COMP432 Project Report

Jaspreet Singh (40259729), Evan Teboul (40238390), Kevin Courey (40245966)

November 27, 2025

1 Introduction

1.1 Data Analysis

1.1.1 Label Distribution

The label distribution is balanced, with each class having approximately 2308 samples. The figure below shows the label distribution.

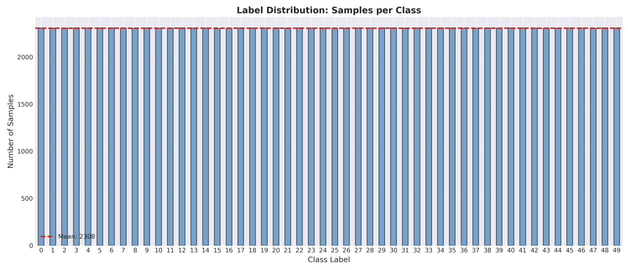

Figure 1: Label Distribution

1.1.2 Feature Distribution

The features are very sparse (meaning $f_i = 0$ for feature i), because across all features mean sparsity is 85.12%. A plot of the feature distribution is shown in Figure 2. The reason for the sparsity is probably because the output of the CNN model was passed through ReLU activation function, which zeroed out most of the values.

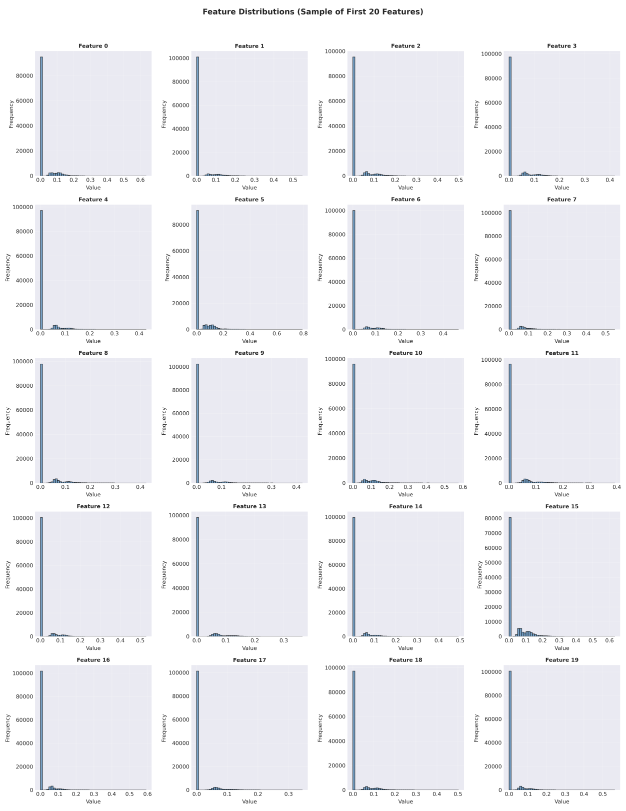

Figure 2: Feature Distribution

1.1.3 Feature Correlation

The features have little correlation with each other, as shown in figure 3 which plots correlation between the first 100 features.

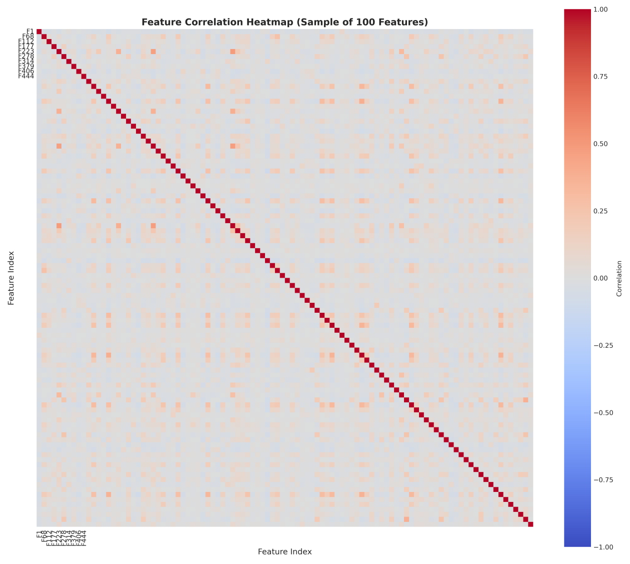

Figure 3: Feature Correlation

This makes sense, as the features were extracted from a CNN model trained on the image dataset and the CNN model is expected to extract features that are not correlated with each other.

