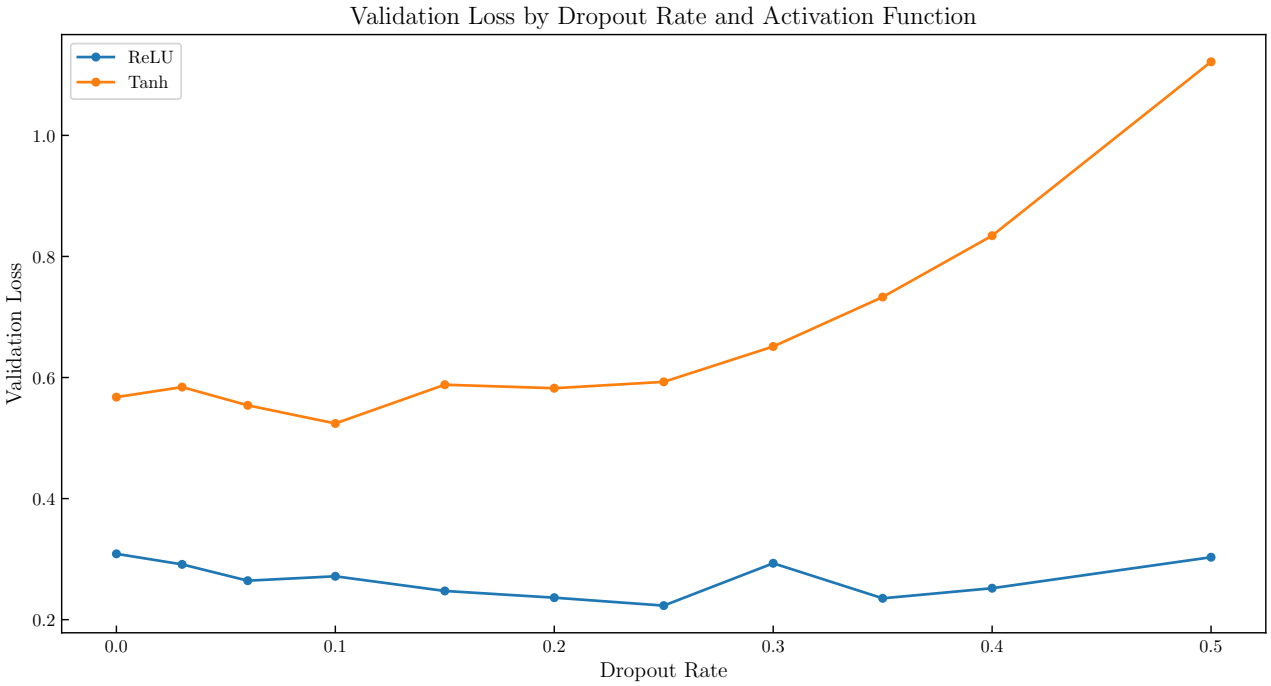


Figure 7: Validation accuracy and loss for different dropout rates, measured with ReLU and Tanh activation functions.

4.4 Batch Size vs. Learning Rate Relationship

The relationship between optimal batch size and learning rate was analyzed by conducting a grid search over combinations of batch sizes and learning rates, as shown in Figure 9. For each combination, the validation loss was evaluated, and the ratio of learning rate to batch size (η/S) was computed. The results revealed that higher η/S ratios consistently correlated with better generalization performance [11]. This trend is attributed to the properties of the Hessian matrix associated with the loss function. Specifically, higher η/S ratios are associated with a lower maximum eigenvalue of the Hessian and a reduced Frobenius norm, indicating flatter minima in the loss landscape. Flatter minima are known to promote better generalization by reducing sensitivity to perturbations in the input data and ensuring smoother optimization dynamics. Notably, the validation loss remained relatively consistent across different batch sizes and learning rates, provided that the η/S ratio was held constant, with an optimal ratio ranging from 10^{-4} to 10^{-3}.

