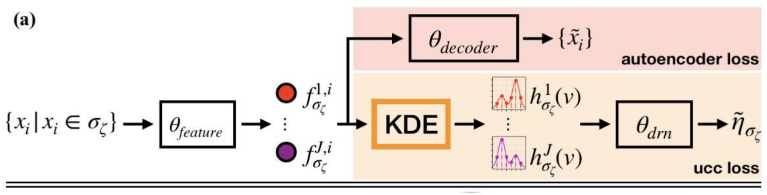
End-to-End Multi-Instance Learner (MIL) w/ Kernal Density Estimation (KDE)

Exploration of “Weakly Supervised Clustering by Exploiting Unique Class Count”



Introduction

Machine learning operates mainly under two frameworks: supervised learning, where data is labeled to guide learning, and unsupervised learning, which uses unlabeled data. Both aim to identify key data features for classification or prediction. Oner et al. [2019] focus on weakly supervised learning, a type of supervised learning with limited label data. This method uses 'bags' of input samples, with supervision provided by the count of unique classes in each bag. The challenge is to classify individual instances within these bags.

Data Pipeline

The process starts by transforming datasets like MNIST, CIFAR10, and CIFAR100 into bags containing up to K unique classes (K being the total number of classes). Training data is created from the power set 2^X of the original dataset. Each bag's label is its count of unique classes, focusing on 1 to 4 classes to manage complexity.

Generating Embeddings (Encoder-Decoder)

Here, data passes through a network based on wide residual networks (Zagoruyko and Komodakis [2016]). This design, preferring wider over deeper networks, balances performance with reduced complexity.

After extracting features, they're decoded to reconstruct the original input, testing the feature extraction's effectiveness. The process uses Mean Square Error loss for optimization.

Learning Permutation-Invariant Features (KDE)

The final step involves a classifier assessing feature distribution in each bag, correlating it with a unique class count label. This is done using Kernel Density Estimation (Parzen, 1962) to create feature probability distributions. This is then passed through a multi-layer perpecptrion (MLP) with dropout and mapped to number of unique classes. The goal is to determine the number of unique classes in unseen bags, optimized with Cross Entropy loss.

KDE is a technique used in the Unique Class Count (UCC) model for estimating the probability distribution of extracted features. Its key characteristic is providing a permutation-invariant property to the model. This means the order of instances in the input set does not affect the output, the ucc label.

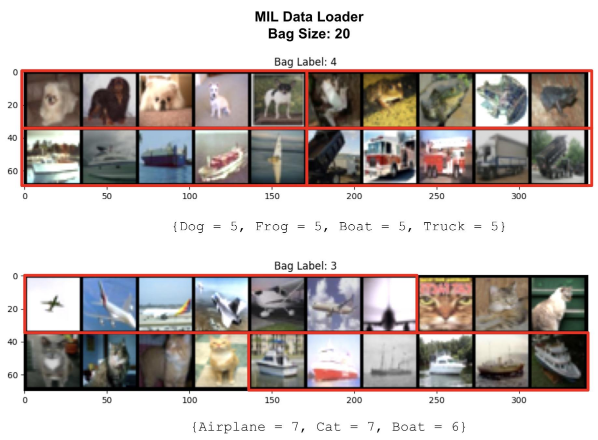
KDE is chosen for its ability to capture the entire shape of the feature distribution, rather than just point estimates like other pooling methods. This comprehensive view allows the model to better use the distribution shape for predictions. The Gaussian kernel used in KDE is differentiable, enabling seamless end-to-end model training.

The Gaussian kernel's bandwidth in KDE is a key parameter. It determines the smoothness of the estimated density. Correctly setting this bandwidth is crucial as it affects the model's ability to accurately capture feature distributions, which in turn influences the prediction of unique class counts.

Other MIL Approaches

In Multiple Instance Learning (MIL), pooling layers like max-pooling are commonly used to summarize features within a bag. However, these methods might miss out on the full distributional information. KDE, by estimating the entire probability distribution, offers a more detailed and accurate representation for weakly supervised clustering.

Implementation



```
{Dog = 5, Frog = 5, Boat = 5, Truck = 5}
```

```
{Airplane = 7, Cat = 7, Boat = 6}
```

FIG1: Bag-Level Image Aggregation

My implementation of the proposed algorithm is provided in PyTorch.

A customized data loader manages the preparation for a MIL downstream task. It generates all combinations for a lower and upper bounds of bag counts. It generates an equal distribution of image instances per class within the bag. Data augmentation is kept simple: random horizontal and vertical flips as well as normalization. This is because given the power law and dataset size, I hypothesize that there is sufficient examples. Color Jitter was initially used but later removed as it negatively impacted training.

The encoder-decoder implementation is simplified to a 3/3 layer CNN. For up- and down-sampling, it implements strided convolution instead of max pooling. I hypothesize that It offers a blend of feature extraction and dimensionality reduction in a single step, potentially leading to more efficient learning and sufficient performance for the CIFAR10 dataset.

Training was implemented using two approaches: one done using the Tensorboard and the Lightning.ai API and the other using a custom trainer class, both of which generated the same outputs.

