Project 2: Kaggle 50-Image Classifier Model

COMP 432: Machine Learning

By
Waddah Daker, 40233651
Karim Nafiz, 40266284
Andrew Pulsifer, 40234525

1. Introduction

This report outlines the process by which we developed a high-accuracy 50-class household-object classification model utilizing multilayer perceptrons (MLPs) constructed with the Python library PyTorch. Since the project forbade the use of pretrained models and transfer learning, it centered on how we enhanced the model's performance by carefully experimenting with its architecture design, regularization and normalization techniques, and optimization strategies.

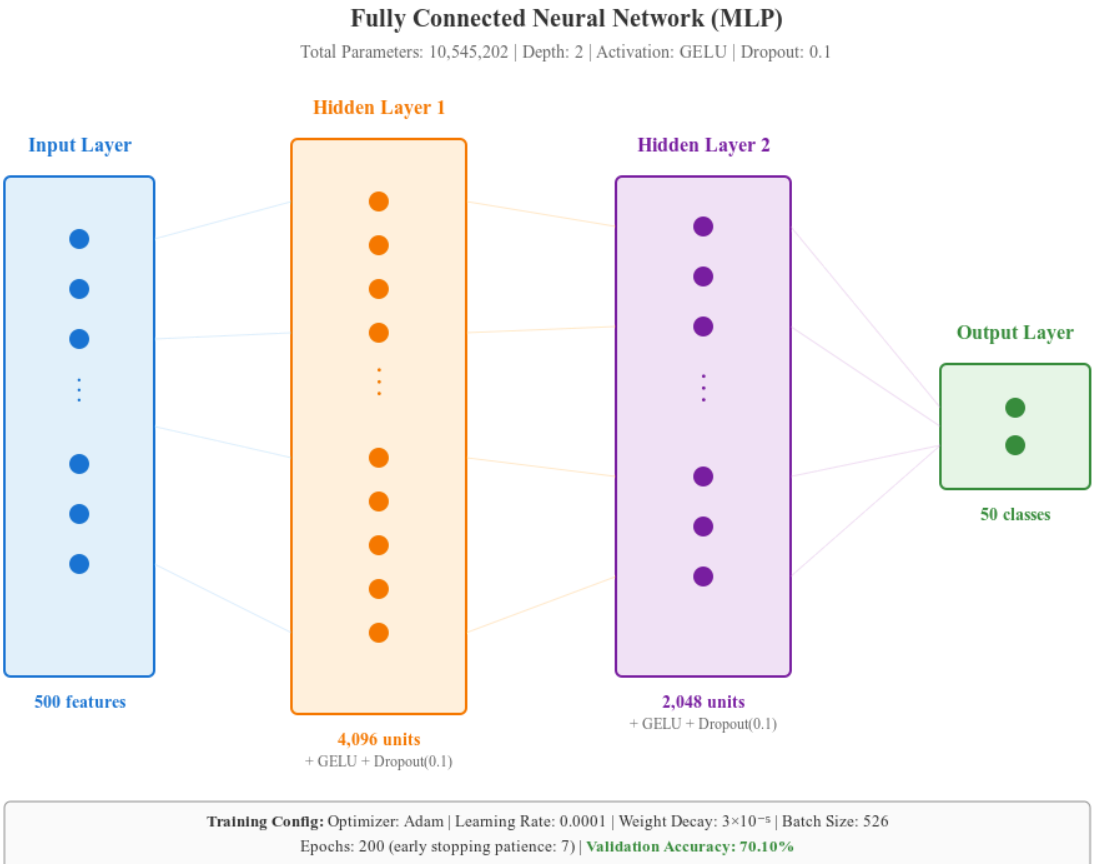
Each member began with a simple baseline MLP and was tasked to systematically experiment with different model parameters—such as layer width, network depth, activation functions, learning rates, batch sizes, optimizers, and normalization strategies—to determine which combinations yielded the highest accuracies. With each experimental run, we gained valuable insight into underfitting, overfitting, and training stability, allowing us to refine our approach with progressively more effective models.

Through this iterative process, we identified the key characteristics that consistently led to better performance. Using the insight obtained from these iterations, we ultimately developed the model that achieved the highest test accuracy on our dataset, which will serve as the primary subject of analysis in this report.

