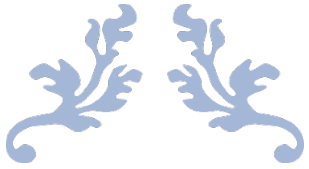


THE OPEN SET RECOGNITION PROBLEM

Project Summary

- YOAV SCHWAMMENTHAL

Contents

1. Method Description and Rationale

1.1. Core Approach

My approach leverages the observation that in a well-trained neural network, samples from the same class tend to cluster together in the embedding space. I developed a hybrid method combining a convolutional neural network (CNN) with clustering-based outlier detection to identify unknown classes.

1.2. Data Augmentation Strategy

My method begins with a crucial data preparation step: augmenting the MNIST training dataset. For each original training image, I created two additional augmented versions using a combination of:

- Gaussian blur
- Smart rotation (limited rotation for digits 6 and 9 to prevent ambiguity)
- Random affine transformations (translation and scaling)

This augmentation strategy serves multiple purposes:

1. **Improved Generalization**: By exposing the model to various transformations of digits, it learns to recognize them under different conditions
2. **Robust Embedding Space**: The augmented samples help create more comprehensive clusters in the embedding space, leading to better boundaries between known and unknown samples
3. **Out-of-Distribution Detection**: By teaching the model that certain transformations of digits are still "known", it becomes better at identifying truly unknown samples

1.3. Architecture Overview

My architecture was carefully designed to balance feature extraction, embedding quality, and computational efficiency:

1. **Convolutional Backbone**:
    - Two convolutional blocks with increasing channels (32→64→128)
    - Rationale: Progressive feature extraction captures both fine details (crucial for digit discrimination) and higher-level patterns (important for unknown detection)
    - Double convolution per block allows learning more complex features while keeping the network depth manageable

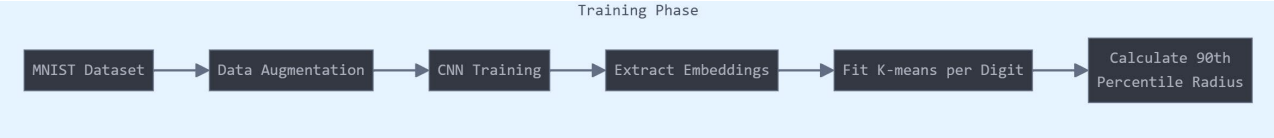
2. **Regularization Strategy**:
    - Batch normalization after each convolution
    - Different dropout rates: 0.25 for conv layers, 0.5 for dense layers
    - Rationale: This dual-rate approach prevents overfitting while maintaining spatial feature coherence in convolutional layers
3. **Embedding Space Design** (128-dimensional):
    - Dimensionality chosen to be:
        - Large enough to capture digit variations
        - Small enough to form meaningful clusters
        - Computationally efficient for distance calculations
4. **Output Layer**:
    - 11 outputs (0-9 + unknown)
    - Softmax
    - During inference, logits are modified based on cluster distances

The model architecture gradually reduces image size while increasing feature complexity, leading to a final compact and well-separated feature space. This final embedding space allows the model to accurately classify known digits while distinguishing unknown samples using clustering.