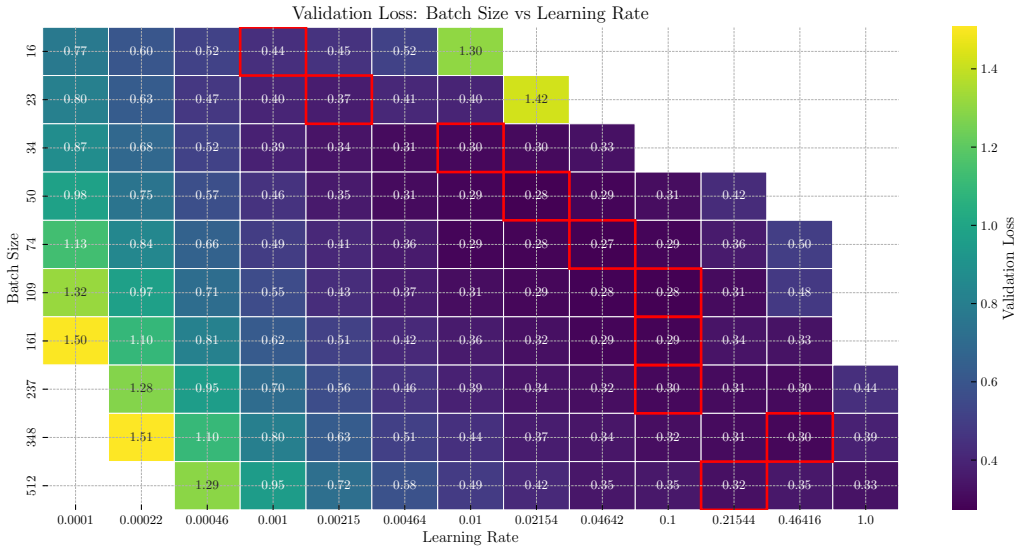


Figure 8: Heatmap of validation loss for different learning rate and batch size combinations. Combinations with the lowest loss per batch size are highlighted in red.

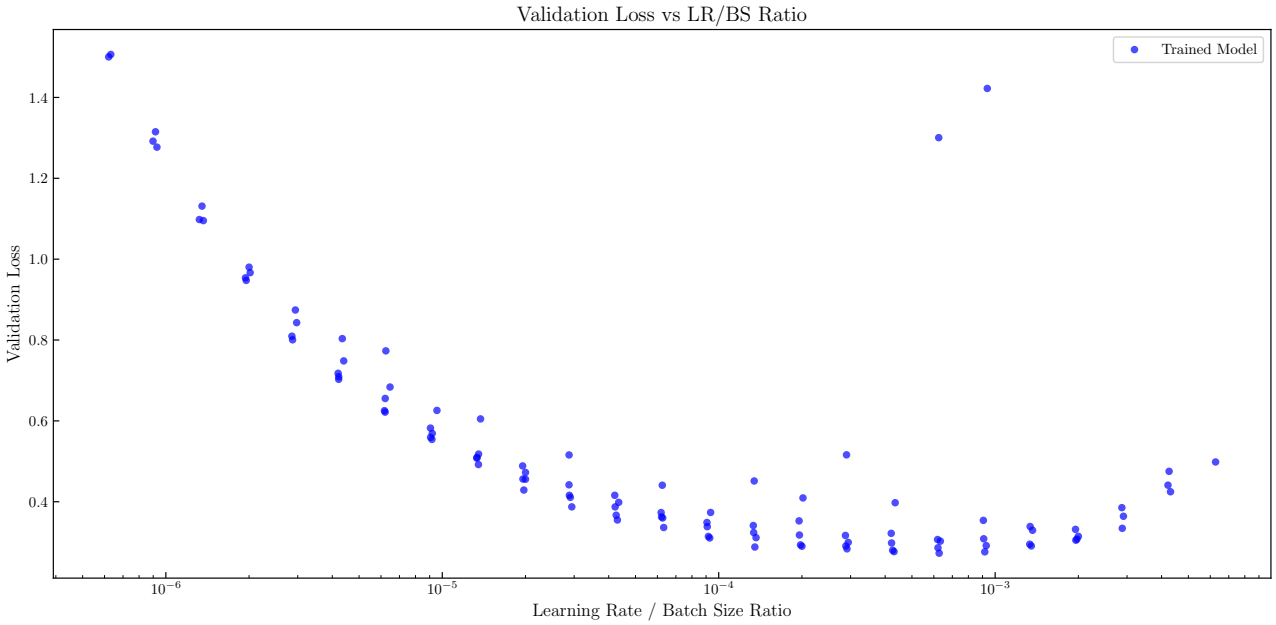


Figure 9: Validation loss versus the ratio of learning rate to batch size (η/S). Higher η/S ratios are associated with lower validation loss, reflecting better generalization and wider minima in the loss landscape.