2. Model Architecture and Training

This section will detail the architecture of our best-performing MLP model, alongside the key design choices and training methodology that helped it achieve such strong generalization across the 50 household object classes.

2.1. Architecture Details



Architecture Overview

Our best-performing model follows a shallow-wide MLP structure designed to maximize representational capacity without relying on excessive depth to avoid overfitting. The network contains two extremely wide hidden layers (4096 and 2048 units), each followed by GELU activation and 0.10 dropout. This design results in approximately 10.5 million trainable parameters, allowing the model to learn complex relationships within the 500-dimensional input features while maintaining stable training dynamics due to GELU's smooth nonlinearity and the regularizing effect of dropout.

Model Structure

- Input dimension: 500
- Hidden Layer 1: 500 → 4096, GELU, Dropout 0.10
- Hidden Layer 2: 4096 → 2048, GELU, Dropout 0.10
- Output Layer: 2048 → 50

Activation and Regularization Choices

- Activation: GELU activation was selected due to its smoother non-linearity and improved performance compared to ReLU in deep/wide MLPs.
- Regularization: Dropout provided regularization by preventing co-adaptation of neurons, especially important given the high layer widths.
- Normalization: No normalization was implemented in this model, as it yielded the best results. To prove this, Andrew ran models with different normalization techniques on the validation set and logged their results. Here are the aforementioned results:

```
========== Training with Raw ==========
epoch 01 | train_loss 3.0504 | val_acc 0.3193
epoch 02 | train_loss 2.3873 | val_acc 0.3760
                    ...
epoch 29 | train_loss 0.0101 | val_acc 0.5573
epoch 30 | train_loss 0.0089 | val_acc 0.5571
  Best validation accuracy for Raw: 0.5582

========== Training with Z-score ==========
epoch 01 | train_loss 2.9331 | val_acc 0.3760
epoch 02 | train_loss 1.9077 | val_acc 0.4514
                    ...
epoch 13 | train_loss 0.1353 | val_acc 0.5220
epoch 14 | train_loss 0.1305 | val_acc 0.5201
```

```
Early stopping at epoch 14 (best val_acc=0.5229)
  Best validation accuracy for Z-score: 0.5229

========== Training with Clip(p99.5)->Z ==========
  epoch 01 | train_loss 2.9007 | val_acc 0.3819
  epoch 02 | train_loss 1.8770 | val_acc 0.4567
                      ...
  epoch 17 | train_loss 0.2543 | val_acc 0.5025
  epoch 18 | train_loss 0.2180 | val_acc 0.5096
Early stopping at epoch 18 (best val_acc=0.5207)
Best validation accuracy for Clip(p99.5)->Z: 0.5207

  ========== Training with log1p->Z ==========
  epoch 01 | train_loss 2.8677 | val_acc 0.3896
  epoch 02 | train_loss 1.8630 | val_acc 0.4572
                      ...
  epoch 16 | train_loss 0.1949 | val_acc 0.5123
  epoch 17 | train_loss 0.1863 | val_acc 0.5092
Early stopping at epoch 17 (best val_acc=0.5198)
  Best validation accuracy for log1p->Z: 0.5198

  ========== Training with sqrt->Z ==========
  epoch 01 | train_loss 2.9025 | val_acc 0.3774
  epoch 02 | train_loss 1.8978 | val_acc 0.4512
                      ...
  epoch 18 | train_loss 0.1877 | val_acc 0.5065
  epoch 19 | train_loss 0.1641 | val_acc 0.5094
Early stopping at epoch 19 (best val_acc=0.5166)
  Best validation accuracy for sqrt->Z: 0.5166
```

2.2. Training Methods

The model was trained using supervised multi-class classification with the goal of mapping 500-dimensional feature vectors to 50 household object categories. This section summarizes the training procedure, hyperparameters, and optimization techniques used to achieve the final results.

Dataset Split

The dataset consisted of 500-dimensional feature embeddings and 50 class labels. Since the team concluded that no normalization yielded the highest results, the final model was fed with raw input data.

- Training set: 70%
- Validation set: 15%
- Testing set: 15%

The validation set was used exclusively for hyperparameter tuning, early stopping decisions, and tracking generalization performance.

Batching and Data Loading

Training was performed in batches of 1024, chosen specifically to stabilize gradient updates in our model's extremely wide layers and to fully utilize the GPU's parallelism.