1.4. Training Strategy

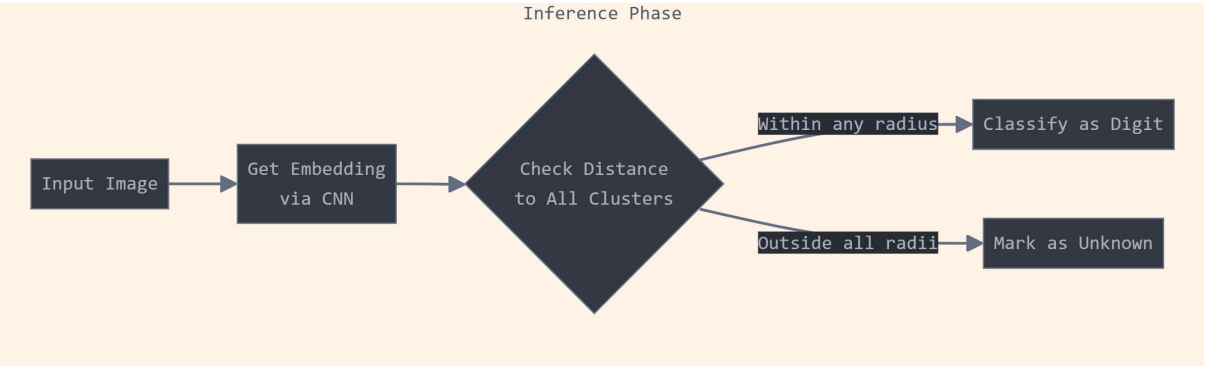
The model is trained in two phases:

- **Classification Training**: The network is trained on MNIST digits using cross-entropy loss
- **Cluster Fitting**: After training, I fit K-means clusters (k=1) to each digit's embeddings. For each cluster, I determine its radius using the 90th percentile of training sample distances. This means that if we arrange all distances from training samples of a particular digit to their cluster center in ascending order, we set the radius to the value that is greater than 90% of these distances. For example, if we have 1000 training samples of digit '0', and we arrange their distances to the cluster center from smallest to largest, we would set the radius to the 900th distance value. This approach ensures that the cluster encompasses 90% of the training samples for that digit, while treating the 10% most distant samples as potential outliers.
- **Visualization of the Training Phase**



1.5. Inference Process

- **During Inference:**
  1. Input image is mapped to the embedding space
  2. Distances to all digit clusters are computed
  3. If the embedding falls within any cluster's radius, it's classified normally
  4. If it falls outside all clusters, it's marked as unknown
- **Visualization of the Inference Phase**



2. Hyperparameter Selection and Configuration

2.1. Model Architecture:

- Embedding dimension: 128
- Dropout rates: 0.5 (dense), 0.25 (conv)
- Two convolutional blocks with increasing channels (32→64→128)

2.2. Training Parameters:

- Learning rate: 0.001
- Weight decay: 0.01
- Batch size: 128
- Epochs: 10

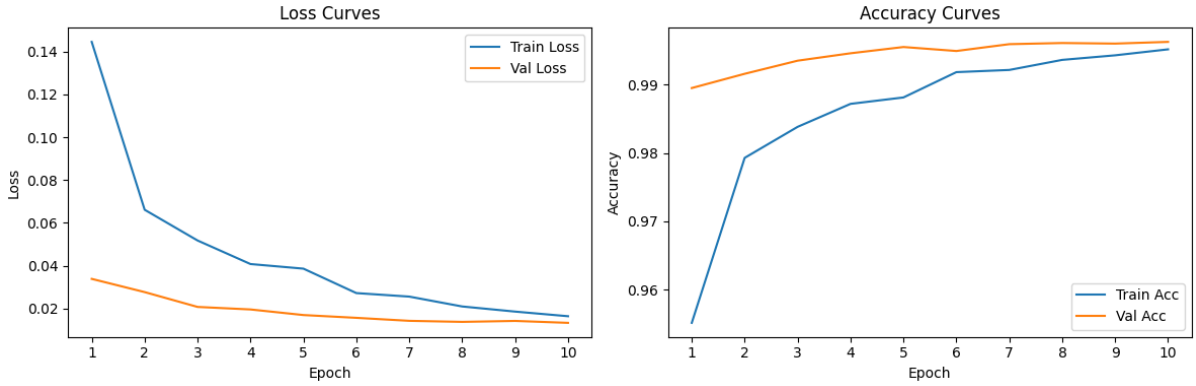
2.3. Cluster Configuration:

- Percentile threshold: 90th

2.4. Selection Methodology and Analysis

The hyperparameter selection process began with established baseline values from literature. I first validated these choices by analyzing training dynamics to ensure proper model behavior:

2.4.1. Generalization Analysis



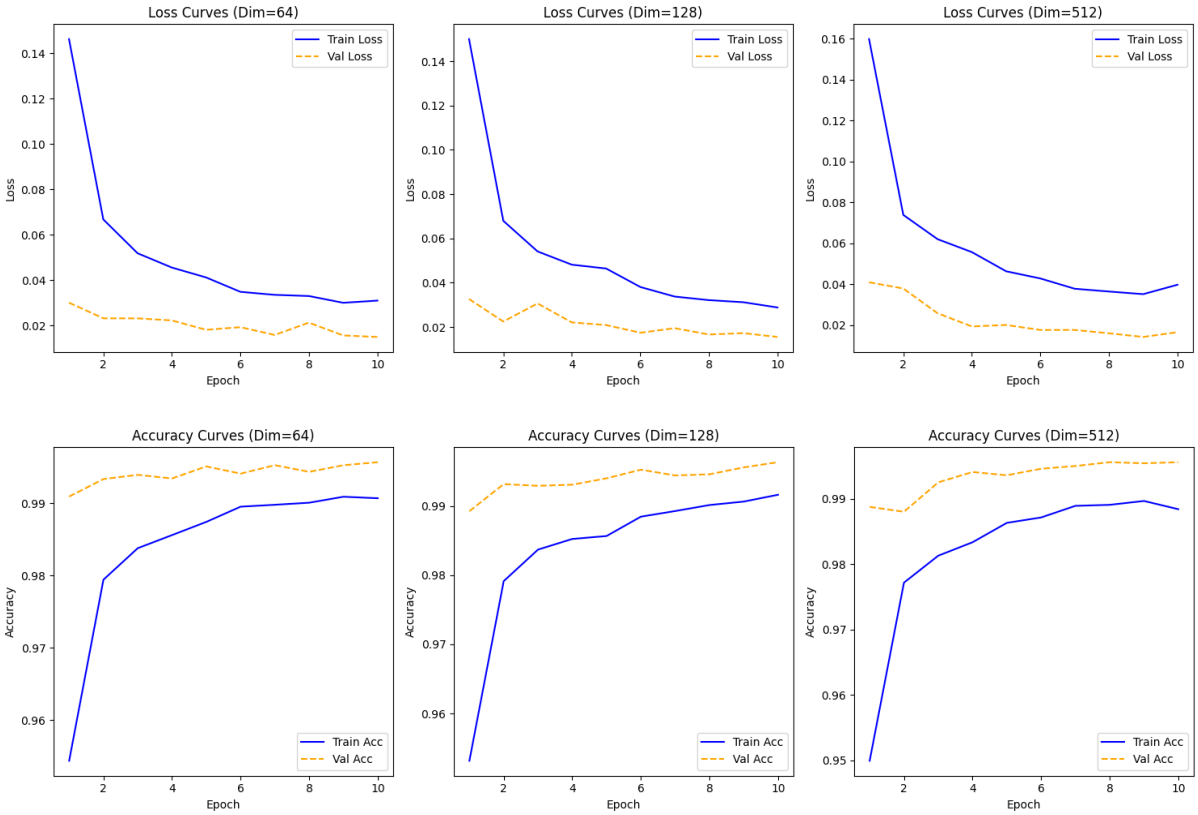
The training curves demonstrate:

1. Consistent convergence without overfitting (validation loss decreases alongside training loss)
2. Stable validation accuracy tracking training accuracy
3. Efficient learning dynamics reaching convergence within 10 epochs

2.4.2. Critical Hyperparameters

Three hyperparameters proved particularly crucial due to their significant impact on model performance and their direct relationship to my core embedding-based approach:

1. *Number of Epochs (10):*
   - Chosen based on convergence analysis and computational constraints
   - Training curves show stabilization around epoch 9
   - Maintains notebook runtime under 10 minutes while achieving optimal performance
2. *Embedding Dimension (128):*

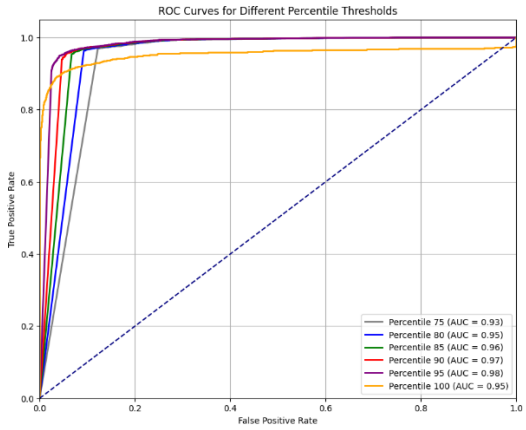


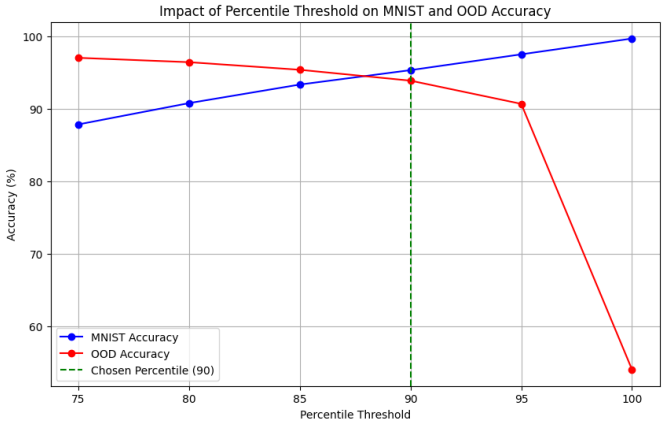
1. **Validation Accuracy:**
   - All dimensions achieve similar validation accuracy (~99%), meaning increasing the embedding size beyond 128 does not significantly improve generalization.
2. **Training Stability:**
   - **Dim=64** shows slightly lower training accuracy, indicating that the model may not capture enough information.
   - **Dim=512** does not provide additional accuracy benefits but shows signs of unstable training dynamics, as decrease is starting at epoch 9 (potential learning rate issues or overparameterization).
3. **Hyperparameter Tuning Consideration:**

- For **Dim=512**, I could try adjusting other hyperparameters, such as choosing a better learning rate, to stabilize training. However, this might introduce additional challenges and require extensive tuning.
- Since **Dim=128** already shows great results, we prefer to stick with it rather than risk new issues from additional hyperparameter tuning.

4. **Final Decision:**
    - I choose **Dim=128** as it provides the best balance between accuracy, stability, and computational efficiency, without the need for further tuning.

3. Percentile Threshold (90th)





1. **High AUC in ROC Curve**
    - The AUC for percentile 90 is 0.97, meaning it has strong classification ability while avoiding unnecessary false positives.
    - While 95% (AUC = 0.98) is slightly higher, it does not provide a significant improvement over 90%.
2. **Balanced MNIST and OOD Accuracy**
    - At 90%, both MNIST and OOD accuracy are well-balanced.
    - Increasing the percentile to 95% or 100% sharply decreases OOD accuracy, meaning the model struggles to reject out-of-distribution samples.
    - Lower percentiles (e.g., 75% or 80%) reduce MNIST accuracy, making the model too uncertain.
3. **Avoiding Overconfidence**
    - At 95% and 100%, the model becomes overconfident, leading to poor generalization on OOD data.
    - 90% maintains strong performance on both MNIST and OOD cases without excessive confidence.
4. **Final Decision**
    - I choose percentile 90% as it provides the best trade-off between classification accuracy, robustness to OOD samples, and stability, making it the optimal choice.