2 Model Architecture and Training

2.1 Data Preprocessing

L2 normalization was used to scale the data. StandardScaler and MinMaxScaler were also tried, but they were marginally worse than L2 normalization. The reason for the marginal difference is probably because the data is already very sparse and the scaling doesn't really affect it much.

2.2 Architecture Details

An ensemble of five 2-layer MLP models, each trained on a different fold of the dataset (used stratified k-fold cross-validation from scikit-learn [4]) is built. The outputs logits from each model in the ensemble is averaged and passed through a softmax function to get the final predictions.

Each model takes in the 500 input features and outputs predictions for all the 50 classes. The architecture consists of two hidden layers, each containing 4096 units, resulting in a [500 $->$ 4096 $->$ 4096 $->$ 50] network structure. Each layer block follows the composition: Linear transformation $->$ LayerNorm [6] normalization $->$ GELU [7] activation $->$ Dropout [8] (rate=0.4). ReLU and Leaky ReLU activations, as well as BatchNorm [9] normalization, were employed as well, but GELU combined with LayerNorm provided the best performance. The model architecture is illustrated in Figures 4 and 5. The first figure shows the architecture of the input and first hidden layer block, and the second figure shows the architecture of the second hidden layer block and output layer in the ensemble. The **Gemm** layer is the linear transformation layer. And the **Div**, **Erf**, **Add** and **Mul** is the GELU activation layer but PyTorch has expanded the GELU activation function into these four layers for optimization reasons.

