

---

# BLISSNet: Zero-Shot Classification using BLISS

---

Michael Menezes   Sanya Garg  
Rice University  
Houston, TX 77005  
{mnm5, sg157}@rice.edu

## 1 Abstract

Image classification has become an important technique used in many forward-facing applications such as satellite imagery analysis and traffic cameras. Zero-shot classification, where models attempt to classify images from classes they were not trained on, has been a particularly difficult setting. A popular approach for zero-shot image classification is gaze embeddings which operate by extracting features from images and using the feature histogram to associate new classes [3].

In this work, we pursued a similar technique for zero-shot learning. Instead of feature histograms, we use the BaLanced Index for Scalable Search (BLISS) [1] algorithm to cluster features extracted from a resnet feature extractor [2]. BLISS utilizes iterative repartitioning to separate classes into load-balanced, relevant buckets.

We specifically look at the Canadian Institute For Advanced Research - 100 (CIFAR) dataset. Using this dataset, we train  $n$  separate models – one per metaclass mapping. Then as done in BLISS, to classify any image we have associations for, we use each model to determine its  $n$  metaclasses, which we combine to associate the image with its predicted label. To incorporate a novel class into our classification system, we use an inverted index with a frequency filter to determine the top associated metaclasses for the new label and place the label in those metaclasses.

While developing our zero-shot classifier, we discovered a bias in the BLISS algorithm and redirected our attention to solving this issue through initialization schemes, repartitioning algorithms, and loss functions. Ultimately, while some gains were made, we were unable to find significant improvements to the bias issue.

## 2 Literature Survey

In the past, gaze embeddings have been used to perform zero-shot image classification. Gaze embeddings utilize the concept of human gaze as auxiliary information instead of past zero-shot image classification methods that require the annotation of discriminative object properties. Gaze embeddings work by extracting discriminative descriptors from the data and using three gaze embeddings to learn a compatibility function between images and the concept of gaze. The benefits of gaze embeddings are that data collection is very fast and implicit (because class attributes do not have to be explicitly discerned) and that data labeling and object discrimination do not necessarily have to be performed by experts.

Furthermore, BLISS is a highly scalable indexing algorithm that can be used for information retrieval by finding an approximate set of nearest neighbors for a point in a high-dimensional vector space. Its small index sizes make it very scalable and its iterative repartitioning technique ensures that buckets are balanced. BLISS operates by placing items in buckets based on query-item relevance data. Additionally, BLISS uses a power-of-K choices strategy when determining which bucket to place each point in to ensure that buckets are balanced overall. BLISS can be used for both near-neighbor retrieval and extreme classification.

BLISS aims to address the pitfalls of other space partitioning techniques such as Local-Sensitivity Hashing, K-Means, and Learning to Hash. It does this by implementing indexing while aiming to achieve more balanced partitions. Unlike LSH and K-Means, which often result in uneven partitions and slow queries, BLISS uses non-linear mapping to create partitions that handle complex data more effectively. It also addresses the load imbalance problem seen in Learning to Hash and Learning to Index, where some partitions end up overloaded while others are underused.

Throughout our project, we used resnet feature extractors to apply BLISS to image data in CIFAR-100. A resnet feature extractor addresses the drop in accuracy that occurs when layers are continually added to a deep convolutional network model. It does so by utilizing a deep residual learning framework, reformulating added layers as learning residual functions with respect to the inputs to the layer. Overall, resnet models are easier to optimize and able to achieve higher accuracies than their respective traditional deep learning models.

**We believe that iterative partitioning can provide an efficient method of classifying new images using trained features.**

### 3 Methods

Our goal is to use BLISS to perform zero-shot image classification. Since BLISS is an approximate nearest neighbors that also generates buckets of similar points, there are two main ways that BLISS can be used. The first is by using BLISS to create metaclass clusters of similar classes. We do this  $n$  times. Then, we train  $n$  image classification models. Each image classification model is trained to associate images with its metaclass, given by the metaclass mappings generated from BLISS. Inferencing on an image operates by using each image classification model to predict a metaclass. Then, we use an inverted index to pick the most frequent class. The second way to use BLISS is to associate image vectors with similar image vectors. Then, inferencing an image is querying BLISS to get labels of the nearest neighbors and returning the most frequent class.

#### 3.1 Experimental Settings

##### 3.1.1 Early Experiments

For our early experiments, we have two main experimental settings: Beta, and Cifar. In the Beta setting, we are only testing BLISS. In the Cifar setting, we are testing BLISS-Net. The Beta setting has its parameters set as follows:

```
d = 2 # input data dimension
R = 4 # number of repetitions
B = 5 # number of buckets
K = 3 # load balance parameter
M = 3 # number of top buckets to consider while querying
T = 60 # training epochs
EPOCHS_PER_LOAD_BALANCE = 2 # epochs per load balance
L = 256 # number of labels
BATCH_SIZE = 4
P = 5 # number of data points in each y_bar
Q = 32 # number of data points to manually rank in query
```

We generate synthetic 2D data by sampling x-coordinates from a bimodal beta distribution ( $\alpha = \beta = 0.1$ ), and likewise for y. While the data distribution is held constant, we vary the distance metric, rebalancing schemes, and initialization point. We have tried Mean Squared Error (MSE), and Cosine Distance for the distance metrics. For rebalancing schemes, we have tried the original formulation of picking the least loaded from the top  $k$  buckets; and we have tried picking the top from the  $k$  least loaded buckets. We have also tried initializing the model from default values as well as from Kaiming initialization.

In the Cifar setting, the parameters are set as follows:

```

d = 100352 # input data dimension
R = 4 # number of repetitions
B = 20 # number of buckets
K = 4 # load balance parameter
M = 4 # number of top buckets to consider while querying
T = 60 # training epochs
EPOCHS_PER_LOAD_BALANCE = 2 # epochs per load balance
L = 40000 # number of labels
BATCH_SIZE = 64
P = 128 # number of data points in each y_bar
Q = 128 # number of data points to manually rank in query

```

As the name suggests, instead of sampling from a beta distribution, we sample from the CIFAR-100 dataset. We use the Resnet-50 feature extractor to convert images into semantic feature vectors hence why the input of the BLISS algorithm has  $d = 100352$ . In this setting, we vary the image classification algorithm as explained at the start of section 3.