2.4.3. Dataset Strategy for Hyperparameter Selection

To select the optimal percentile threshold, I used:

1. **MNIST Validation Set (Unseen during training and Validation process)**
    - Ensures the model correctly classifies known digits.
2. **OOD Validation Set (CIFAR-10 / FashionMNIST)**
    - Used only for evaluating OOD detection—not for training.
    - Helps balance MNIST accuracy and OOD rejection performance.
    - This OOD validation set is separate from the final OOD test set, ensuring that the final evaluation remains unbiased.

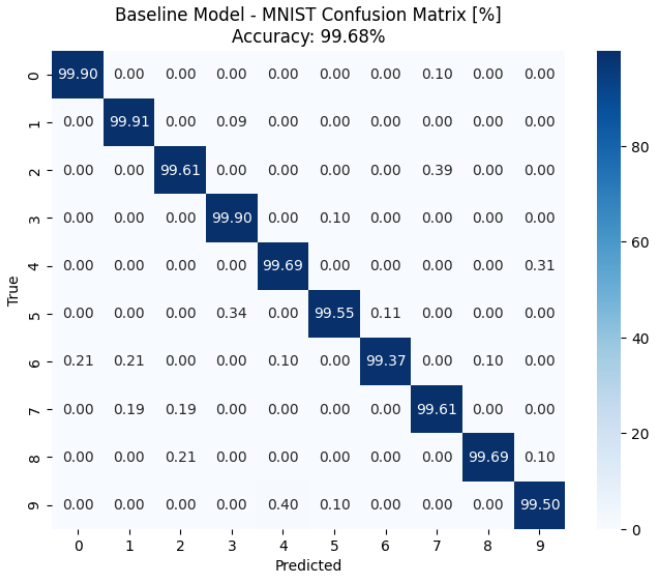
Both datasets were completely separate from the training (and validation) and test sets, ensuring an unbiased threshold selection.

2.4.4. Consideration for Open-Set Recognition

My model is designed for open-set recognition, meaning it should generalize to unknown classes beyond the training data. Since I evaluate OOD detection primarily on CIFAR-10 and FashionMNIST, it's important to be mindful that extensive use of these datasets could make the model more tailored to them rather than truly generalizable, namely, overfitting over the CIFAR-10 and FashionMNIST. To avoid this, I used them in a limited and controlled manner, ensuring they serve only as benchmarks for evaluating threshold selection and overall performance.