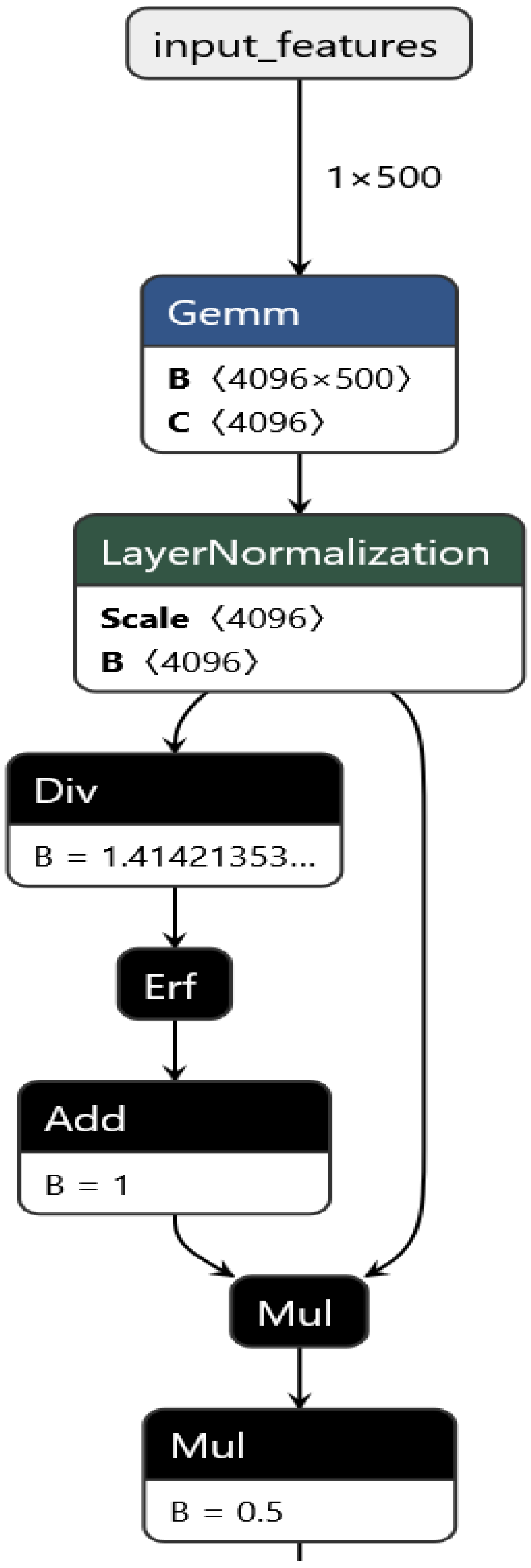

Figure 4: Ensemble MLP Architecture - Part 1

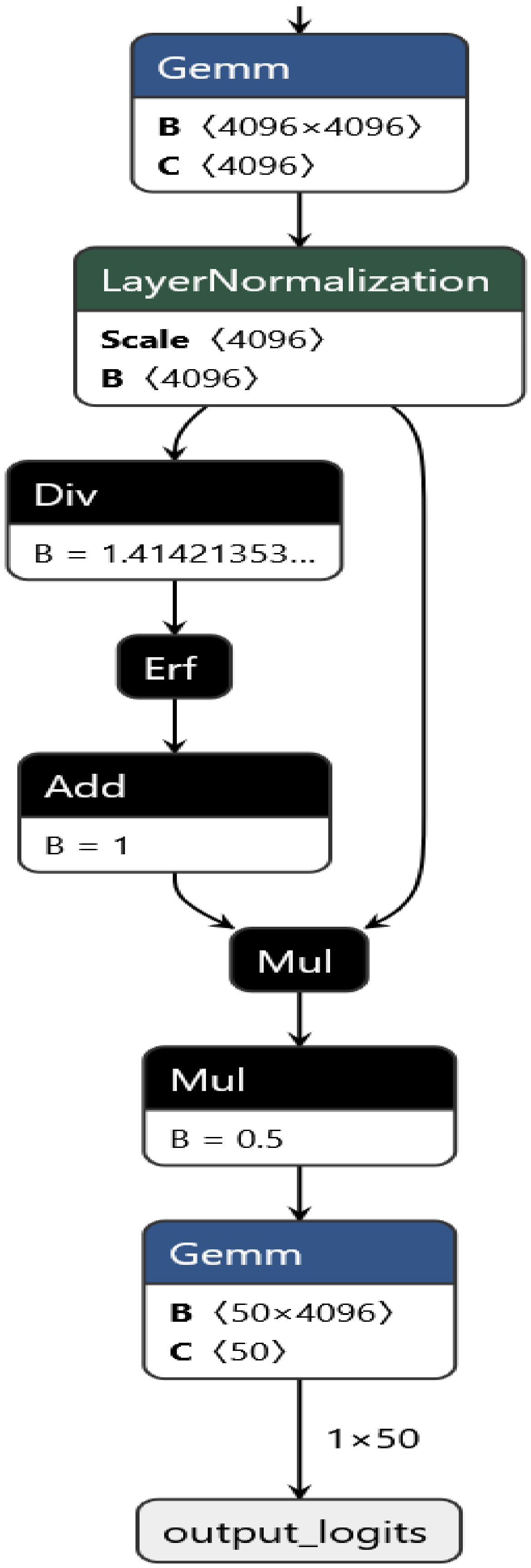

Figure 5: Ensemble MLP Architecture - Part 2

2.3 Training Methodology and Optimization Techniques

Initially single models were trained to establish a baseline and determine optimal training duration. For our initial submissions, a single model was trained using an 80% training and 20% validation split to find the number of epochs required to reach the best validation loss. Then a new model was trained from scratch for that number of epochs, but this time on the full training dataset without a validation split. This approach allowed all available training data to be used for the final model while using the validation split to determine the optimal stopping point.

Subsequently, an ensemble of 5 models, each trained on a different fold of the dataset was built. The basic training loop remained the same for both single and ensemble models. Each individual model in the ensemble is trained separately using the same hyperparameters (except the seed) and training procedure, and the dataset was split into 5 folds, and with each model being trained on 80% of the data (its training fold) with 20% held out for validation. Importantly, each model was trained only on its own training split. This approach ensures that the models are generalizable and do not overfit to the training data, while the ensemble averaging provides robustness through model diversity.