### 3.1.2 Later Experiments

Our early experiments helped guide the direction of our later experiments. In particular, we noticed that BLISS was biased towards putting points inside of the same buckets rather than grouping them based on similarity like we expected. As a result, the numbers of buckets that BLISS would produce would exactly correspond with the value of  $K$  that we provided when querying the  $K$ -top buckets during the repartitioning phase. Our following experiments sought to rectify this bias by more closely investigating and quantifying this bias. We use Silhouette score, a popular internal metric, and V-Measure, a popular external metric, for quantifying how well BLISS clustered the data.

Our BLISS model was set with the following hyper parameters:

```

d = 2 # input data dimension
R = 4 # number of repetitions
B = 20 # number of buckets
K = 4 # load balance parameter
M = 4 # number of top buckets to consider while querying
T = 60 # training epochs
EPLB = 2 # epochs per load balance
N = 128 # number of labels
BATCH_SIZE = 8
P = 8 # number of data points in each y_bar
Q = 8 # number of data points to manually rank in query

```

We varied the number of clusters  $\{ 2, 4, 11, 13, 20, 22 \}$  in the dataset. We chose these values as we fixed BLISS's number of buckets to 20 and we wanted to test how the algorithm performed across the under-saturated, saturated, and over-saturated settings. Clusters were generated by sampling, for each coordinate, from normal distributions with means uniformly chosen from  $U[-10, 10]$  with variance chosen from  $U[0.5, 1]$ .

### 3.1.3 Varying Loss Functions

Furthermore, to address the bias issue with a different approach, we also explored using different loss functions within BLISS's internal model. The BLISS algorithm utilized cross entropy loss, which we also originally incorporated into our implementation. Cross Entropy loss leads the model to assign high probabilities to the correct classes by minimizing the negative log-likelihood of the true class probabilities.

We explored replacing Cross Entropy loss with Triplet Loss based on Dr. Shrivastava's suggestion. Triplet loss works by using triplets, which include an anchor item, a positive representing a similar item, and a negative representing a dissimilar item. It aims to create a space in which embeddings similar samples are closer together than different samples. The model can later use those distances to



group samples together. We hypothesized that the negative or dissimilarity component of triplet loss would cause it to perform better than Cross Entropy loss.

## 4 Results

We started off our work by porting BLISS to pytorch. After running some basic experiments that use BLISS to cluster points from a beta distribution, we noticed that BLISS tended to place all points in the same  $k$  buckets, as seen in Figure 1.

As BLISS is a crucial component of our project, we tried to rectify this issue by trying different load-balancing schemes. Instead of picking the least loaded bucket from the top  $k$  buckets we tried picking the top bucket from the  $k$  least loaded. The results are seen visually in Figure 2. This had the opposite effect of what we wanted as the clusters degenerated into just two clusters.

Independent from trying a different rebalancing scheme, we also tried initializing the neural net from a better starting point. By initializing the neurons using Kaiming initialization, we saw slightly better load balancing with the resulting number of clusters being close to  $k$  but not exactly. Results can be seen in Figure 3 and 4.

In the following results, we have two types of plots. The first plot takes an  $(x, y)$  coordinate, queries the model, computes the distance to the returned nearest neighbor, and plots this distance as a color. Here, we are looking for bright colors around data points which shows that the nearest neighbor estimated by the model is actually near by. Data points are colored by which cluster they were generated in. See 5 for an example. For the second type of plot, we color data points based off of what bucket the BLISS model put the data point in. More specifically, each data point is mapped to a tuple like  $(b_1, b_2, \dots, b_n)$  where we have  $n$  model repetitions and each  $b_i \in \{0, \dots, B\}$  where  $B$  is the number of buckets. The tuple is then converted into a  $B$ -ary fraction which is then used to select a color from a perceptually uniform gradient which ranges over  $[0, 1]$ . See 6 for an example.

In Figure 5, we see four distinct clusters. These clusters are colored purple, green, yellow, and blue based on their ground truth value (which normal distribution they were generated from). This plot is a result of Experimental Setting 1 which used Kaiming initialization, the regular re-balancing algorithm, and triplet loss. As evidenced by the monochromatic coloring, the BLISS algorithm did a good job at partitioning the dataset as the data points are surrounded by whiter pixels. Despite the plot showing good nearest neighbor selection, the Silhouette and V-measure scores indicate poorer performance.

In Figure 6, we see the same clusters as in Figure 5. The data points here, unlike in Figure 5, are colored based on their BLISS bucket. While the colors do indicate some separation between clusters, some colors are split among multiple clusters and some clusters contain multiple colors. Some clusters containing multiple colors can be good as it indicates that we are load balancing effectively and making querying faster. On the other hand, having a single color split across multiple clusters indicates that a bucket contains unrelated points and is generally not good for clustering. This may explain the lower Silhouette and V-measure scores we saw in Figure 5.