5 Classical Machine Learning and Weak Linear Classifiers

This section evaluates the performance of classical machine learning methods—Random Forest Classifiers (RFC) and Support Vector Machines (SVM)—as well as Weak Linear Classifiers (WLCs) for the addition classification problem. These approaches are compared against the fully connected neural network (FCNN) in terms of accuracy and computational efficiency.

5.1 Weak Linear Classifiers

Weak Linear Classifiers are simple models with a single linear layer that maps input features directly to class probabilities. Two configurations of WLCs were evaluated:

- **56x28 Stacked Images:** The WLC was trained directly on paired and flipped images stacked vertically into a 56x28 input.
- **28x28 Sequential Images:** Predictions were generated sequentially for two 28x28 images, and their predicted labels were summed to classify the addition result.

Performance of these classifiers across varying training sizes is shown in Figure 10, with the 56x28 and 28x28 models achieving a best accuracy of **19.2%** and **83.4%**, respectively. For the 56x28 model, the high validation loss relative to the training loss, particularly for smaller datasets, combined with poor accuracy, indicates simultaneous overfitting issues (lack of generalization) and also indicates severe underfitting problems (insufficient learning capacity due to the simplicity of the model combined with the issue of fitting to noise) across the entire dataset sizes. The 56x28 weak linear classifier struggles with this task because its simple architecture lacks the ability to model the complex dependencies introduced by the paired structure of the input images. The interaction between the stacked features requires a more expressive model to effectively capture these relationships. While the 28x28 model performs better, it still faces limitations inherent to weak linear classifiers, such as an inability to effectively learn non-linear patterns critical for more nuanced classification tasks.

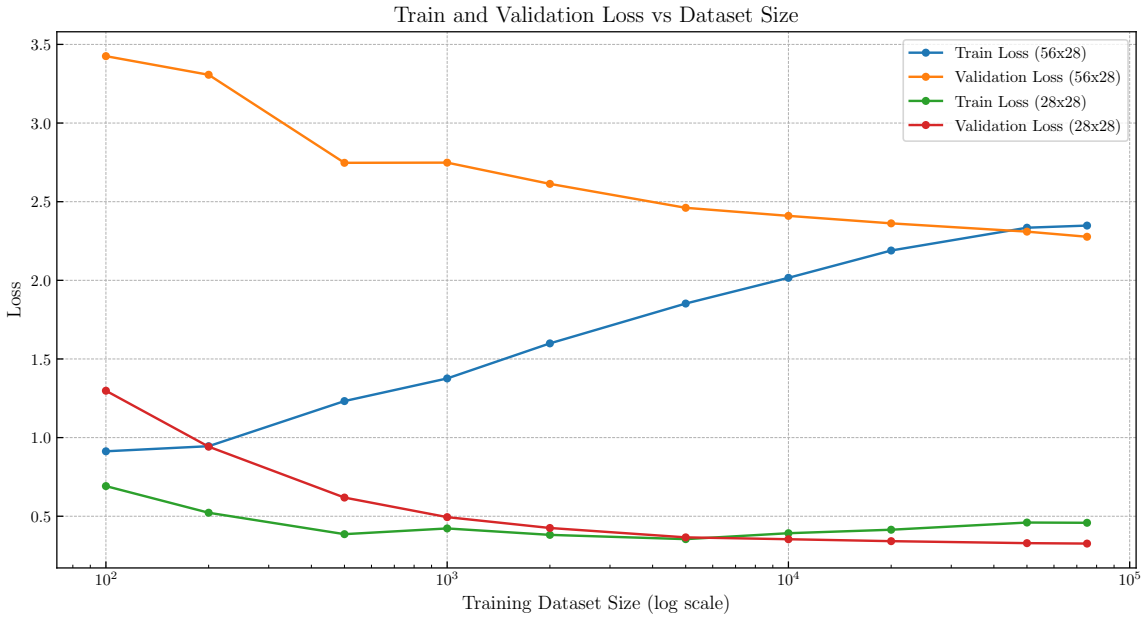


Figure 10: Performance of Weak Linear Classifiers across different training sizes, for the 56x28 and 28x28 sequential models.

