Model Design and Rationale

Our implementation is based on the Classification-Reconstruction learning for Open-Set Recognition (CROSR) approach presented by Yoshihashi et al. We chose this method after carefully reviewing the Recent Advances in Open Set Recognition Survey recommended in the project instructions.

Classification-Reconstruction learning for Open-Set Recognition is an innovative approach to the open-set recognition problem that combines supervised classification with unsupervised reconstruction learning. The core innovation lies in how it addresses a fundamental limitation of previous deep open-set classifiers: their reliance on feature representations trained solely through supervised learning on known classes, which makes them overly specialized and less effective at identifying unknown classes.

The CROSR architecture consists of two primary components:

1. **Deep Hierarchical Reconstruction Nets (DHRNets)**: These networks are designed to simultaneously:
   - Classify inputs among known classes (supervised task)
   - Reconstruct the input data (unsupervised task)
2. **Open-Set Classifier**: This component combines:
   - A traditional closed-set classifier for known class prediction
   - An unknown detector that leverages both prediction and latent representations

The key technical innovation is the **bottlenecked lateral connections** in DHRNets. These connections extract latent representations (z) from multiple levels of the network's classification pathway and feed them to a reconstruction pathway. This creates a hierarchical set of compact latent representations that capture different aspects of the input data.

In mathematical terms, CROSR extends the traditional classifier equation:

- Traditional approach: $y = f(x)$, where y is the classification output
- CROSR approach: $(y, z) = f(x)$ and $\tilde{x} = g(z)$, where z is the latent representation and $\tilde{x}$ is the reconstruction

For unknown detection, CROSR uses a hyperspherical model based on both y and z: $d(x, C_i) = |[y, z] - \mu_i|_2$, where μ_i is the mean of [y, z] for class Ci.

Some figures from the paper (for visualization):

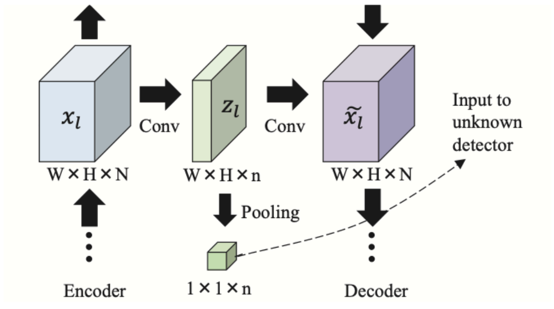


Figure 3. Implementation of the deep hierarchical reconstruction net with convolutional layers.

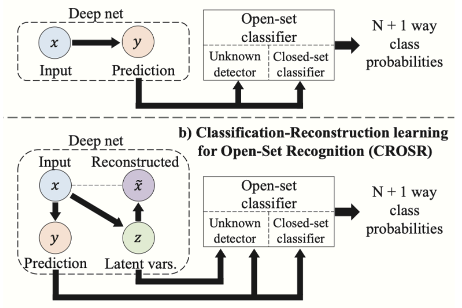

a) Existing deep open-set classifiers (Openmax, G-Openmax, DOC)
Deep net
x
y
Input
Prediction
Open-set classifier
Unknown detector
Closed-set classifier
N + 1 way class probabilities
b) Classification-Reconstruction learning for Open-Set Recognition (CROSR)
Deep net
Input
Reconstructed
x
x̃
y
z
Prediction
Latent vars.
Open-set classifier
Unknown detector
Closed-set classifier
N + 1 way class probabilities

Figure 1. Overview of existing and our deep open-set classification models. Existing models (a) utilize only their network's final prediction y for classification and unknown detection. In contrast, in CROSR (b), a deep net is trained to provide a prediction y and a latent representation for reconstruction z within known classes. An open-set classifier (right), which consists of an unknown detector and a closed-set classifier, exploits y for closed-set classification, and y and z for unknown detection.

We experimented with multiple different architectures, all based on the original CROSR paper.

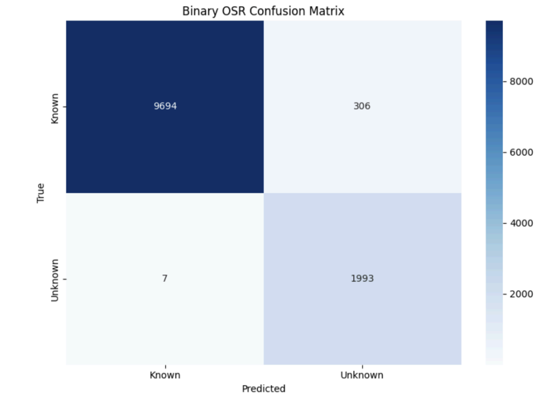
The original implementation included:

a. **Base Network Structure:**
   i. Five convolutional layers with 3×3 kernels
   ii. 100 output channels for each convolutional layer
   iii. ReLU non-linearities after each convolutional layer
   iv. Max pooling layers with a stride of 2 inserted after every two convolutional layers
   v. Two fully connected layers at the end with 500 and 10 units
b. **DHRNet Specific Components:**
   i. Lateral connections placed after every pooling layer
   ii. The dimensionality of all latent representations (z_1) fixed to 32
   iii. Global max pooling used to reduce spatial dimensions of latent representations

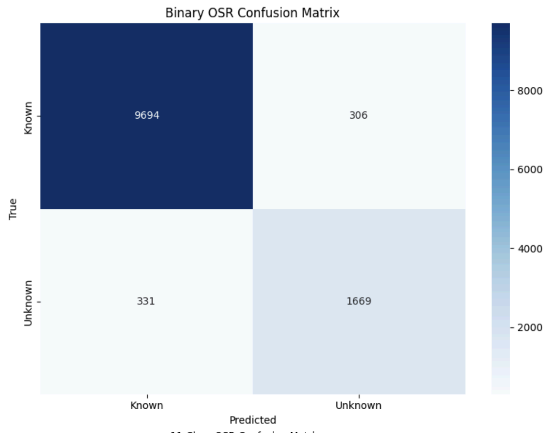
We ultimately followed the core architectural principles described in the paper, but we made changes, since the original implementation took a long time to train and did not comply with the time constraints of 10 minutes. In addition, the original implementation uses libmr library to compute the parameters in Weibull distribution; however libmr is not a library available on Colab so we tried implementing it ourselves. The results we got from this were not very good and thus we implemented a simpler statistical threshold that uses the mean and standard deviation.