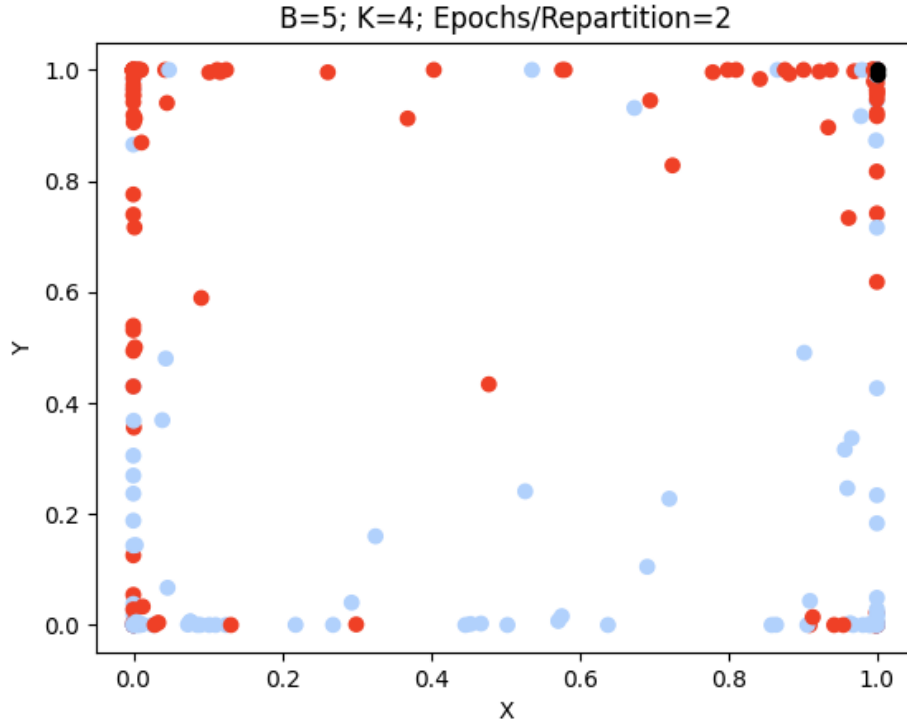


Figure 2: Top Bucket From  $k$  Least Loaded Plot

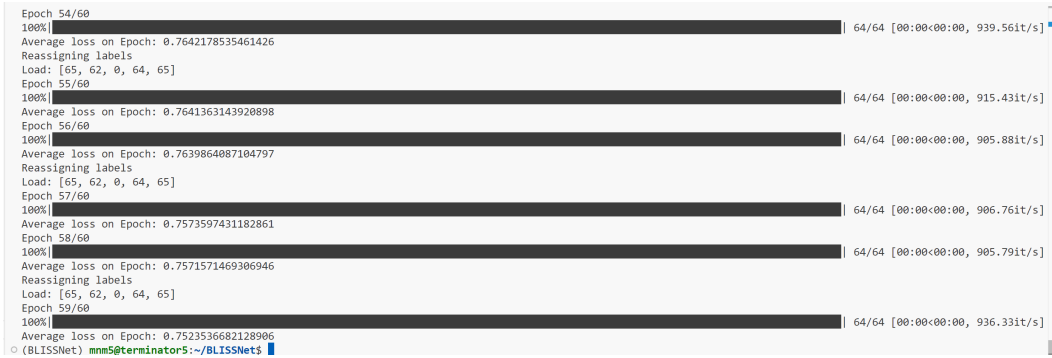


Figure 3: BLISS+Kaiming Training Load Distribution K=3

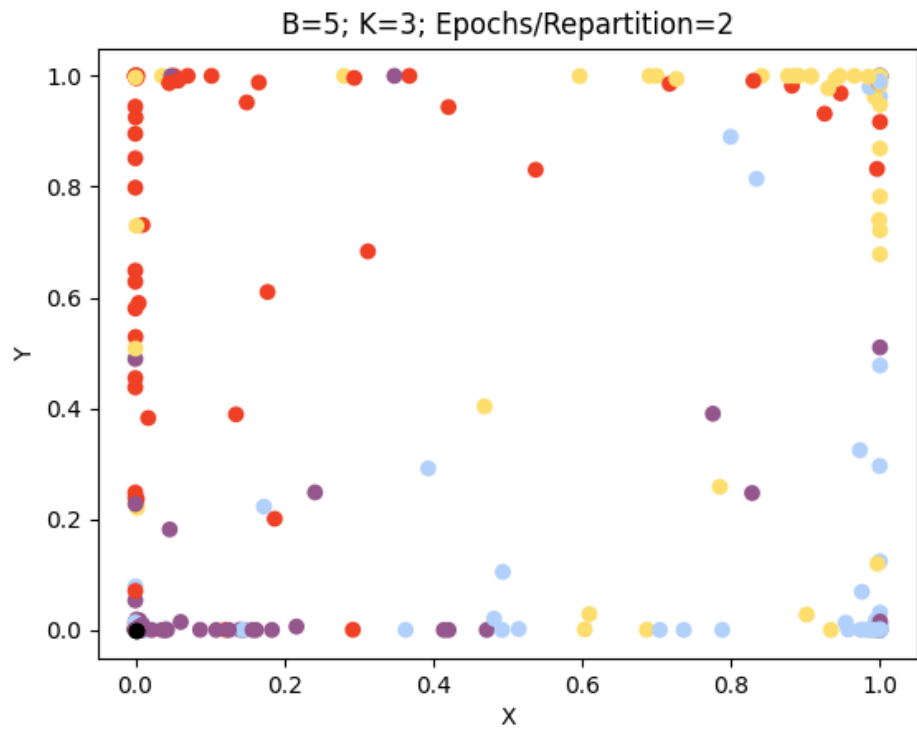


Figure 4: BLISS+Kaiming Training Load Distribution K=3 Plot

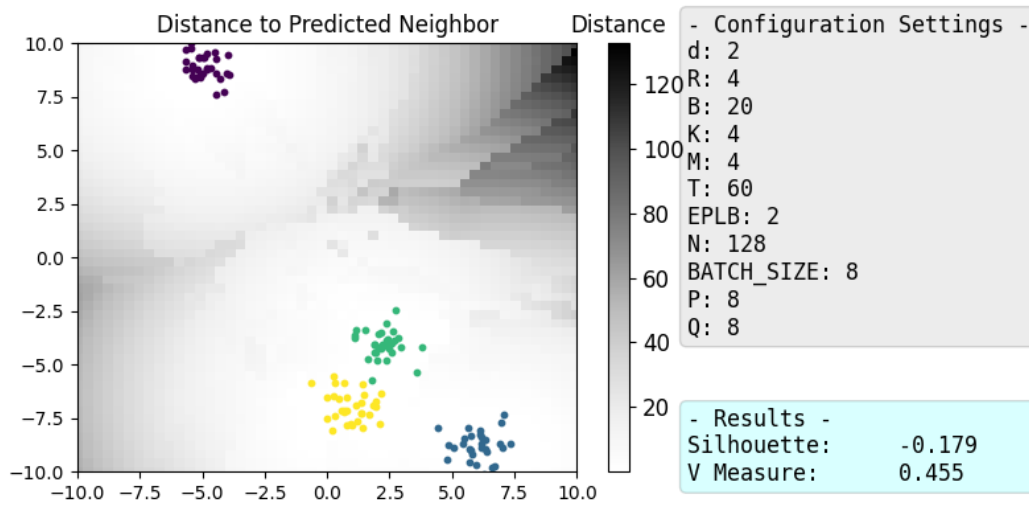


Figure 5: BLISS Training Load Distribution K=3

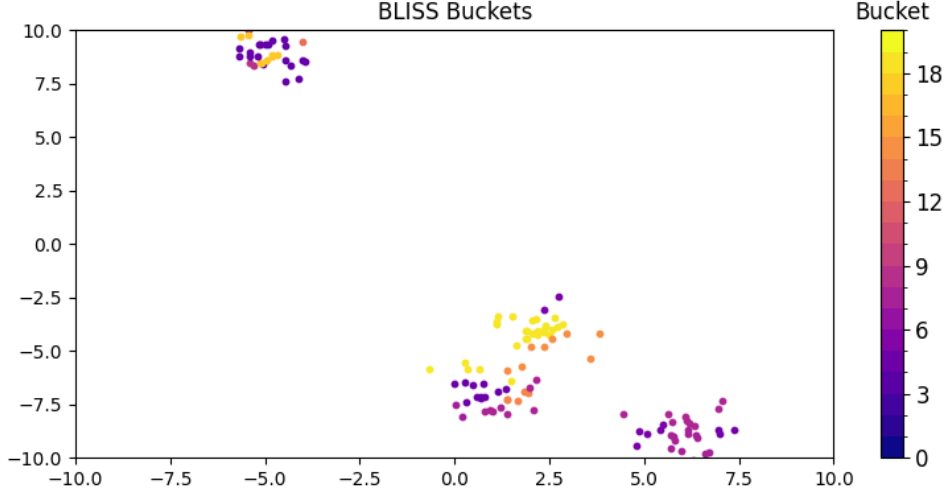


Figure 6: BLISS Training Load Distribution K=3

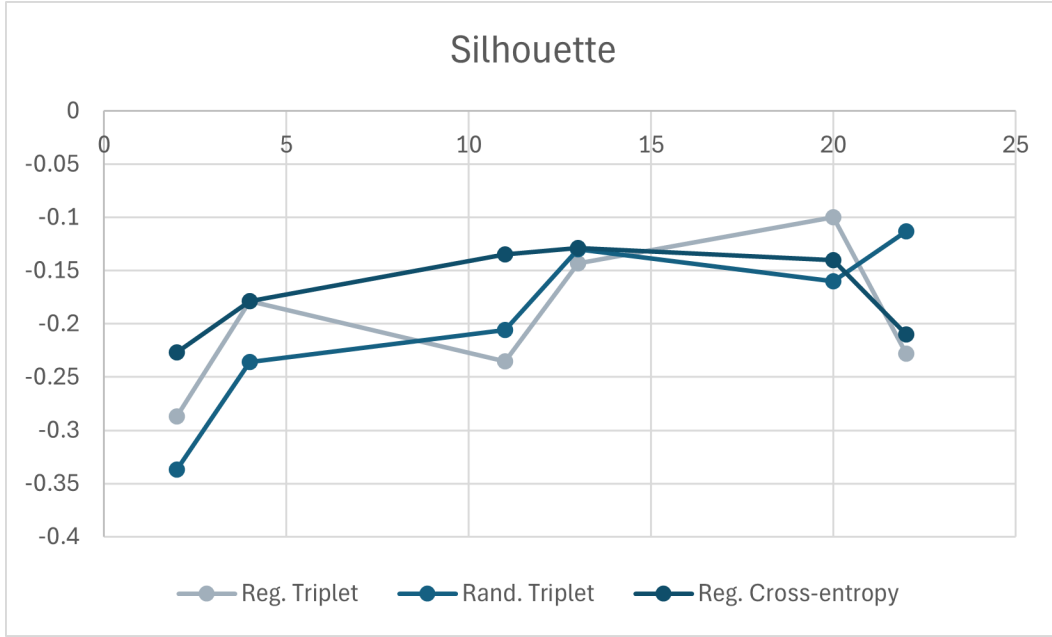


Figure 7: Silhouette vs. No. Cluster

The Silhouette scores across the various experimental settings are displayed in Table 3. Likewise, the V-Measure scores are displayed in Table 4.

In Table 3, we can see that all scores are negative. This implies that data points have been assigned to suboptimal clusters. Additionally, we can see that scores generally improve in the saturated region but fall again when over-saturation hits.

In Table 4, we can see that most scores are around 0.5. This implies that data points have been assigned to moderately okay clusters. Additionally, we can see that scores generally improve with saturation.

In Figure 7, we can see that the regular cross-entropy setting generally outperforms the others until over-saturation starts kicking in. Once the number of clusters is high in comparison to the number of buckets, Triplet loss seems to perform better. Surprisingly, the random initialization outperforms the