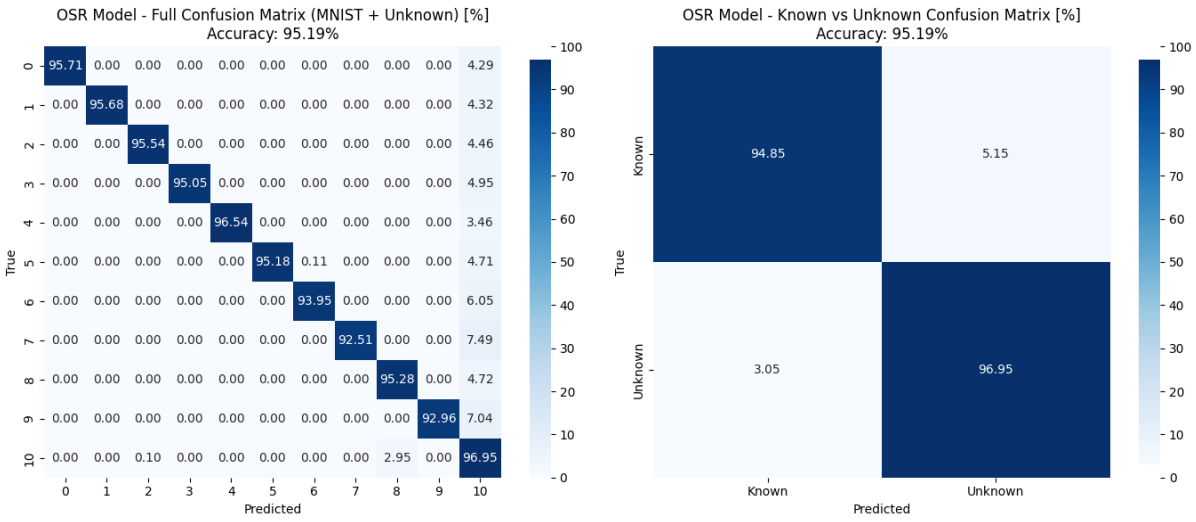
3. Performance Analysis

3.1. Baseline Model Performance



The baseline model achieves 99.68% accuracy on MNIST test set, demonstrating excellent digit classification capability.

3.2. OSR Model Performance



The OSR model achieves:

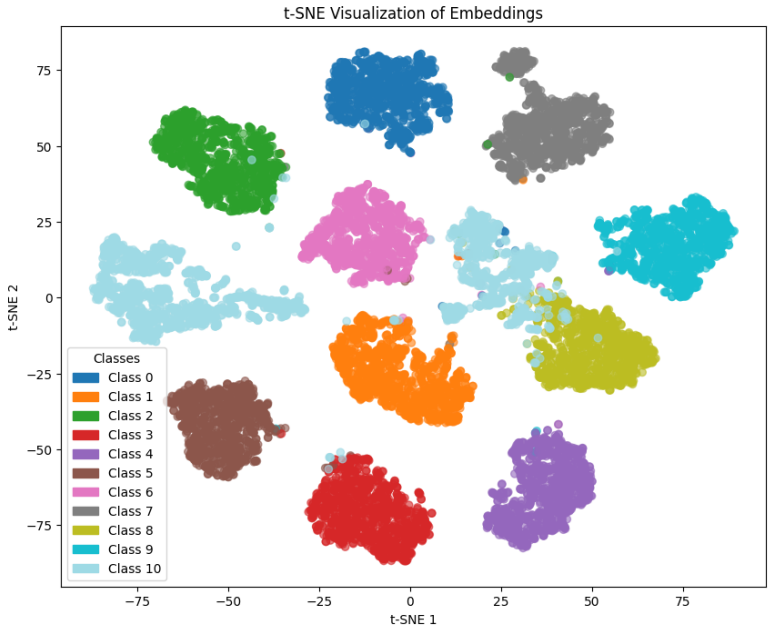
- 94.85% accuracy on MNIST digits
- 96.95% accuracy on unknown samples
- 95.19% overall accuracy

The confusion matrices reveal:

- Strong separation between known and unknown classes (Binary CM)

- Minimal confusion between digits (Full CM)
- We see that most unknown samples are correctly classified as "unknown". However, when unknown samples are misclassified, there is a noticeable tendency for them to be predicted as class '8' more often than other digits. This is an outlier compared to the rest of the known classes, where misclassification rates remain close to zero.

3.3. Embedding Space Analysis



The t-SNE visualization shows:

- Clear separation between digit clusters
- Tight clustering within each digit class
- Unknown samples (Class 10) forming distinct clusters
- Unknown samples cluster (Class 10) slightly overlaps with class 8 cluster, explaining the observation in the confusion matrix where some unknown samples are predicted as 8. This happens because the model checks whether an input falls into a digit cluster, and some unknowns are closer to class 8.