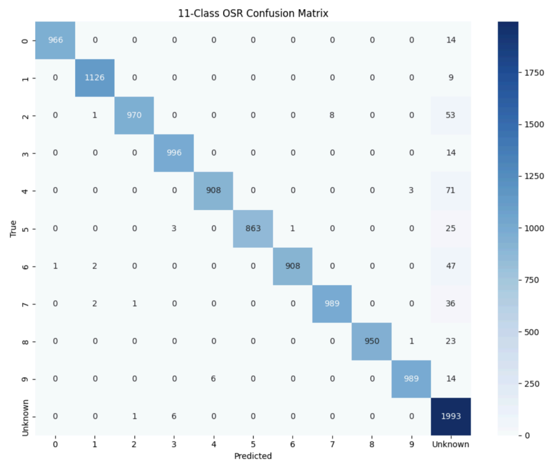
Here are the some of the results when initially implementing the paper:

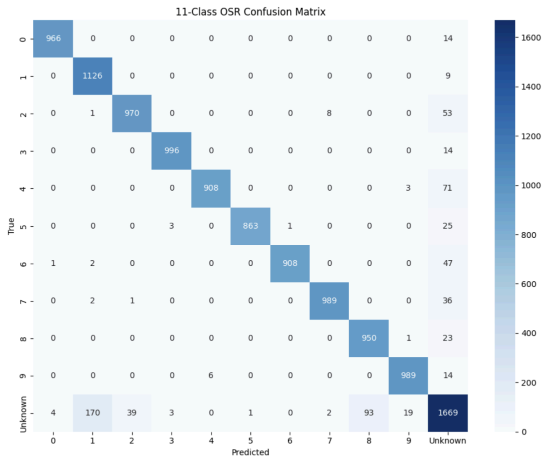
```
Results for CIFAR10 as OOD:
Feature dimensionality: 170
```

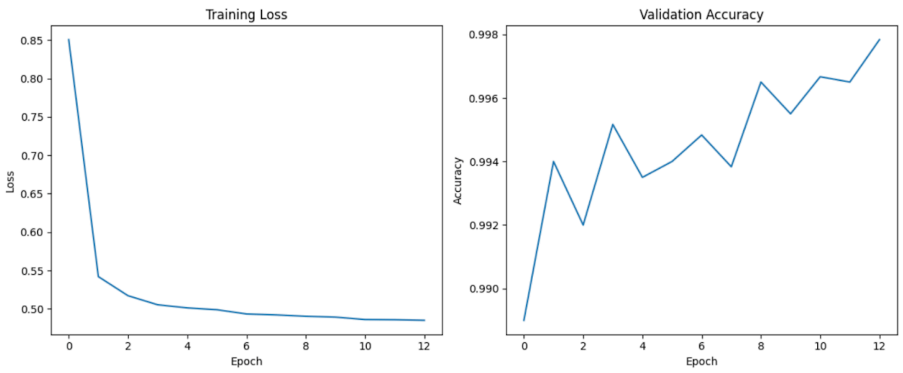


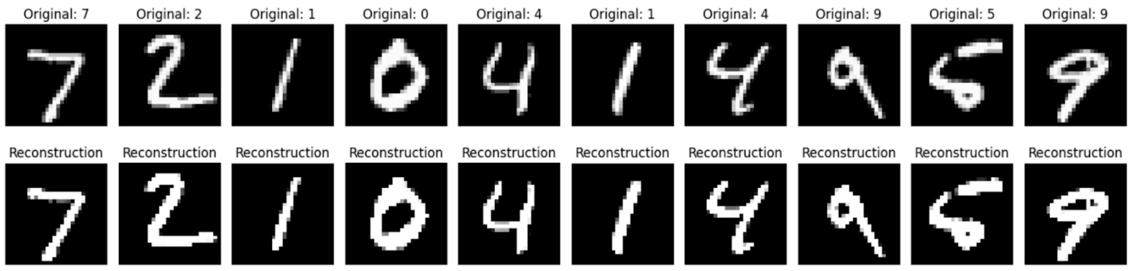
```
Results for FashionMNIST as OOD:
Feature dimensionality: 170
```