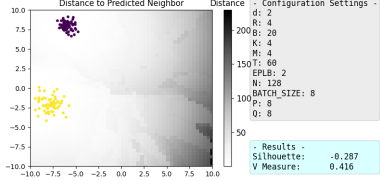
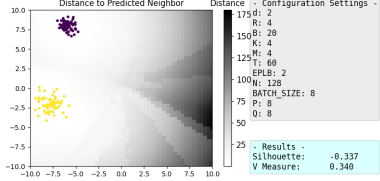
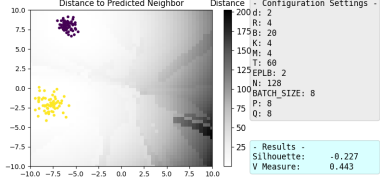
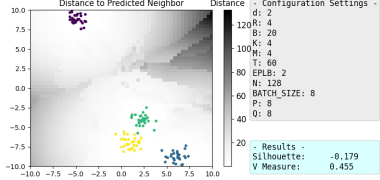
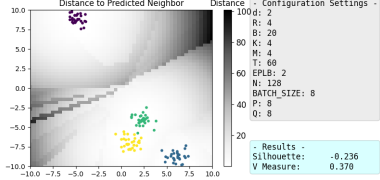
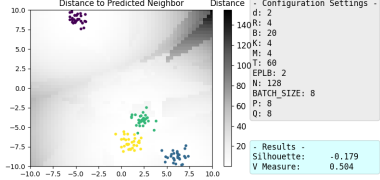
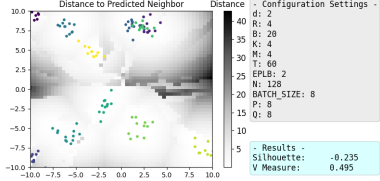
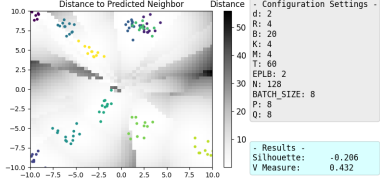
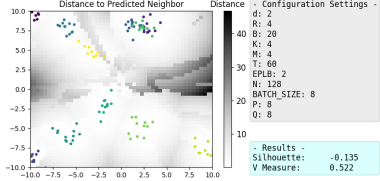
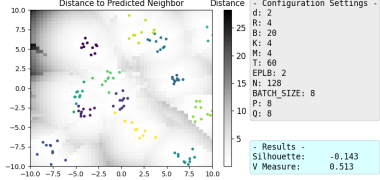
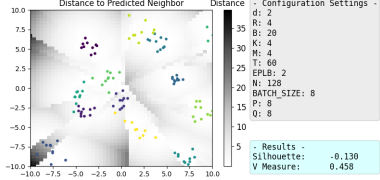
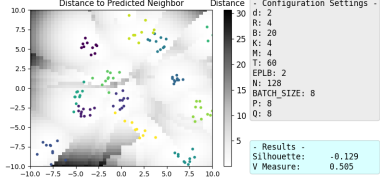
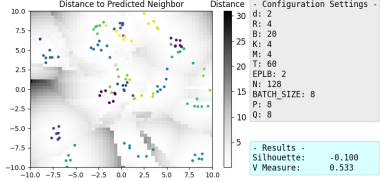
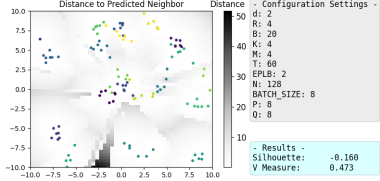
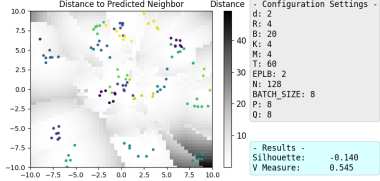
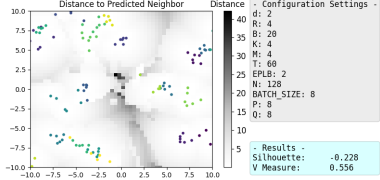
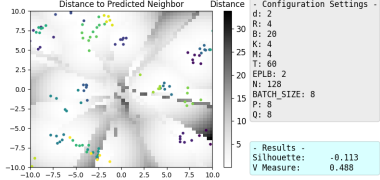
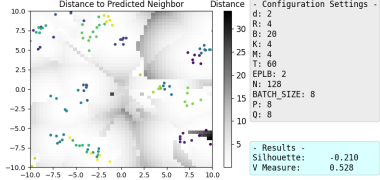
No. Clusters	Kaiming initialization Reg. balance Triplet loss	Random initialization Reg. balance Triplet loss	Kaiming initialization Reg. balance Cross-entropy loss
2			
4			
11			
13			
20			
22			