5.2 Random Forest Classifier (RFC)

Random Forest is an ensemble learning method that combines predictions from multiple decision trees. It is effective for datasets with structured relationships and categorical targets. Optuna was used to tune the hyperparameters, as summarized in Table 4. The best-performing RFC achieved a test accuracy of **74.0%** with the parameters shown in the table.

Table 4: Random Forest Hyperparameter Ranges and Optimal Parameters

Hyperparameter	**Range**	**Best Value**
Number of Trees (`n_estimators`)	50 to 150	149
Maximum Depth (`max_depth`)	10 to 20	17
Minimum Samples to Split (`min_samples_split`)	2 to 8	3
Minimum Samples per Leaf (`min_samples_leaf`)	1 to 4	2

This model demonstrated computational efficiency with a training time of **10 minutes**, making it suitable for scenarios where quick training is necessary.

5.3 Support Vector Machine (SVM)

Support Vector Machines find a hyperplane in a high-dimensional space to separate data points by class. Due to the dataset's high dimensionality, Principal Component Analysis (PCA) was applied to reduce the feature space to 592, whilst retaining 95% of the total variance in the data, significantly improving training efficiency. The SVM was trained using a subset of 50,000 stratified examples from the training set.

Table 5: SVM Configuration and Performance Metrics

Parameter	**Value**
Regularization Parameter (`C`)	1.0
Kernel	Linear
PCA Components	300
Training Subset Size	20,000
Test Accuracy (%)	19.0

The SVM achieved a test accuracy of **19.0%**; its limitations in scalability and flexibility, particularly when dealing with highly dimensional data, were evident compared to neural networks, both in terms of performance and computational efficiency.

5.4 Performance Comparison

Table 6 compares the performance of the RFC, SVM, and FCNN models. This comparison highlights the advantages of deep learning methods for capturing complex relationships in image-based datasets.

Table 6: Performance Comparison of Classical Models and Neural Network

Model	**Test Accuracy (%)**	**Training Time (m)**
Weak Linear Classifier (56x28)	19.2	3
Weak Linear Classifier (28x28)	83.4	3
Random Forest Classifier (RFC)	74.0	10
Support Vector Machine (SVM)	22.0	44
Fully Connected Neural Network (FCNN)	94.3	88

6 Conclusion

This study demonstrated the superior performance of an (FCNN) for addition classification, while classical methods like SVM and RFC struggled to model the hierarchical and spatial relationships inherent in the data. Future improvements could include using Convolutional Neural Networks (CNNs), which are better suited for image-based tasks due to their ability to leverage spatial hierarchies. Additionally, incorporating GPU acceleration could enable more extensive experimentation and optimization, further enhancing model performance and generalization.

References

[1] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.

[2] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

[3] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.

[4] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade: Second edition*, pages 437–478. Springer, 2012.

[6] CC Kokonendji, Tristan Senga Kiesse, and Silvio S Zocchi. Discrete triangular distributions and non-parametric estimation for probability mass function. *Journal of Nonparametric Statistics*, 19(6-8):241–254, 2007.

[7] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[8] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[9] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[10] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[11] Stanislaw Jastrzębski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. Width of minima reached by stochastic gradient descent is influenced by learning rate to batch size ratio. In *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III 27*, pages 392–402. Springer, 2018.

Appendix

A Declaration of Autogenerative Tools

In this project, I used GitHub Copilot as a coding assistant. I mainly used it to generate the plots in Figures, from 1 to 10, alongside the generation of the 6 tables. I also used it within my code to create comments, docstring my functions, debug and format markdown cells explaining the different sections in the notebook.

In regards to the report, I used the Overleaf AI which uses ChatGPT to help write the mathematics in latex format, and also used it to correctly format various sections of the report as I am quite new to LaTeX.