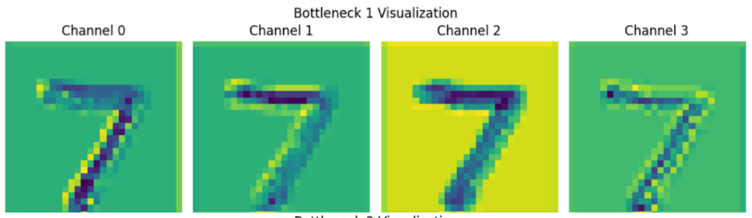


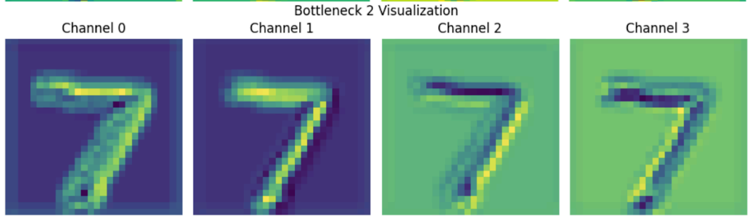


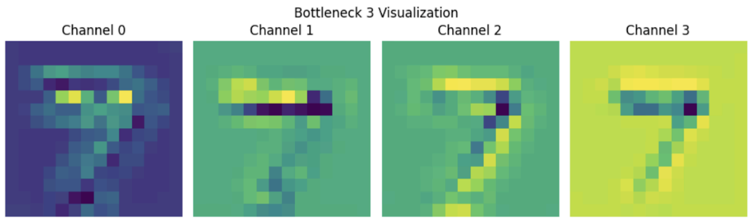


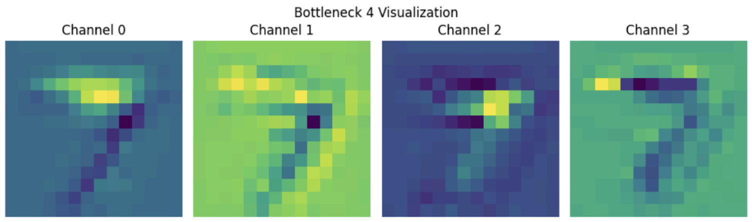


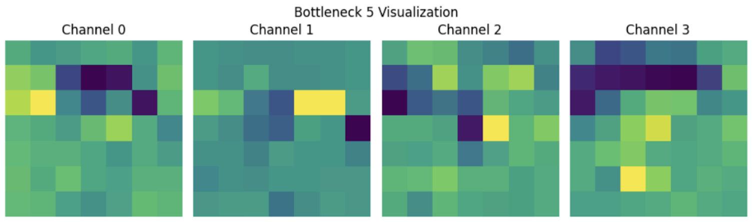












```
Setting statistical threshold: 14.1970 (adjusted for 160-dim space)
```

Hyper-parameters and Specific Design Choices

Note, the results above were a decent starting point and we probably could have continued to make minor adjustments, but we only ran 13 epochs and this took us 21 minutes.

Thus, we experimented with architecture in the following way:

Intermediate Architecture:

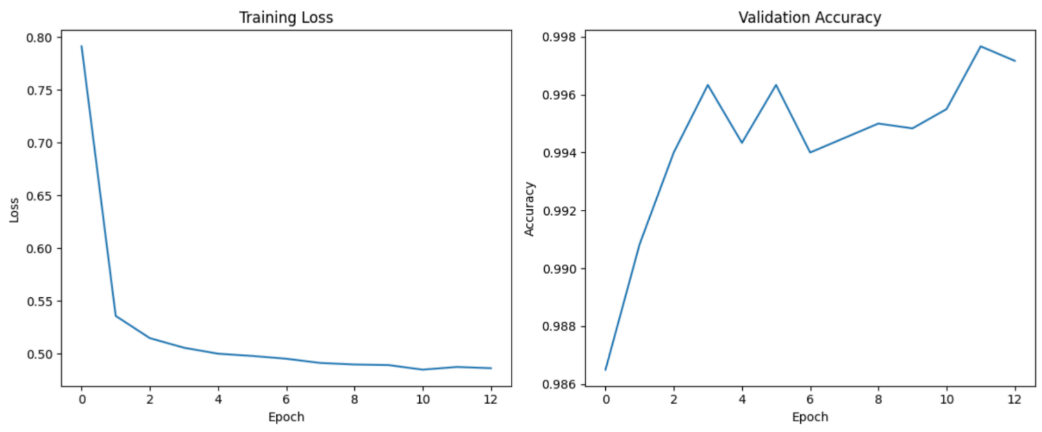
- Closely follows the paper with the same layer structure and dimensions
- Used 100 output channels in convolutional layers as specified in the paper
- Two bottlenecks after pooling layers
- Simple encoder-decoder structure for the bottlenecks
- Adaptive pooling for feature extraction
- Adam optimizer with weight decay (1e-5)
- ReduceLROnPlateau scheduler for learning rate adjustment

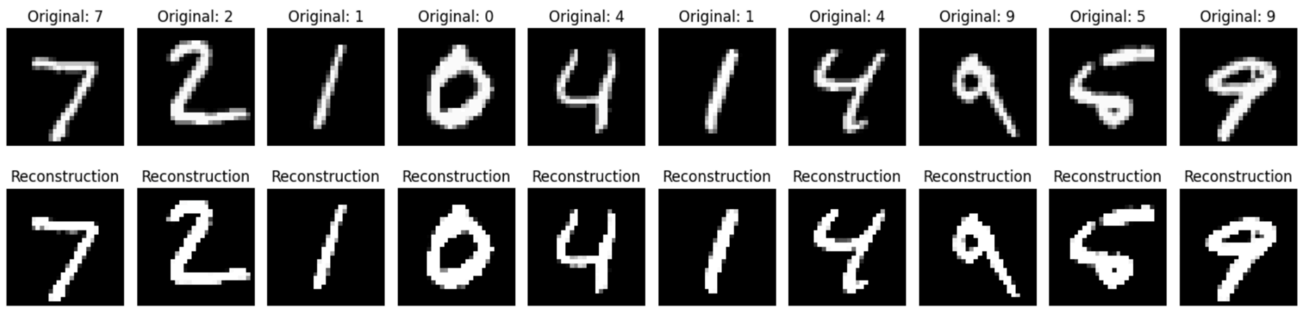
We also experimented here with different:

- Bottleneck dimensions (24, 32, 64), which controls the dimensionality of the latent representation
- Reconstruction Weights (1.0, 2.0, 3.0), which balances the classification and reconstruction objectives during training
- Epochs, to find the most time we can train within the given constraints
- Parameter for Statistical Threshold Method:
    - While the original paper uses Weibull distribution fitting, we found a simpler statistical approach using the mean and standard deviation of distances from class means worked better in practice (we experienced issues with implementing the Weibull Distribution). This involves:
        - Computing the distances of all training samples to their respective class means
        - Setting the threshold at `mean_distance + sigma_multiplier * std_distance`
        - The sigma_multiplier parameter (set to 2.0) controls the trade-off between false positives and false negatives
        - Since heavily tuning this hyperparameter runs counter to the motivation of open-set recognition for handling unknowns, we did not try to find the perfect setting for this case, rather a reasonable one that worked well with the 2 test sets we were given

Intermediate Architecture - Best Results:

(Bottleneck Dim: 32, Recon Weights: 2.0, Epochs: 13, sigma_multiplier: 2.0):