Table 1: Distance to Nearest Neighbor across Experimental Settings

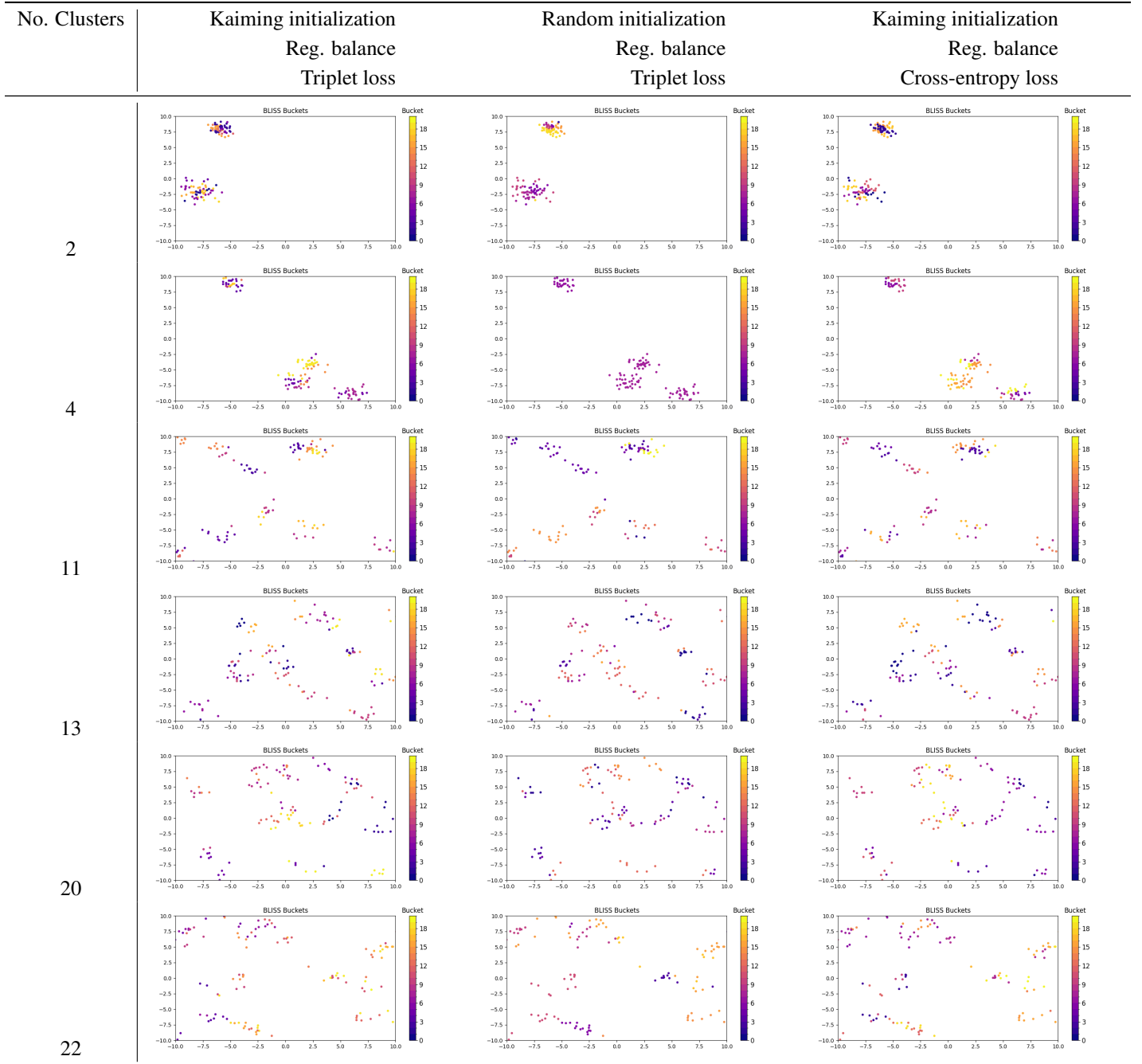


Table 2: BLISS Buckets across Experimental Settings

Table 3: Silhouette			
no. clusters	Reg. Triplet	Rand. Triplet	Reg. Cross-entropy
2	-0.287	-0.337	-0.227
4	-0.179	-0.236	-0.179
11	-0.235	-0.206	-0.135
13	-0.143	-0.13	-0.129
20	-0.1	-0.16	-0.14
22	-0.228	-0.113	-0.21

Table 4: V-Measure

no. clusters	Reg. Triplet	Rand. Triplet	Reg. Cross-entropy
2	0.416	0.34	0.443
4	0.455	0.37	0.504
11	0.495	0.432	0.522
13	0.513	0.458	0.505
20	0.533	0.473	0.545
22	0.556	0.488	0.528

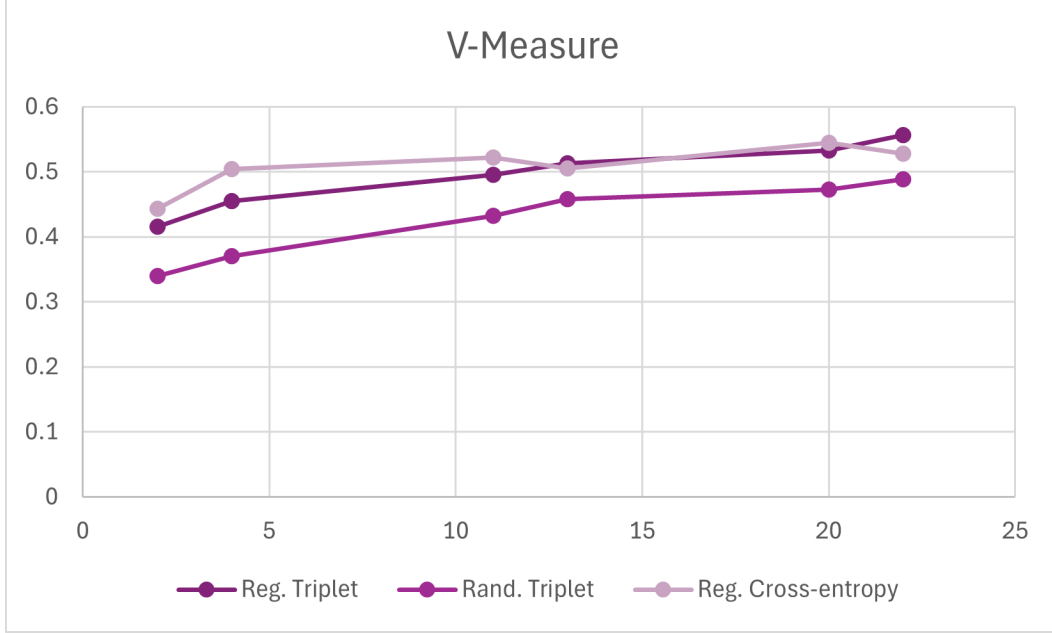


Figure 8: V-Measure vs. No. Cluster

other two in the 22 cluster setting. Also, surprising is Triplet’s drop in performance in the 11 cluster setting.

In Figure 8, we see that regardless of the initialization scheme, both Triplet configurations trend similarly across the number of clusters. Again, we see that Cross-entropy outperforms the Triplet until saturation starts kicking in and the number of clusters approach that of the number of buckets.

In Figure 9, we plotted CIFAR-100 images transformed by PCA and colored based on their BLISS bucket assignments in the algorithm’s fourth repetition. The PCA visualization of the CIFAR-100 images does not appear to reveal any clear clustering patterns based on the BLISS bucket assignments. It seems like the data points are scattered across the 2D projection without distinct groupings of the same color, suggesting the BLISS algorithm may not have been able to partition the high-dimensional feature space in a way that reflects meaningful relationships between the samples.

In Figure 10, we can see the change in logit norms during the training of BLISS across epochs on the synthetic beta distribution data set. There is a clear and consistent upward trend in the logit norms as epochs increase. The increasing logit norms reflect a stronger confidence or separation between the buckets being formed.

Figure 11 depicts the variance of the logit values across epochs during the training of the BLISS algorithm on the synthetic beta distribution data set. The variance fluctuates without stabilizing, indicating that while the algorithm may be making some progress in forming buckets, it does not achieve consistent separation or convergence.