CrossEntropyLoss with label smoothing (0.05) [1] was used to reduce overconfidence and improve generalization. The model was optimized using AdamW [2] with a learning rate and weight decay of 0.0005, and the learning rate was scheduled using CosineAnnealingLR [3] to decay the learning rate smoothly to 0 to prevent overfitting but still train the model for a longer time. Training was performed with a batch size of 512 for up to 75 epochs, with early stopping triggered if the validation loss failed to improve for 10 consecutive epochs. The hyperparameters used for training are summarized in Table 1.

Hyperparameter	**Value**
Number of ensemble models	5
Train/validation split	80% / 20%
Loss function	CrossEntropyLoss
Label smoothing	0.05
Optimizer	AdamW
Learning rate	0.0005
Weight decay	0.0005
Learning rate scheduler	CosineAnnealingLR
Batch size	512
Maximum epochs	75
Early stopping patience	10 epochs
Dropout rate	0.4

Table 1: Training hyperparameters

3 Evaluation

3.1 Metrics

Each model was evaluated using multiple metrics to comprehensively assess performance across all 50 classes:

- **Classification Accuracy**:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{1}$$

  The primary metric measuring the proportion of correctly classified samples across all classes.

- **Precision (Macro-averaged)**:

$$\text{Precision} = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i + FP_i} \tag{2}$$

where C is the number of classes, TP_i is the number of true positives for class i, and FP_i is the number of false positives for class i. Precision measures the proportion of positive predictions that are correct, averaged across all classes without bias toward majority classes.

- **Recall (Macro-averaged)**:

$$\text{Recall} = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i + FN_i} \tag{3}$$

where FN_i is the number of false negatives for class i. Recall measures the proportion of actual positives that are correctly identified, providing insights into the model's ability to detect each class.

- **F1-Score (Macro-averaged)**:

$$\text{F1-Score} = \frac{1}{C}\sum_{i=1}^{C}\frac{2 \cdot \text{Precision}_i \cdot \text{Recall}_i}{\text{Precision}_i + \text{Recall}_i} \tag{4}$$

The harmonic mean of precision and recall, providing a balanced metric that considers both false positives and false negatives. It offers insights into class-specific performance across all 50 classes.

- **Cross-Entropy Loss**: We used CrossEntropyLoss with label smoothing (0.05) to reduce overconfidence and improve generalization. This loss function was monitored during training and validation to track model convergence and detect overfitting.

3.2 Training and Validation Loss

The training and validation loss for the first model in the Ensemble MLP is shown in Figure 6.