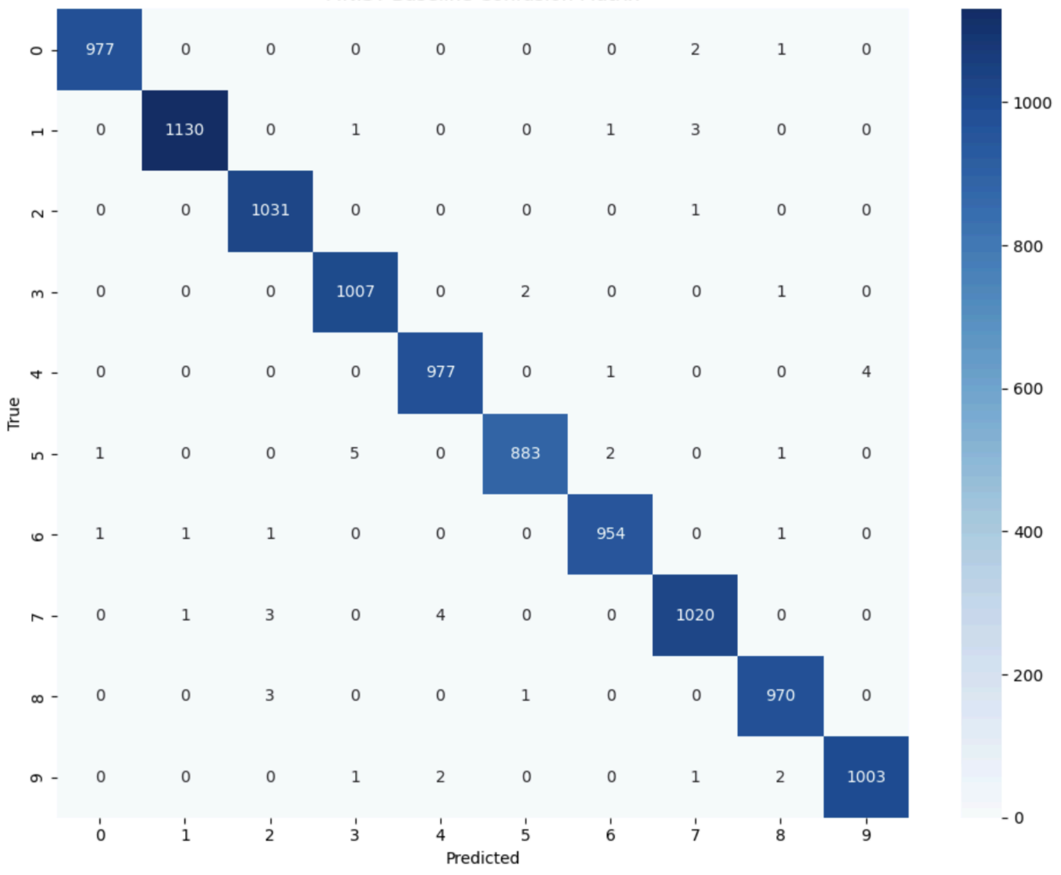




```
Setting statistical threshold: 15.4192
```

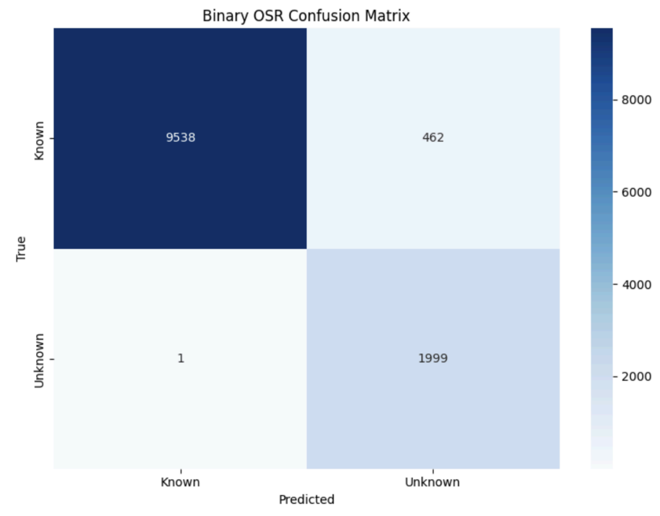
```
=== Evaluation - CIFAR10/Fashion-MNIST ===
Baseline MNIST Accuracy: 99.52%
```

MNIST Baseline Confusion Matrix



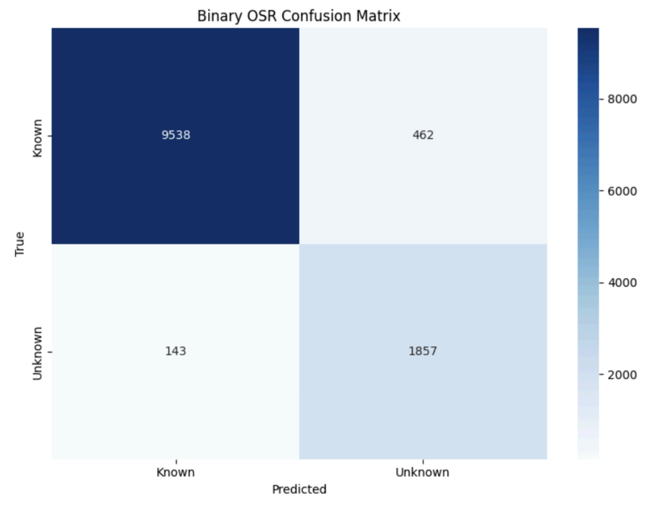
True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	977	0	0	0	0	0	0	2	1	0
1	0	1130	0	1	0	0	1	3	0	0
2	0	0	1031	0	0	0	0	1	0	0
3	0	0	0	1007	0	2	0	0	1	0
4	0	0	0	0	977	0	1	0	0	4
5	1	0	0	5	0	883	2	0	1	0
6	1	1	1	0	0	0	954	0	1	0
7	0	1	3	0	4	0	0	1020	0	0
8	0	0	3	0	0	1	0	0	970	0
9	0	0	0	1	2	0	0	1	2	1003

```
Results for CIFAR10 as OOD:
```