- They also reduced the frequency of weight updates, which helped prevent overfitting
- PyTorch DataLoader objects handled shuffling, batching, and device transfers.

Loss Function

Training optimized the CrossEntropyLoss, a natural choice for multi-class classification tasks due to:

- Its compatibility with softmax logits
- Strong convergence properties

This provided a reliable measure of learning progress across both training and validation sets.

Optimization Algorithm and Scheduler

To ensure fast, stable, and generalizable training for a model with over 10 million parameters, we employed a combination of the Adam optimizer and a learning-rate scheduler. This combination allowed the model to converge rapidly in the initial epochs while refining the weights as the training advanced, thus avoiding both instability and overfitting.

The Optimal learning rate was found to be 0.0001 after multiple experimental runs. More low values, for example, 0.00001, made the model converge too slowly with worse validation accuracy. Larger values, for instance, 0.01, caused unstable updates or sharp oscillations because the model has very wide hidden layers (4096 → 2048). This chosen value best balances fast learning and stable gradients.

To further refine the optimization, the training loop employed a ReduceLROnPlateau-style scheduler that automatically reduced the learning rate when validation accuracy stopped improving. This helped the optimizer take large steps early in training—when the loss landscape is smooth—and progressively smaller, more precise steps later, preventing overshooting and improving generalization.

Training Loop Algorithm

The problem we noticed in most of the early runs was related to having a fixed number of epochs. In this case, we either overfitted the model by training past its peak or underfitted the model by stopping early before convergence was complete. So, we have designed a training pipeline that will dynamically stop at the point where the model reaches the generalization peak.

During each epoch, the training loop evaluated the model on the training and validation sets. One epoch is a full forward and backward pass through the training set, updating parameters using the Adam optimizer. On completion of all training batches, the model was evaluated on the validation set without gradient update. The validation loss and accuracy were the main indicators of generalization and were used for checkpointing the best-performing model.

In order to avoid unnecessary over-training, we applied early stopping with a patience value of 10 epochs. If the model did not achieve a meaningful improvement in validation accuracy within this window, training was stopped. This approach was most helpful since the model's large hidden-layer widths made it capable of memorizing the training set if allowed to run too long.

Because this training structure relied on the performance of the validation set, rather than relying on a fixed count of epochs, this ensured that the saved model represented the point of optimal generalization and minimized overfitting and wasted computation.

3. Evaluation Methods & Error Analysis

This section evaluates the performance of our final and best-performing model. Performance is assessed using standard multi-class classification metrics, confusion analysis, and per-class behavior.

3.1. Evaluation Metrics

The final model was evaluated on the held-out test split using a set of metrics that capture different aspects of multi-class performance. Since the dataset contains 50 balanced classes, we used macro-averaged metrics so that each label contributes equally.

Metrics Computed:

- Accuracy – overall proportion of correct predictions
- Macro Precision – average precision across all 50 classes
- Macro Recall – average class-level sensitivity
- Macro F1-Score – harmonic mean of macro precision and recall
- Cross-Entropy Loss – measures prediction confidence and calibration
- Confusion Matrix – reveals class-specific error patterns
- Per-Class Recall – identifies the strongest and weakest classes

Using macro-averaging is critical because it prevents the model from appearing artificially strong if it performs well on only a subset of the classes.

Test Performance Summary:

Metric	Value
Accuracy	65.95%
Macro Precision	0.6637
Macro Recall	0.6595
Macro F1-Score	0.6589
Loss	1.3736

Overall, the model shows solid generalization for a 50-class problem trained entirely from scratch, especially considering that the dataset embeddings are high-dimensional and the classes can be visually similar.

3.2. Error Analysis

To analyze how the model behaves across all 50 classes, we inspected the normalized confusion matrix and extracted the most recurrent misclassification patterns.

Top 5 Most Confused Class Pairs

True Class	Predicted Class	Confusion Rate
2	3	8.1%
20	22	6.6%
31	30	6.6%
22	24	6.6%
14	12	6.1%

These pairs suggest that several classes have overlapping feature patterns in the input space. For example, classes like 2/3 or 20/22 may correspond to objects that are visually or semantically similar (similar shapes, textures, or usage context), making them harder to distinguish for our MLP.