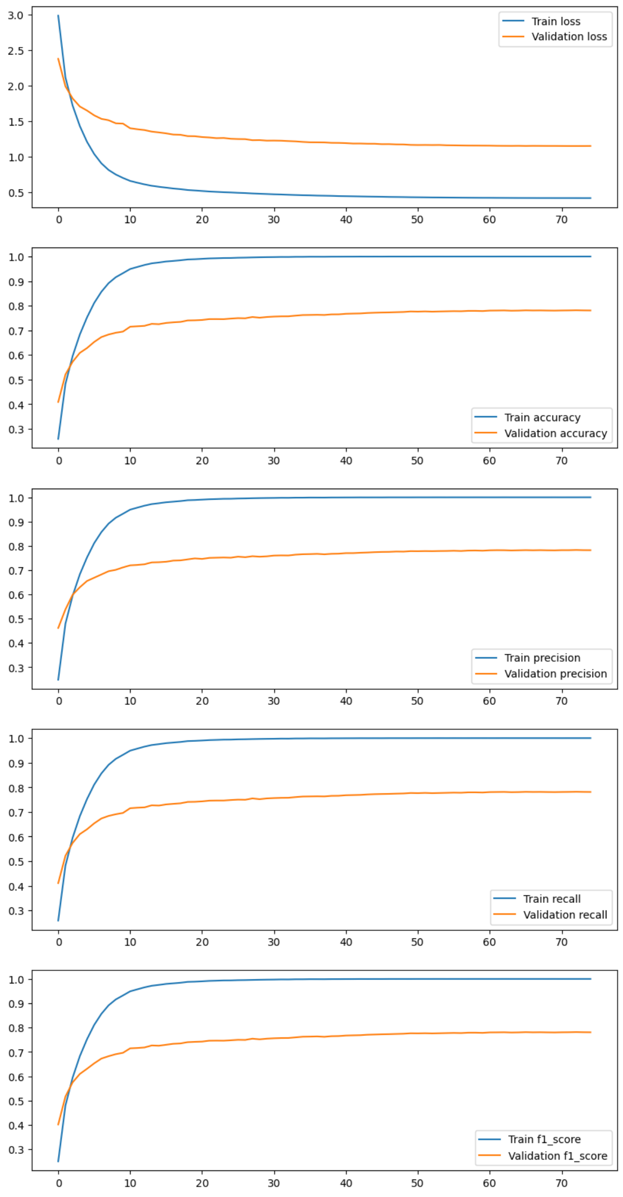

Figure 6: Training and Validation Loss for the first model in the Ensemble MLP

3.3 Model Performance

Each model in the ensemble achieved approximately 78% +- 1% accuracy with similar precision, recall and F1-scores on the validation set. The macro-averaged precision, recall, and F1-score on the validation set were 78.16%, 78.15%, and 78.05% respectively, showing balanced performance across all metrics. This is despite the fact that it overfit on the training data (100% training accuracy) it was still able to maintain a validation accuracy of 78.08%, indicating that our regularization efforts helped it generalize on the unseen data despite the large capacity of the network.

Table 2 summarizes the results of all our Kaggle submissions.

Model	Score
Individual model (25 epochs)	0.788
Individual model (58 epochs)	0.813
Ensemble model (5-folds)	0.827

Table 2: Kaggle submission results

3.4 Performance Analysis and Error Patterns

The model exhibits balanced precision and recall values (both 0.78). This balance is reflected in the F1-score of 0.78, which closely matches both precision and recall, suggesting consistent performance across all classes.

Performance evolution through hyperparameter tuning showed significant improvements: initial deep networks with funnel architecture with ReLU activation achieved only 57-60% validation accuracy with severe overfitting. Transitioning to GELU activation improved performance to approximately 60%, and further improvements to 70% were achieved through layer normalization, increased dropout, and label smoothing. The final architecture with wider layers (4096 units per layer) and optimal regularization achieved 78% validation accuracy, representing a substantial improvement from the initial baseline.

The confusion matrix reveals that there were some contiguous classes (classes with adjacent indices) that were often confused with each other, as shown in Figure 7. E.g. classes 0-4 confuse with each other. Another example is classes 5-9 confuse with each other. This pattern suggests that the original images of the objects looked similar to each other.