In Figure 12, we can see the change in logit norms during the training of BLISS across epochs on CIFAR-100. Clearly, the logit norms fluctuate significantly without converging to a fixed value or

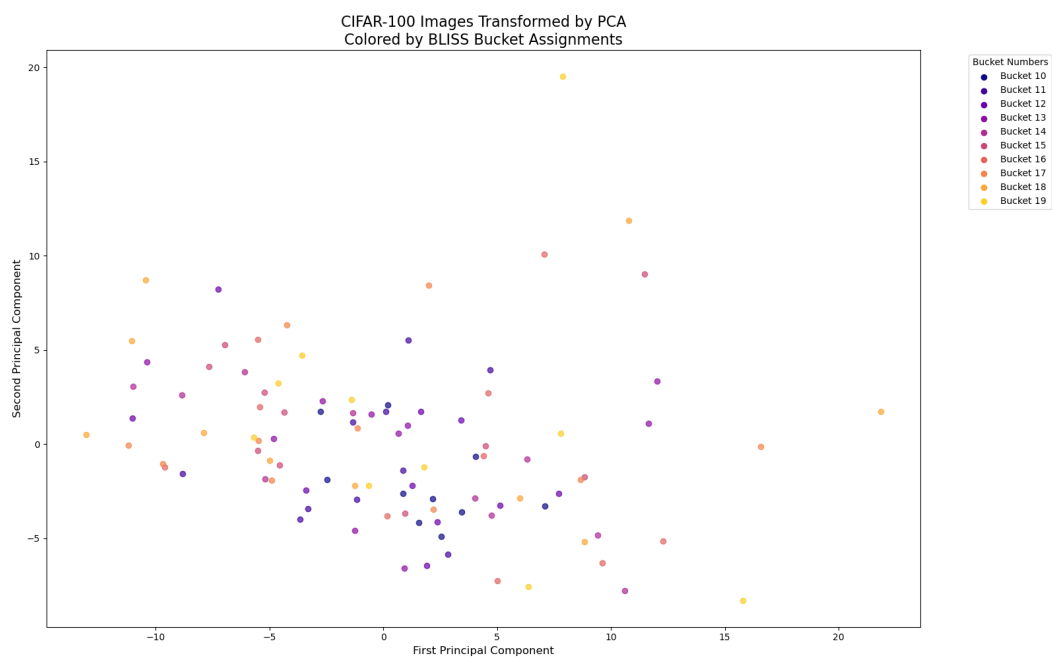


Figure 9: CIFAR-100 Images Transformed by PCA and Colored by BLISS Bucket Assignments (4th repetition)