Weakest Classes (Lowest Recall)

Class	Recall
9	0.476
6	0.480
40	0.490
29	0.507
16	0.519

These classes have a recall below 0.52, meaning that nearly half of their samples are misclassified. Likely reasons include:

- Their features overlap with several other classes.
- They may have higher intra-class variability or more ambiguous samples.
- The model's current capacity and regularization may favor more common or cleaner patterns, leaving these classes underrepresented in the learned decision boundaries.

Strongest Classes (Highest Recall)

Class	Recall
13	0.803
11	0.827
5	0.836
48	0.855
46	0.876

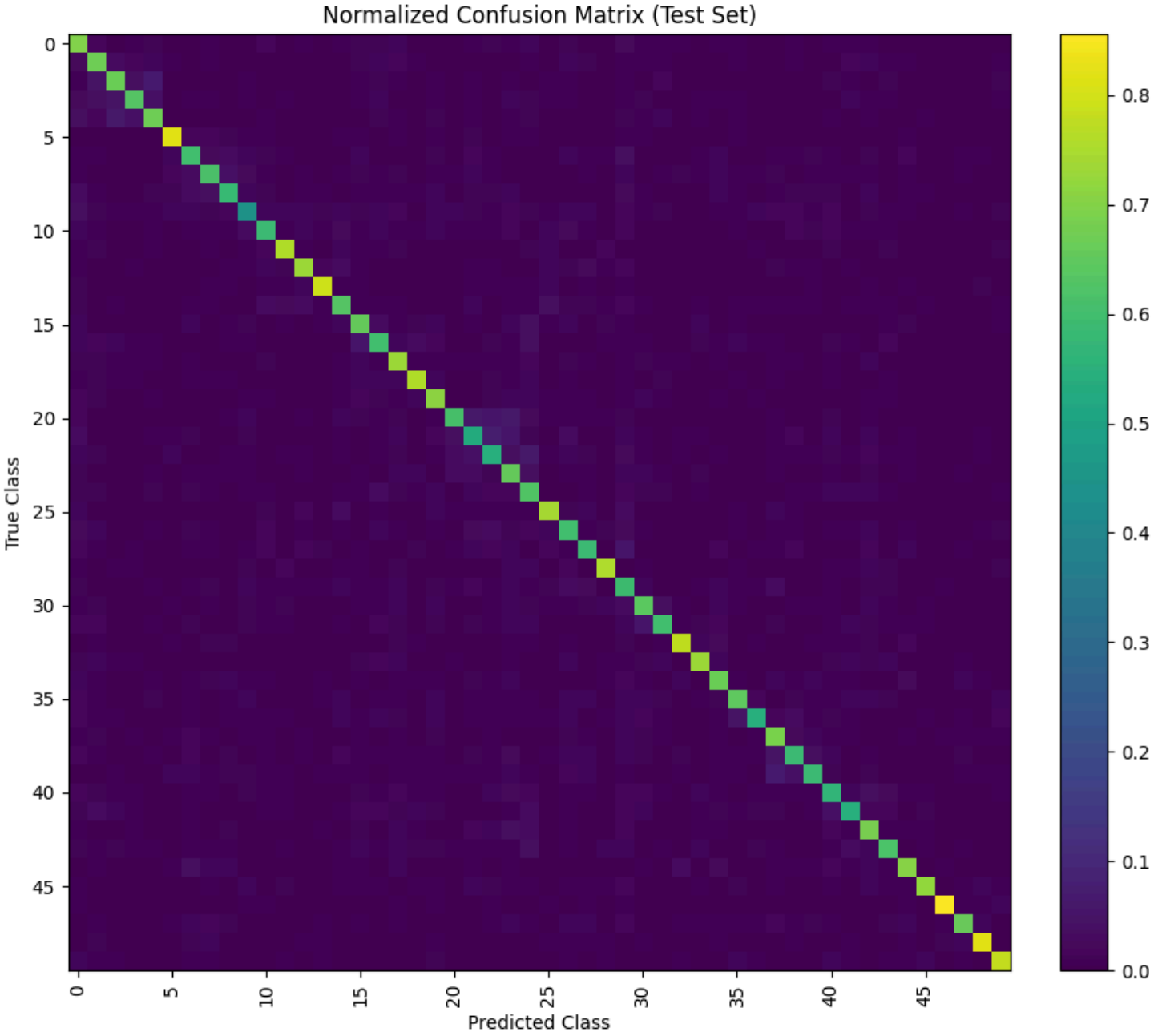
These “strong” classes are recognized very reliably, with a recall above 0.80. That suggests:

- Their feature representations are distinct and well-separated from other classes.
- The model consistently learns stable decision regions for these categories, even under dropout and weight decay.