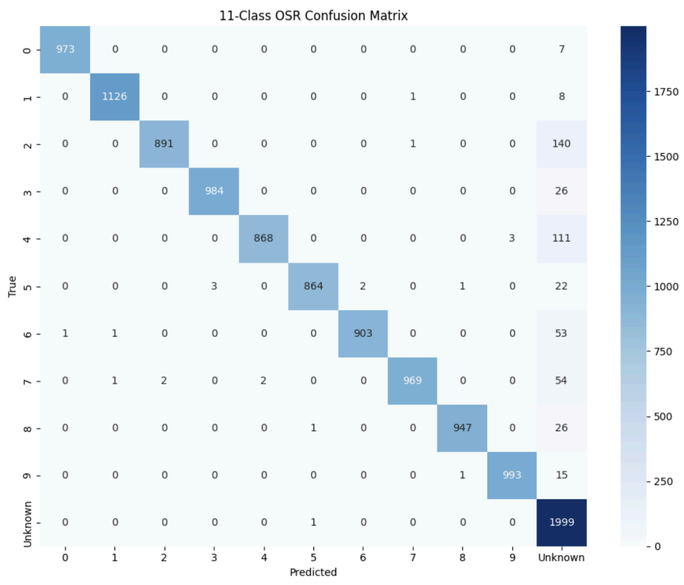


True \ Predicted	Known	Unknown
Known	9538	462
Unknown	1	1999

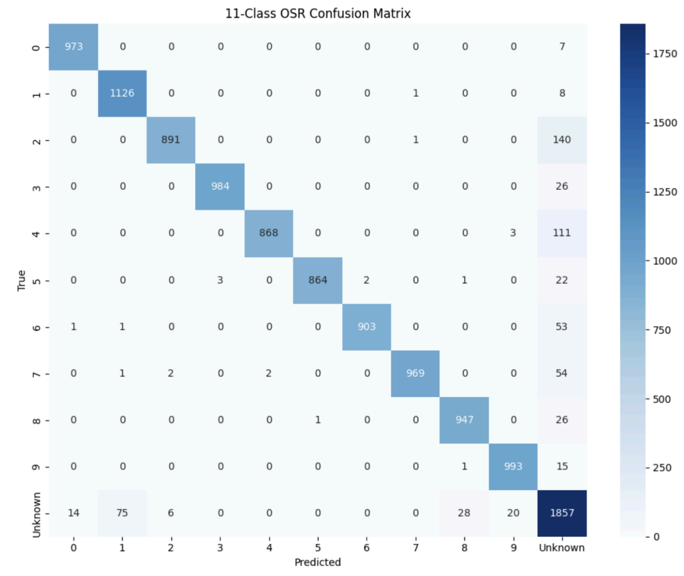
```
Results for FashionMNIST as OOD:
```



True \ Predicted	Known	Unknown
Known	9538	462
Unknown	143	1857



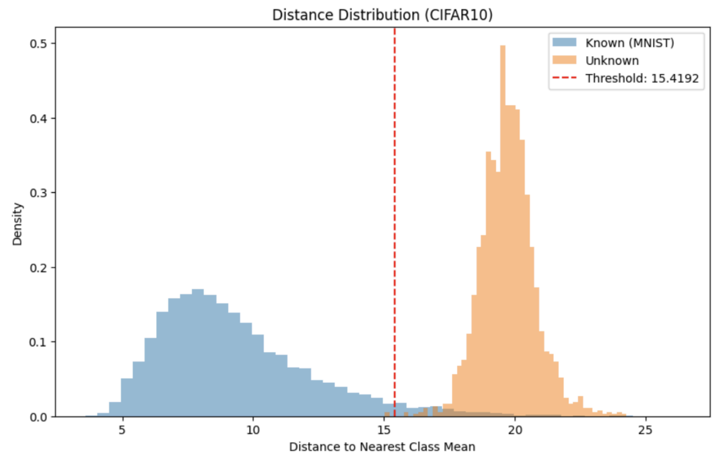
True \ Predicted	0	1	2	3	4	5	6	7	8	9	Unknown
0	973	0	0	0	0	0	0	0	0	0	7
1	0	1126	0	0	0	0	0	1	0	0	8
2	0	0	891	0	0	0	0	1	0	0	140
3	0	0	0	984	0	0	0	0	0	0	26
4	0	0	0	0	868	0	0	0	0	3	111
5	0	0	0	3	0	864	2	0	1	0	22
6	1	1	0	0	0	0	903	0	0	0	53
7	0	1	2	0	2	0	0	969	0	0	54
8	0	0	0	0	0	1	0	0	947	0	26
9	0	0	0	0	0	0	0	0	1	993	15
Unknown	0	0	0	0	0	1	0	0	0	0	1999



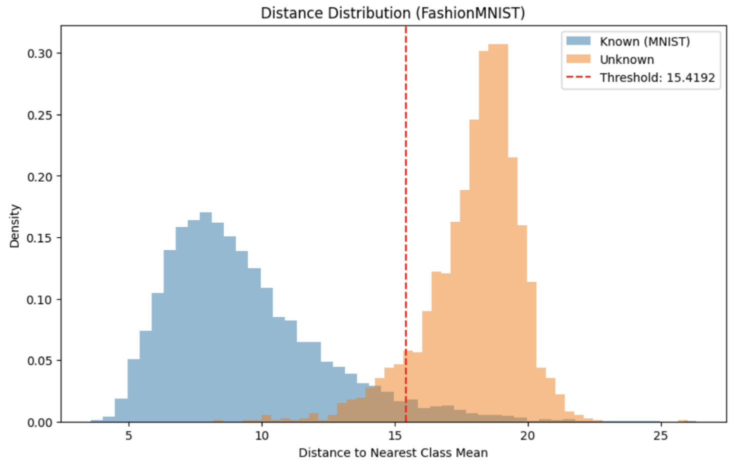
True \ Predicted	0	1	2	3	4	5	6	7	8	9	Unknown
0	973	0	0	0	0	0	0	0	0	0	7
1	0	1126	0	0	0	0	0	1	0	0	8
2	0	0	891	0	0	0	0	1	0	0	140
3	0	0	0	984	0	0	0	0	0	0	26
4	0	0	0	0	868	0	0	0	0	3	111
5	0	0	0	3	0	864	2	0	1	0	22
6	1	1	0	0	0	0	903	0	0	0	53
7	0	1	2	0	2	0	0	969	0	0	54
8	0	0	0	0	0	1	0	0	947	0	26
9	0	0	0	0	0	0	0	0	1	993	15
Unknown	14	75	6	0	0	0	0	0	28	20	1857

```
MNIST Accuracy: 95.18%
```

```
MNIST Accuracy: 95.18%
CIFAR10 OOD Accuracy: 99.95%
Total Accuracy: 95.97%
```



```
MNIST Accuracy: 95.18%
FashionMNIST OOD Accuracy: 92.85%
Total Accuracy: 94.79%
```



Remarks:

While the results were good, training was not very stable as shown above. We decided to experiment further.