4. Limitations and Edge Cases

My approach, while effective, exhibits several key limitations and specific scenarios where performance may degrade:

4.1 Structural Limitations

1. **Fixed Cluster Radius**
    - Using a single percentile threshold (90th) for all digit clusters assumes uniform distribution of samples

- May not optimally capture digits with naturally wider variation in appearance
- Could lead to overly strict or lenient boundaries for certain digits

2. **Embedding Space Constraints**
    - The t-SNE visualization reveals overlap between unknown samples and digit '8' cluster
    - This structural limitation explains the higher misclassification rate of unknown samples as '8' seen in the confusion matrix
    - Suggests the embedding space doesn't fully separate complex digit structures from certain unknown patterns

4.2 Performance on Different Data Types

My approach should perform well on:

1. **Structured Digit-like Data**
    - Clean, well-centered handwritten digits
    - Digits with moderate noise or distortion (due to the augmentation strategy)
2. **Clearly Different Data**
    - Natural images very different from digits (as seen with CIFAR10)
    - Patterns with distinctly different structure from handwritten digits
    - High-frequency textures or complex patterns

My approach may perform poorly on:

1. **Ambiguous Cases**
    - Poorly written or heavily distorted digits
    - Characters or symbols that share structural similarities with digits
    - Partial or incomplete digit patterns
2. **Edge Cases Identified**
    - Complex patterns that share features with digit '8' (as evidenced by confusion matrix)
    - Samples that fall near cluster boundaries in the embedding space

4.3 Architectural Limitations

1. **Fixed Embedding Dimension**

- The 128-dimensional embedding space may be:
    - Too restrictive for capturing very complex unknown patterns
    - Potentially redundant for simpler patterns

2. **Computational Considerations**
    - Distance calculations to all cluster centers required for each inference
    - Memory requirements scale with the number of known classes

4.4 Dataset Dependencies

1. **Training Data Bias**
    - Model's understanding of "unknown" is implicitly shaped by MNIST characteristics

2. **Validation Limitations**
    - Current validation uses CIFAR10/FashionMNIST as unknown samples
    - Real-world unknown samples might have different characteristics
    - Performance metrics might not fully reflect real-world scenarios

5. Alternative Approach: Enhancing Embedding Space with Triplet Loss

5.1 Motivation

My original embedding-based approach relied on the natural clustering of same-class samples in the embedding space. I hypothesized that I could improve this clustering behavior by explicitly enforcing stricter geometric constraints on the embedding space using triplet loss.

Triplet loss works by:

1. Taking three samples: anchor, positive (same class), and negative (different class)
2. Pushing the positive sample closer to the anchor while pulling the negative sample away
3. Maintaining a minimum margin between positive and negative distances

This approach seemed promising because:

- It could create more distinct boundaries between digit clusters
- It might make unknown detection more reliable by creating larger gaps between clusters
- It could potentially reduce the overlap between digit clusters that caused misclassifications

5.2 Implementation Details

My enhanced model incorporated several key modifications:

1. **Hard Negative Mining**
    - Instead of random triplets, I specifically chose the "hardest" negative samples
    - For each anchor, I selected the negative sample closest to it in the embedding space
    - This focused the training on the most challenging cases
2. **Larger Margin**
    - Used an increased triplet margin of 2.0 (standard is usually 1.0)
    - Aimed to create larger gaps between different digit clusters
    - Should theoretically make unknown detection more robust