Together, the weakest and strongest classes illustrate how the model performs very differently across the 50-class space: some categories are almost linearly separable in the learned representation, while others remain highly entangled.

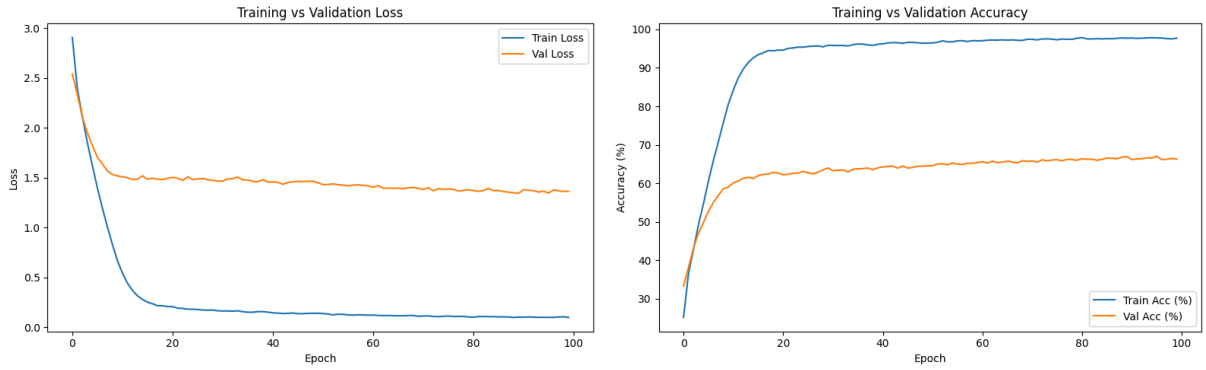
3.3. Confusion Matrix Heatmap

A normalized confusion matrix heatmap was generated to visualize model performance across all 50 classes. This visualization provides an intuitive view of where the model confuses between similar classes and where it performs strongly.



The matrix remains largely diagonal, with a few off-diagonal clusters, indicating some class confusion.

3.4. Training Vs Validation Behaviour



Both graphs seem to be depicting the same scenario. The model fits the training set extremely well, training loss continues to decrease while training accuracy approaches 97–98%, but validation performance levels off much earlier at around 12 epochs, with validation accuracy stabilizing around 66–67% and validation loss plateauing after its initial decline. This growing gap between training and validation metrics reflects overfitting, where additional epochs improve memorization of training data but do not translate into better generalization on unseen samples.

3.5. Summary

The best MLP model, which had two wide hidden layers (4096 and 2048 units), GELU activation, dropout regularization, Adam optimization, and early stopping, achieved the following:

65.95% test accuracy on the 50-class household object classification task.

Thus, macro precision, recall, and F1-score are all around 0.66, indicating balanced yet far-from-perfect performance across classes.

Strong performance for several classes, recall ≥ 0.80, with significantly weaker results for a handful of challenging classes, recall ≈ 0.48–0.52.

The combination of the confusion matrix and learning curves, therefore, suggests that while the model captures a large amount of useful structure in the data, it is still overfitting, and there is still significant confusion between semantically similar classes.