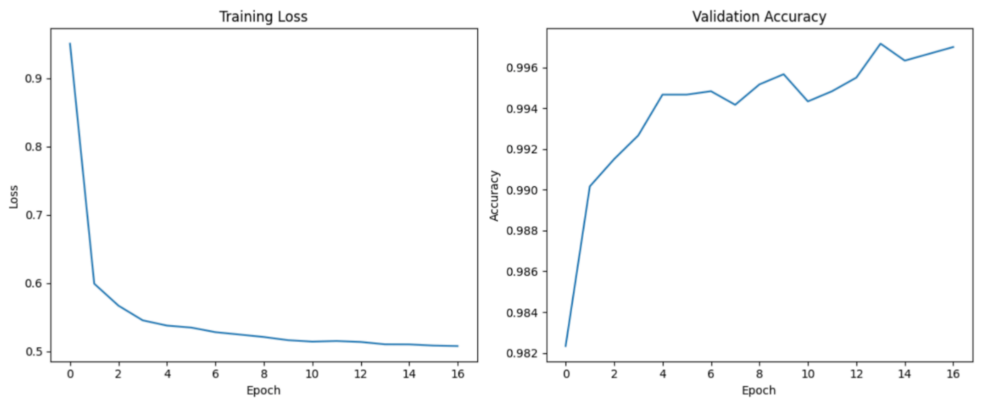
Final Architecture:

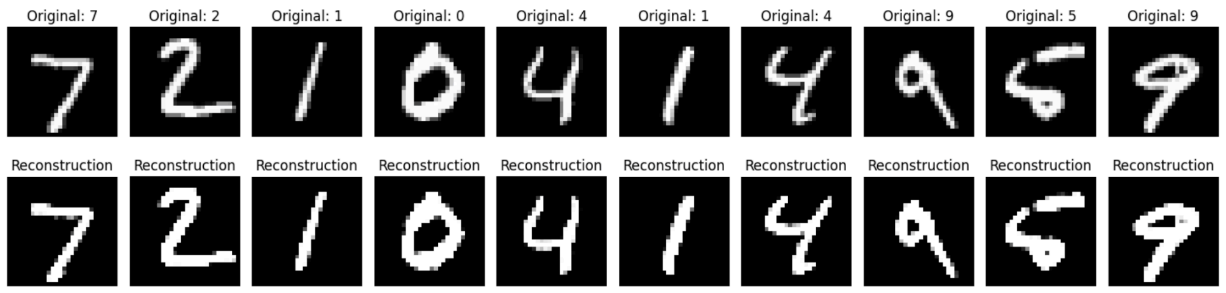
The architectural changes:

- Reduced to 64 output channels in convolutional layers (from 100), made progressive
- Incorporated dropout layers throughout the network (dropout_rate=0.3), in order to provide regularization that helps prevent overfitting to known classes
- Introduced additional bottleneck at a deeper level (three bottlenecks vs. two), to create richer hierarchical representations
- Enhanced bottleneck with BatchNorm for training stability
- More sophisticated encoder-decoder structure
- Global max pooling instead of adaptive pooling for feature extraction

Final Architecture - Best Results:

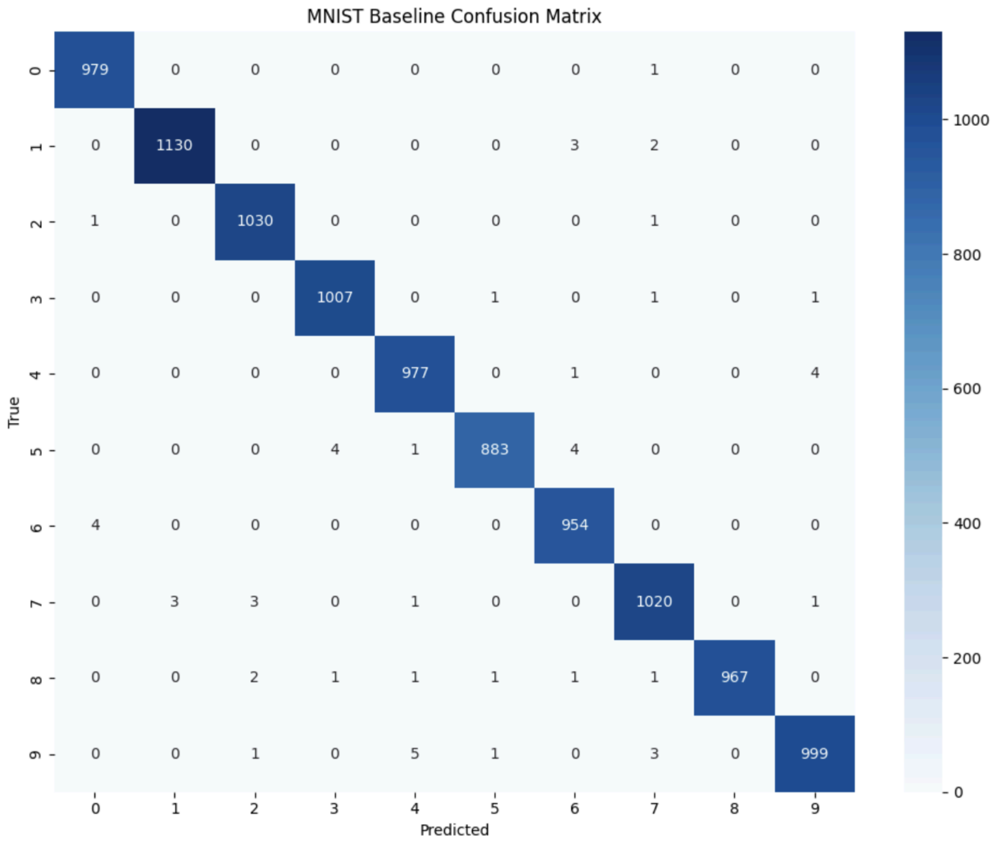
(Bottleneck Dim: 32, Recon Weights: 1.0, Epochs: 17, sigma_multiplier: 2.0)