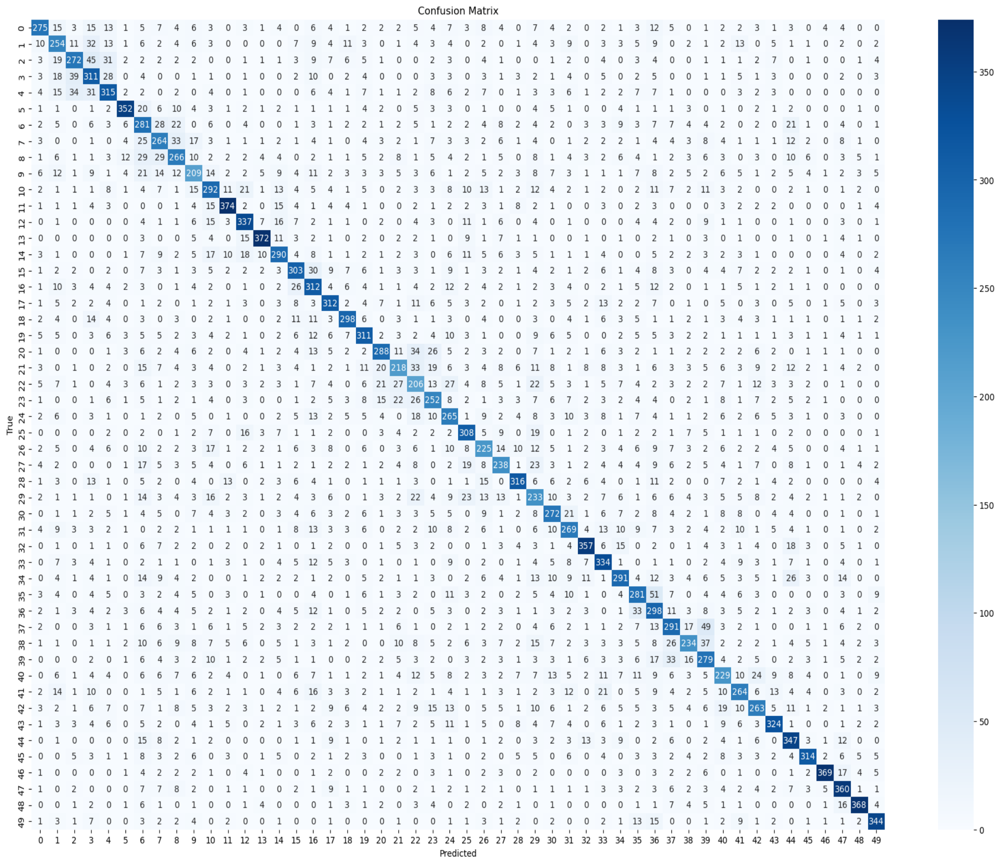

Figure 7: Confusion Matrix Before Label Smoothing - Showing Contiguous Class Confusion

The introduction of label smoothing (0.05) significantly reduced this contiguous class confusion pattern, as shown in Figure 8. We can notice that the confusion matrix is more uniform now. E.g. 45 images were originally wrongly classified as class 4 instead of 3. But after label smoothing, this went down to 23 images.

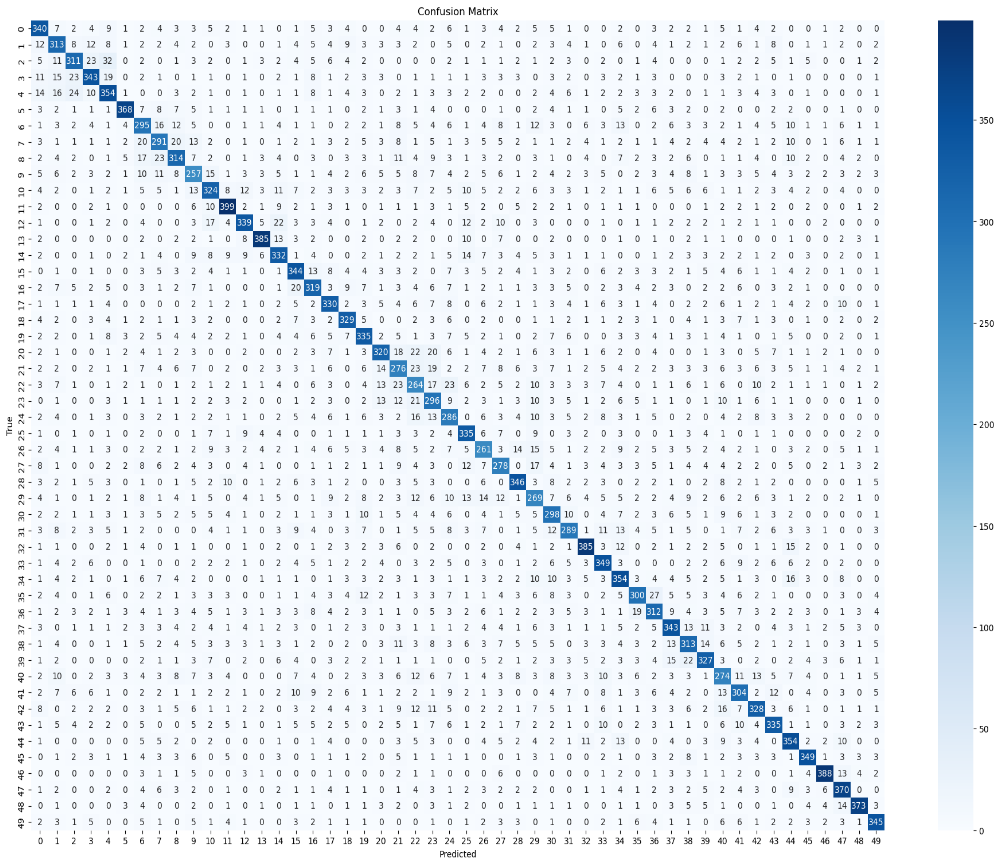

Figure 8: Confusion Matrix After Label Smoothing (0.05) - Reduced Contiguous Class Confusion