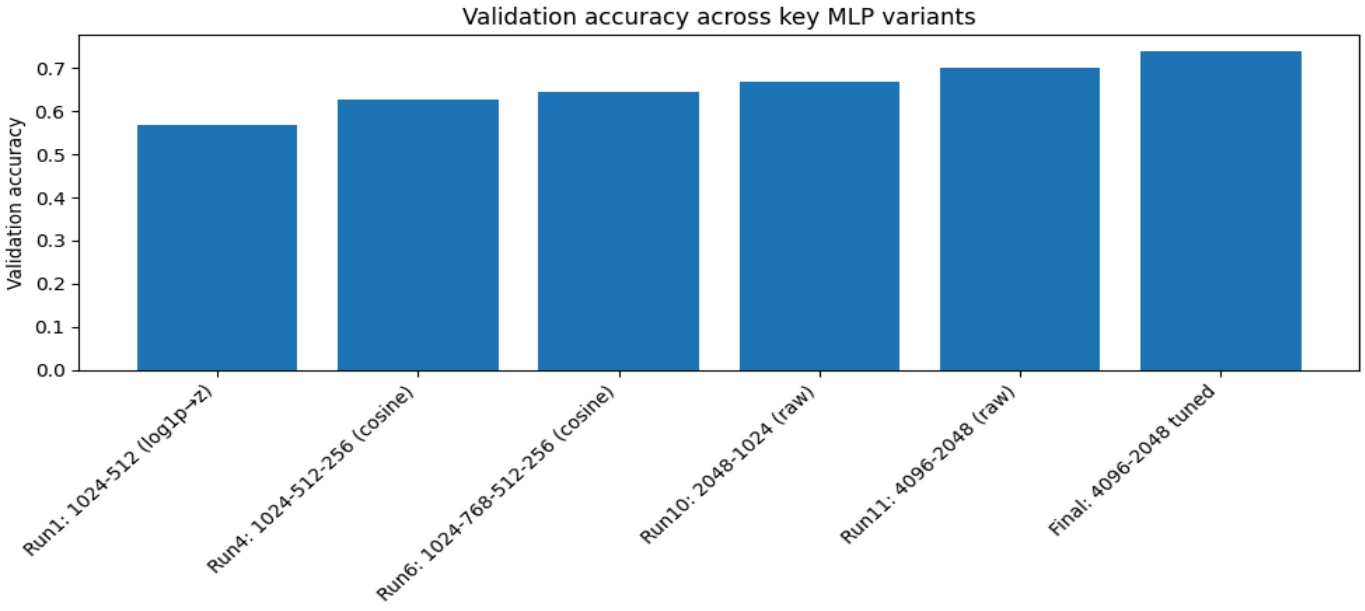
In the context of the project, this model provides a high-capacity, well-optimized baseline. It shows the performance of a shallow-but-wide MLP if it is carefully trained and stopped early, and therefore can serve as an important reference point against which to consider alternative architectures-e.g., deeper MLPs or CNNs-and normalization strategies explored by the rest of the team

4. Analyzing our Earlier Models

This section compares the most informative experimental runs leading up to the final model. Rather than listing all 12 attempts, we focus on six representative runs that best illustrate the design decisions, performance trends, and underlying patterns that guided the progression toward the final architecture.

4.1. Summary Table of Key Experimental Runs

Run	Preprocessing	Hidden Layers	Dropout	Optimizer (LR/WD)	Batch size	Best Validation Accuracy
Run 1	Log1p → z	[1024,512]	0.15	Adam (2e-3 / 1e-4)	2048	56.86%
Run 2	Log1p → z	[1024,512,256]	0.20	Adam (2e-3 / 4e-4)	2048	62.87%
Run 3	Log1p → z	[1024,768,512,256]	0.20	Adam (2e-3 / 4e-4)	2048	64.38%
Run 4	Raw Embeddings	[2048,1024]	0.10	Adam (3e-3 / 1e-4)	1024	66.89%
Run 5	Raw Embeddings	[4096,2048]	0.10	Adam (3e-3 / 3e-5)	1024	70.10%
Run 6	Raw Embeddings	[4096,2048]	0.10	Adam (1e-4 / 3e-5)	528	74.03%



4.2. Effects of Model Width and Depth

The goal of this experimentation was to understand how performance on the 50-class classification task changed with respect to both width and depth in terms of architectural scale. Across all runs, widening the network consistently produced larger and more stable improvements than increasing depth.

Early narrow models (Run 1 and its variants), with hidden sizes of [1024, 512], achieved validation accuracies of about 56-57%. Those runs showed the network was able to learn the pick up the main patterns in the data, but not enough to pick up on the subtle distinctions. Increasing width to [1024, 512, 256] (Run 2) improved performance to 62.9%, and extending the depth further in Run 3 ([1024, 768, 512, 256]) provided another modest improvement to 64.38%. The added parameters had helped the model learn the more subtle decision boundaries.

Deeper networks showed clear signs of overfitting, however. While the model's training accuracy increased rapidly with growing depth, validation accuracy started plateauing sooner and fluctuating more sharply, even within the same run. For deeper configurations, such as Run 3, the gap between training and validation performance became much wider, and the latter's loss became increasingly unstable.