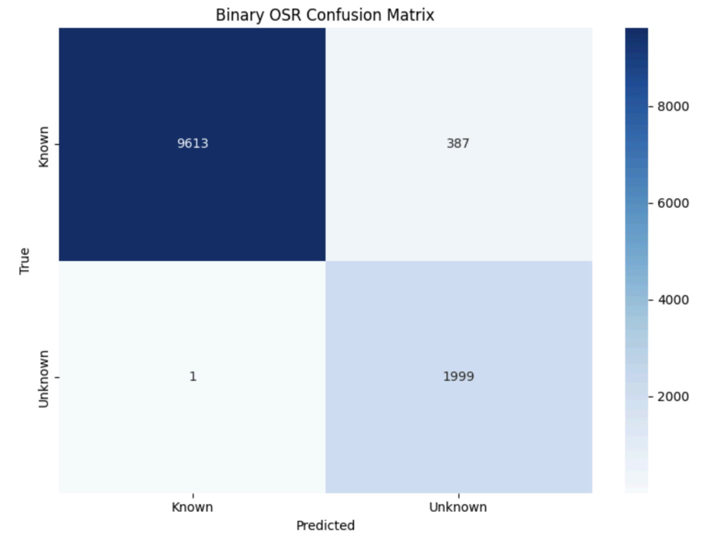
```
Setting statistical threshold: 14.9244 (adjusted for 96-dim space)
```

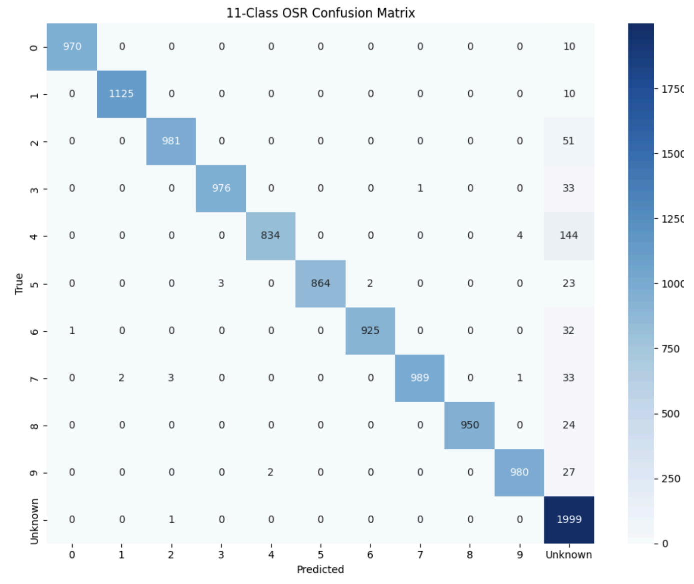
```
=== Evaluation - CIFAR10/Fashion-MNIST ===
Baseline MNIST Accuracy: 99.46%
```



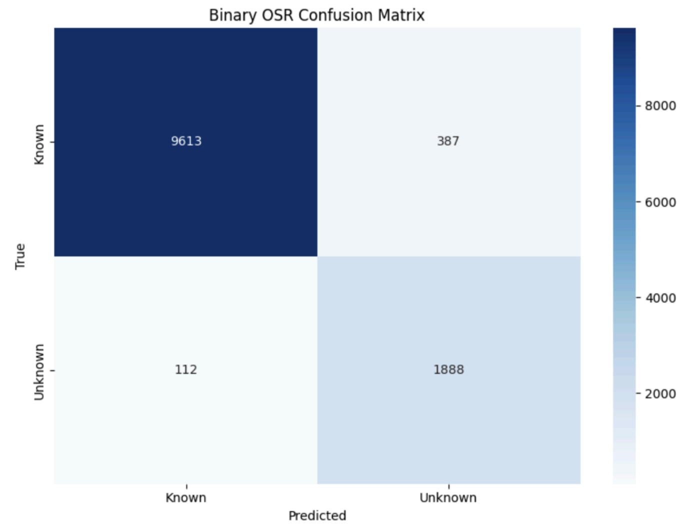
True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	979	0	0	0	0	0	0	1	0	0
1	0	1130	0	0	0	0	3	2	0	0
2	1	0	1030	0	0	0	0	1	0	0
3	0	0	0	1007	0	1	0	1	0	1
4	0	0	0	0	977	0	1	0	0	4
5	0	0	0	4	1	883	4	0	0	0
6	4	0	0	0	0	0	954	0	0	0
7	0	3	3	0	1	0	0	1020	0	1
8	0	0	2	1	1	1	1	1	967	0
9	0	0	1	0	5	1	0	3	0	999

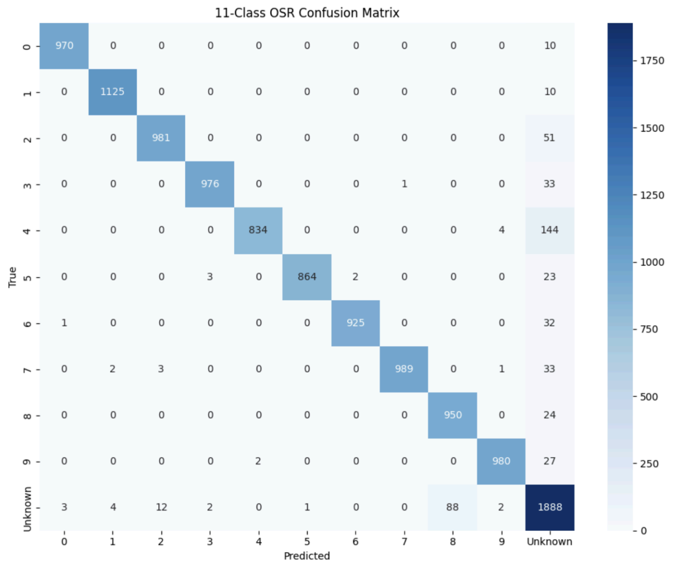
```
Results for CIFAR10 as OOD:
Feature dimensionality: 106
```