3. **Combined Loss Function**

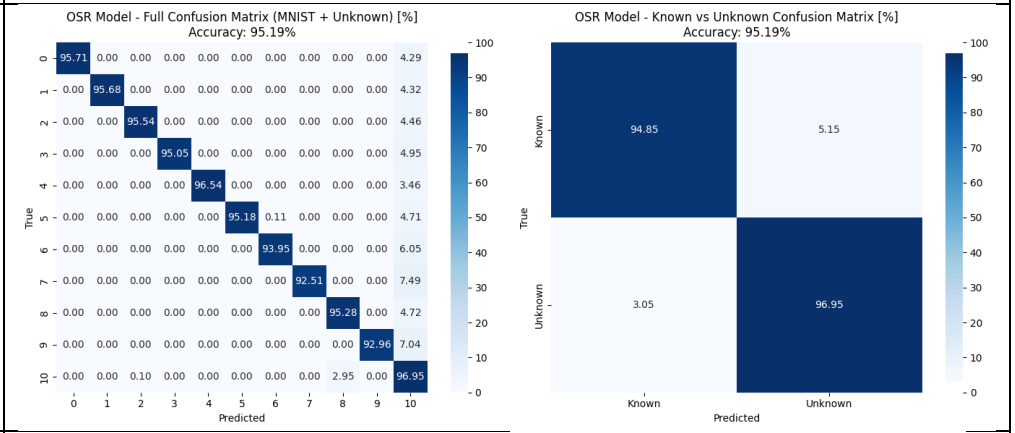
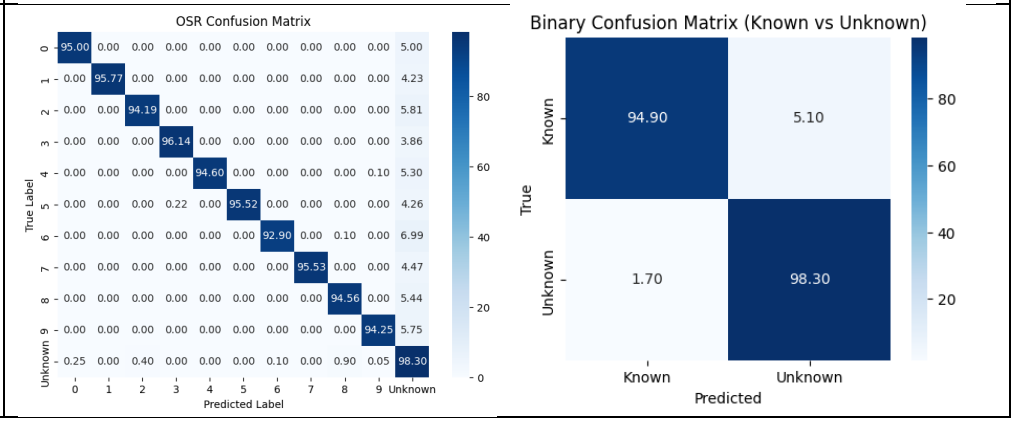
- ```
  loss = ce_weight * cross_entropy_loss + triplet_weight * triplet_loss
  ```
- Balanced classification accuracy with embedding space structure
- Used both global (cross-entropy) and local (triplet) optimization signals

5.3 Why It Wasn't Chosen

Despite the theoretical advantages, this approach was not selected for several reasons:

1. **Some plots:**

	Training time for 10 epochs
Regular	Training (Epoch 10/10): 100% 11250/11250 [07:38<00:00, 25.00it/s]
Triplet-loss	Training Epochs: 100% 10/10 [12:42<00:00, 76.05s/it]

	Confusion Matrices
Regular	
Triplet-loss	

2. **Computational Overhead**

- Hard negative mining required additional distance calculations
  ```

- Training time increased significantly due to triplet selection. For 10 epochs, the regular approach took 7 minutes and 38 seconds, while the triplet-loss approach took 12 minutes and 42 seconds, as shown in the plots.
- Memory requirements were higher due to maintaining triplet batches

3. **Limited Improvement**
    - The accuracy gains were marginal compared to my simpler approach, as seen in the confusion matrices
    - The additional complexity didn't justify the small performance improvement
    - The base model already achieved good cluster separation
4. **Training Instability**
    - More hyperparameters to tune (margin, loss weights)
    - Occasional training convergence issues
5. **Practical Considerations**
    - The simpler approach was more maintainable and interpretable
    - The base model met the project requirements without the added complexity
    - The time constraints of "around 10 minutes max for the entire notebook to run" made this approach less feasible

5.4 Learnings

This exploration provided valuable insights:

1. Sometimes simpler approaches are more robust and practical
2. The trade-off between performance and complexity needs careful consideration
3. The base embedding-space clustering was already effective for the OSR task

This experience reinforced the importance of maintaining a balance between theoretical sophistication and practical implementation constraints.