Where increasing the depth struggled to make large, easily reproducible improvements, broadening the architecture was consistently much better. Run 4 expanded the hidden layers in width to [2048, 1024], leaping 66.89%, and the very wide two-layer configuration of [4096, 2048] in Run 5 pushed past the 70% barrier for the first time. Significantly, these shallow-wide models achieved higher accuracy and maintained much more stable validation curves with significantly reduced overfitting.

4.3. Effects of Preprocessing

Another major factor that was explored during experimentation was the effect of input preprocessing. Several early runs adopted a log1p → z-score normalization pipeline intended to stabilize optimization by rescaling feature distributions. Although this can often be very useful for raw pixel or numerical data, the results showed that it was actually counterproductive for this embedding-based dataset.

All the models trained with log1p→z normalization plateaued far below the performance of their raw-embedding counterparts. For instance, Run 1 remained in the 56–57% accuracy range, and deeper or wider models, even Runs 2–3, topped out around 62–64%. These models often achieved higher training accuracy but considerably weaker validation accuracy, which means that normalization did not help them generalize better; in some cases, it exacerbated overfitting.

This behavior probably emerged because the given embeddings already had some meaningful internal structure and normalization from the upstream model that generated them. The application of z-score normalization disrupted the relationships between embedding dimensions, decreasing class separability, and increasing the relative impact of noise in low-variance directions. Since classification in this task depends on very subtle directional differences within the embedding space, collapsing or rescaling these dimensions generated systematically poorer representations for the MLP to learn from.

And indeed, once normalization was removed and the models were trained on raw embeddings, performance saw a stark improvement: Run 4 jumped to 66.89%, the wide shallow architecture in Run 5 reached 70.10%, and the tuned final configuration exceeded 74% validation accuracy, well beyond the best achievable result under any normalized setting. These improvements were accompanied by more stable validation loss curves and reduced overfitting, further confirming that raw embeddings were indeed the way to go.

5. Summary

This project was focused on the development of a high-performance MLP classifier working with a 50-class household object dataset, using only the given dataset and no pre-trained models or transfer learning. Extensive experimentation allowed the team to investigate a wide array of architectural choices, preprocessing strategies, and optimization configurations to understand what most strongly influences accuracy and generalization.

Three findings were consistent across all runs:

- Model width mattered far more than depth.

The shallow-wide architectures performed the best, while deeper networks quickly overfitted and became unstable. By increasing the hidden-layer width, significant gains were obtained in terms of validation accuracy and training stability.

- Raw embeddings outperformed normalized inputs.

Preprocessing pipelines such as log1p → z-score consistently degraded performance. This was because the raw embeddings preserved meaningful structure, allowing the MLP to learn cleaner decision boundaries.

- Choices of regularization and optimization were crucial.

Carefully tuned weight decay, GELU activations, and moderate dropout combined with early stopping formed a robust setup for training Adam. This combination prevented overfitting and supported smooth convergence despite the model's large parameter count.

The final architecture was a 2-layer shallow-wide MLP with GELU and 0.10 dropout (4096, 2048), giving the best performance among all candidates. After retraining on refined hyperparameters, it reached a validation accuracy of ≈74% and ≈66% held-out test accuracy, outperforming all previous approaches. In all, the project shows that controlled experimentation, systematic testing, and careful architectural design are able to create a powerful classifier without necessarily having recourse to complex deep-learning models. The final MLP strikes a balance in capacity, stability, and generalization, thus becoming the most reliable solution developed during this project.

6. References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[2] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[3] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *Proc. 3rd Int. Conf. Learning Representations (ICLR)*, 2015.

[4] D. Hendrycks and K. Gimpel, “Gaussian Error Linear Units (GELUs),” *arXiv preprint arXiv:1606.08415*, 2016.

[5] L. Prechelt, “Early stopping — But when?,” in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 1998, pp. 55–69.

[6] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

7. LLM

8. Was not able to retrieve the prompt and answer after read the technical analysis however below are the sections of code that used an LLM to help create code.
    a. Logging logic was created using references to lab material and assistance from ChatGPT
    b. Submission logic to load the best model from the drive was also created with the help of ChatGPT.