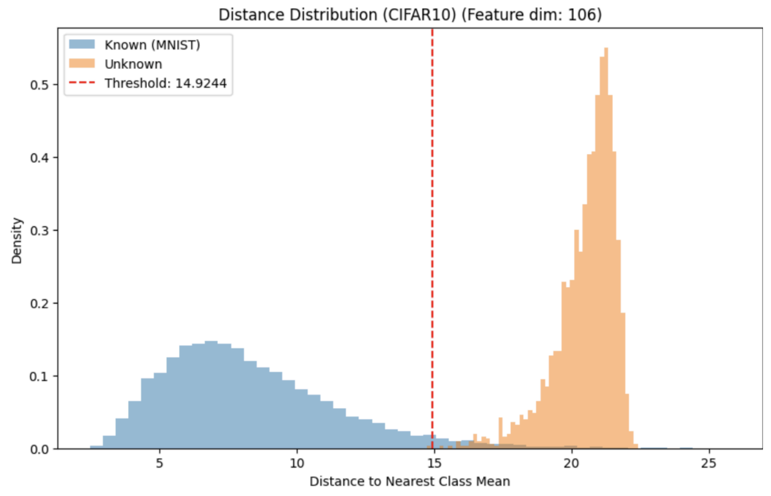


```
Results for FashionMNIST as OOD:
Feature dimensionality: 106
```

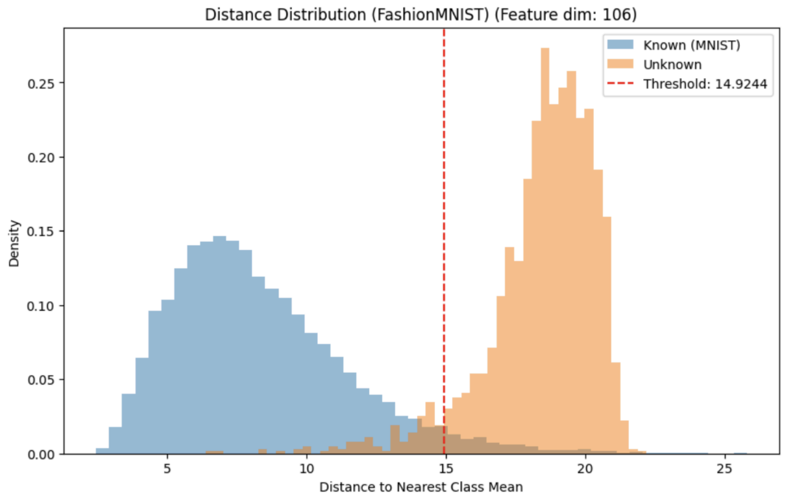




```
MNIST Accuracy: 95.94%
CIFAR10 OOD Accuracy: 99.95%
Total Accuracy: 96.61%
Known distances: mean=8.2987, std=3.1336
Unknown distances: mean=20.5202, std=1.0784
```



```
MNIST Accuracy: 95.94%
FashionMNIST OOD Accuracy: 94.40%
Total Accuracy: 95.68%
Known distances: mean=8.2987, std=3.1336
Unknown distances: mean=18.4677, std=1.9379
```



Comparison of Implementations:

Feature	**Original Paper (our implementation)**	**Intermediate Implementation**	**Final Implementation**
Network Type	Plain CNN with hierarchical bottlenecks	Plain CNN with 2 bottlenecks	Plain CNN/DenseNet with 3 strategic bottlenecks
Convolutional Channels	100 output channels	100 output channels	64-128-256 progression
Regularization	None	None	Dropout layers (0.3 rate)
Bottleneck Design	Lateral connections at each layer	2 bottlenecks after pooling	3 bottlenecks placed strategically
Latent Representation	Hierarchical at each layer	Two-level z1, z2	Three-level z2, z4, z5
Epochs	13 (21 mins)	13 (10 mins)	17 (10 mins)
Baseline MNIST Accuracy	99.58%	99.52%	99.46%
CIFAR10 Binary Accuracy	97.39%	96.14%	96.77%
CIFAR10 11-Class Accuracy	97.15%	95.97%	96.61%

FashionMNIST Binary Accuracy	94.69%	94.96%	95.84%
FashionMNIST 11-Class Accuracy	94.45%	94.79%	95.68%

These figures show that we achieved better performance with each architectural development. The motivation for the changes were mentioned above in the description of each of the implementations. Below we will discuss the limitations that we may not see with these results.

Limitations

As discussed in the Recent Advances Survey, CROSR uses a combined approach where classification and reconstruction are performed jointly within a single framework. This integration, while innovative, creates a fundamental limitation: it must balance these competing objectives simultaneously, which the paper suggests is "extremely challenging." C2AE, for example, supposedly addresses this by separating open set recognition into two distinct subtasks.

The original CROSR paper uses Weibull distribution fitting because it's particularly well-suited for modeling extreme values and has theoretical support from extreme value theory. Our statistical threshold approach is simpler but makes stronger assumptions about the distribution of distances.

The threshold based on mean and standard deviation might work well when:

- The feature space is well-regularized
- Class distributions are relatively compact and similar
- The distance distribution is approximately normal

However, it might be less effective when:

- Class distributions have varying shapes or scales
- The distance distribution has a heavy tail
- There are outliers in the training data that skew the statistics

In addition, while the original CROSR paper proposes a flexible hierarchical bottleneck structure that can potentially extract features from many layers, our implementation uses only three strategic bottlenecks. This reduction may limit the model's ability to capture the full spectrum of hierarchical features necessary for optimal performance on complex data.