3.5 Architecture Experiments

We experimented a lot with various architectures, including different depths, layer widths, activation functions, and regularization techniques. We began with a deep funnel architecture [2048, 1024, 512, 256, 128, 64] using ReLU activation, which achieved high training accuracy (83.7%) but only 57-60% validation accuracy, indicating severe overfitting. The large training-validation gap suggested the deep architecture was memorizing patterns rather than learning generalizable features.

We first explored activation functions, finding that Leaky ReLU provided slight improvement (59%) over ReLU, while GELU demonstrated superior performance. Transitioning to GELU improved validation accuracy to approximately 60%, and further improvements to 70% were achieved through layer normalization, increased dropout (0.2 to 0.4), and label smoothing. The reason why GELU performed better is because it is a smooth activation function and outputs non-zero values for inputs less than 0, which probably helped during the training process as the features were very sparse.

After this we explored using shallower architectures while increasing layer width. We explored wider layers: [3072, 3072, 2048, 1536] achieved 75% validation accuracy, [4096, 2048] achieved 77.6%, and [4096, 4096] achieved the best performance at 78.08% validation accuracy. This means that increasing the layer width is more effective than increasing the depth for this task. We also tried adding one extra layer to the architecture but it didn't improve the performance at all and increased the training time significantly.

We also tuned regularization hyperparameters: label smoothing was optimized from 0.0 to 0.1, and finally to 0.05 for optimal performance; weight decay was increased from 0 to 1e-4, and finally to 5e-4; and normalization was changed from BatchNorm to LayerNorm for slightly better performance.

4 Summary and Conclusions

Our best performing model is a 2-layer MLP ensemble with architecture [4096, 4096] using GELU activation, LayerNorm normalization, and dropout rate of 0.4. The ensemble consists of 5 models trained on different folds, with predictions averaged for final classification.

We found that shallower, wider networks performed significantly better than deeper networks for this task. The optimal 2-layer architecture [4096, 4096] achieved 78% validation accuracy, compared to 57-60% for deeper funnel architectures. GELU activation proved superior to ReLU and Leaky ReLU, and label smoothing (0.05) effectively reduced contiguous class confusion patterns.

The final ensemble model achieved approximately 78% validation accuracy with balanced precision, recall, and F1-score (all around 0.78). On the Kaggle competition test set, the ensemble achieved a score of 0.827, which was our best result.

5 References

References

[1] PyTorch, "CrossEntropyLoss," *PyTorch Documentation.* [Online]. Available: `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`

[2] PyTorch, "AdamW," *PyTorch Documentation.* [Online]. Available: `https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html`

[3] PyTorch, "CosineAnnealingLR," *PyTorch Documentation.* [Online]. Available: `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CosineAnnealingLR.html`

[4] Scikit-learn, "Stratified K-Fold," *Scikit-learn Documentation.* [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html`

[5] PyTorch, "Neural Network Layers," *PyTorch Documentation.* [Online]. Available: `https://pytorch.org/docs/stable/nn.html`

[6] PyTorch, "LayerNorm," *PyTorch Documentation.* [Online]. Available: `https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html`

[7] PyTorch, "GELU," *PyTorch Documentation.* [Online]. Available: `https://pytorch.org/docs/stable/generated/torch.nn.GELU.html`

[8] PyTorch, "Dropout," *PyTorch Documentation.* [Online]. Available: `https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html`

[9] PyTorch, "BatchNorm1d," *PyTorch Documentation.* [Online]. Available: `https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html`

We used ChatGPT to help us with generating comments for our code and refining the report.

A ChatGPT Prompts

- **Generating comments for our code**:

```
Please generate comments for the following code:
```

- **Refining the report**: This generally involved making the report more concise and clear and helping with grammar and punctuation.