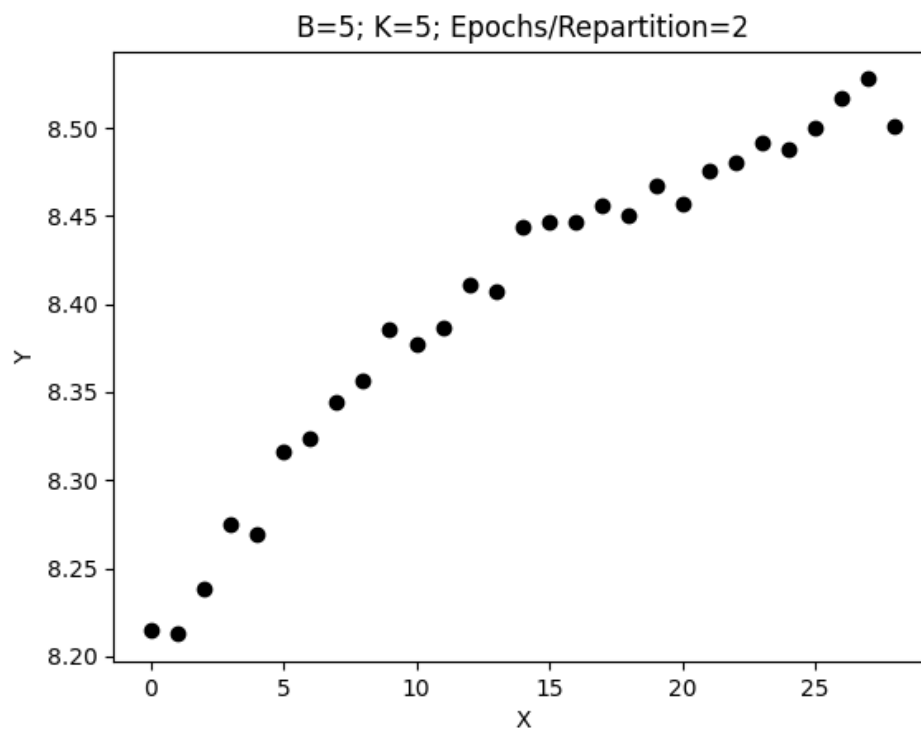


Figure 10: BLISS Change in Logit Norm during Training on Beta Distribution Data

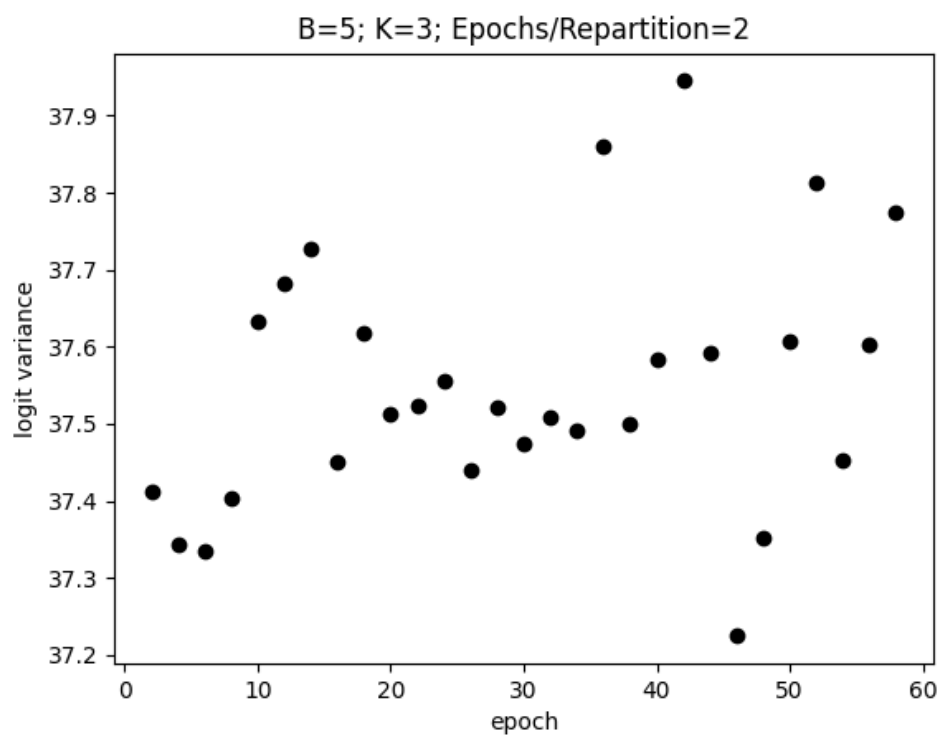


Figure 11: BLISS Change in Variance during Training on Beta Distribution Data

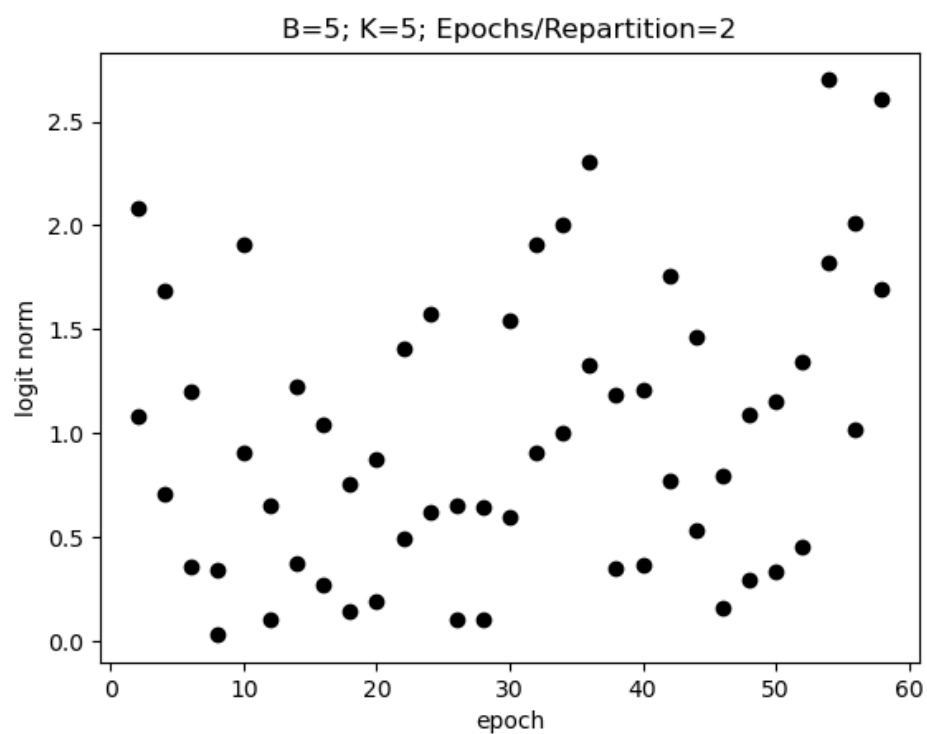


Figure 12: BLISS Change in Logit Norm during Training on CIFAR-100

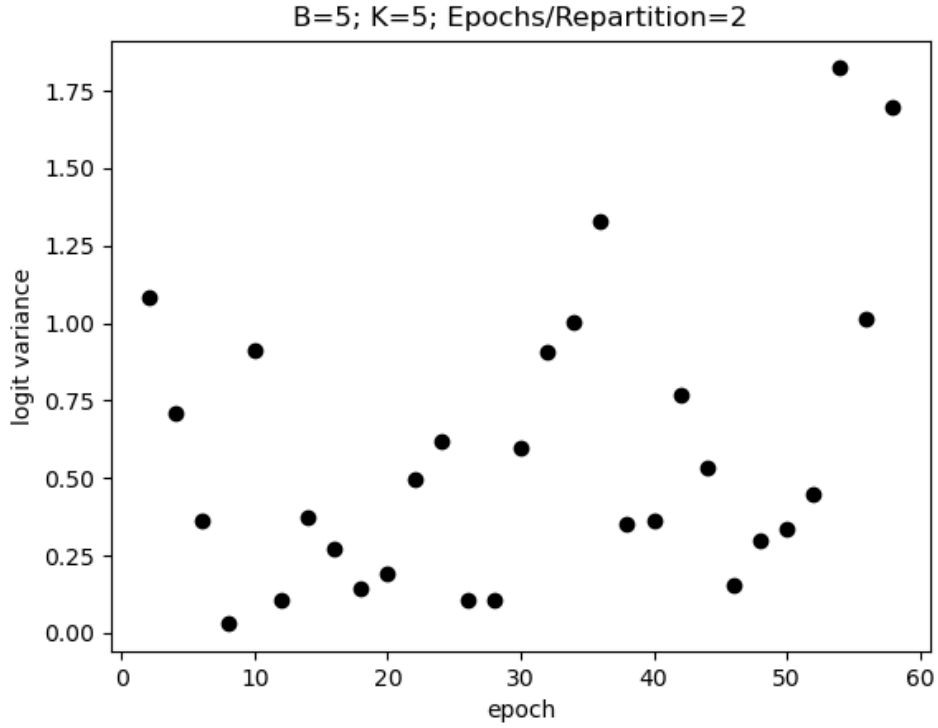


Figure 13: BLISS Change in Variance during Training on CIFAR-100

demonstrating a clear upward or downward trend, indicating variability and instability in the learning process. This suggests BLISS is unable to create well-defined buckets or reach a fixed stopping point.

Figure 13 depicts the variance of the logit values across epochs during the training of the BLISS algorithm on CIFAR-100. The logit variance fluctuates throughout training, with no clear convergence or stabilization point. While the variance occasionally increases (especially towards later epochs), there is no consistent upward or downward trajectory, indicating that the logits are not consistently narrowing toward a single distribution or diverging.

## 5 Conclusion

As a whole, our project initially aimed to train separate BLISS models to generate metaclass mappings and use inverted indices to perform zero-shot image classification. While experimenting with the BLISS algorithm, we identified a significant bias in the algorithm, as we discovered that clusters were influenced more by load-balancing parameters than by actual feature similarity. As a result, we investigated multiple strategies to address this bias, including different initialization schemes, repartitioning algorithms, and loss functions. Despite some improvements, our efforts did not fully resolve the clustering bias present in BLISS. However, our results can be used for future exploration and refinement of the BLISS algorithm.

## References

- [1] Gupta, G., Medini, T., Shrivastava, A., & Smola, A. J. (2022, August). Bliss: A billion scale index using iterative re-partitioning. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 486-495).
- [2] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
- [3] Karessli, N., Akata, Z., Schiele, B., & Bulling, A. (2017). Gaze embeddings for zero-shot image classification. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4525-4534).