Experiments

Config	`num_bins`	`sigma`	`dropout_rate`	`embedding_size`	`fc2_size`
A	10	0.1	0.3	128	256
B	20	0.05	0.5	256	128
C	5	0.2	0.2	64	512

FIG2: Table of Experiments for KDE

To explore the behaviour of the KDE module, I setup 3 experiments with variations to UCC hyperparameters. For other settings, they are kepy within an estimated "middle ground range". Bag classes are bounded to (1,4) inclusive. Bag sizes are arbitrarily kept constant at 50 and batch size at 10. AdamW is used as the optimizer and its learning rate is fixed at 0.0003 because <>. Distributed training is done on A1000 GPUs for 5000 steps.

The intentions between the configurations are as follows:

- Config A: The hypothesis here is that a moderate number of bins and a standard sigma value provide a good starting point for KDE. The dropout rate is set at a typical middle ground to prevent overfitting.
- Config B: I hypothesize that doubling the number of bins might capture more nuances in data distribution, and halving sigma could provide finer estimation. A higher dropout rate is chosen to combat potential overfitting due to the increased complexity from more bins.
- Config C: I hypothesize that with fewer bins and a larger sigma, the the model might generalize better by avoiding capturing noise in the data distribution. A lower dropout rate is tested due to the simpler model.

Analysis & Debugging Attempts

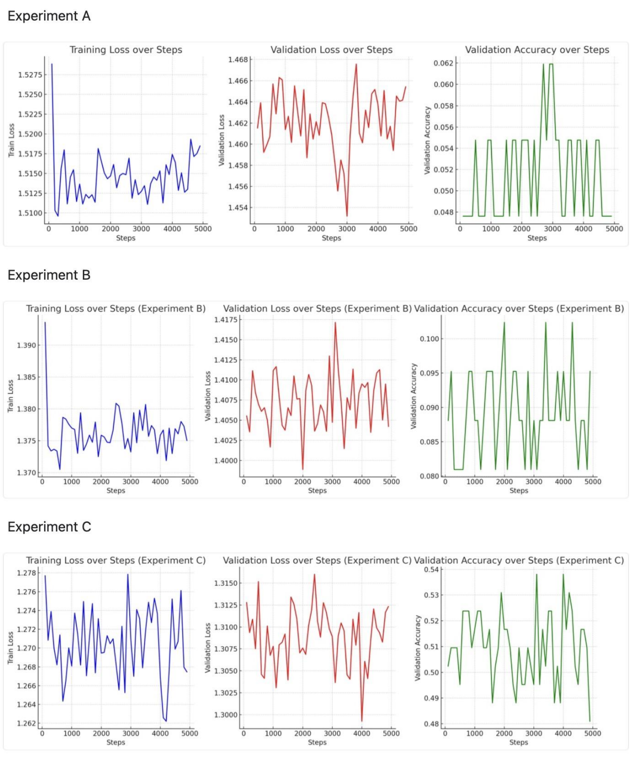


FIG3: Experiment Results

Overall the model failed to converge. Subequent profiling shows the loss of the UCC failing to converge.

The following notes detail the diagnostic attempts to address the issue:

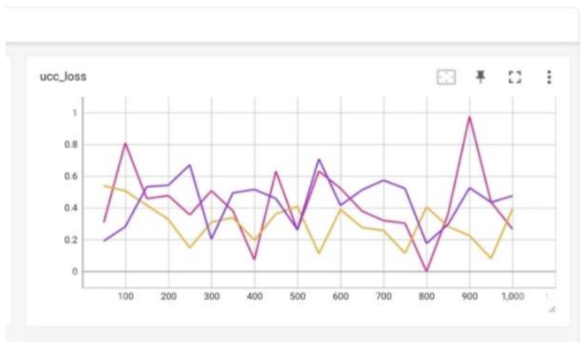


FIG4: UCC Loss over multiple diagnositc attempts

1. Alternate KDE (v3) Implementation: No observable changes were detected when an alternative KDE was implemented. The decision was made to revert to the original KDE.
2. Alternative MLP Architectures (v4-6): Substituting in MLPs of different capacities did

not yield any differences. I decided to revert to our MLP.

3. Learning Rate and Optimizer Adjustment (v6-9): Adjusting the learning rate and changing the optimizer produced no significant changes. The decision was made to continue with the AdamW optimizer and a learning rate of 0.0001.
4. Dataloader Tuning (v9-12): Tuning the dataset parameters between 10, 50 and 100 prevent shuffling did not affect the convergence.
5. Simplified Prediction Problem (v12-15): Simplifying the prediction problem to UCC = 1, 2 and other params did not result in better loss convergence.
6. Hyperparameter Tweaking (v16-18): No differences emerged from tweaking the hyperparameters. Current settings were maintained for the time being.
7. Biasing Alpha (v18, 19): Adjusting the weight of UCC loss did not result in any changes.

Each of these attempts was made with careful consideration and analysis, yet the UCC branch's loss has not shown the expected decrease indicative of convergence.

Nonetheless, using the “best” model C and training for extended steps, the following results were obtained:

Following the authors' recommendations, I evaluated the trained networks' performance on downstream tasks. Using the feature extractor, I conducted classification of UCC labels and K-means clustering on test data. The classification task assessed the distribution classifier's stability, while the clustering evaluated the feature extractor's capability to discern distinct features for each label.

UCC Model	*UCC Acc*	*Clustering Acc*
UCC	0.346	0.108
UCC^{2+}	0.312	0.067
$UCC_{alpha = 1}$	0.365	0.092
$UCC^{2+}_{alpha = 1}$	0.281	0.051

Improvements

I believe this is a novel application of KDE for feature extraction and it serves to strengthen the ucc classifier. For improvements, there are 3 areas that I believe can be considered:

Density Estimation methods

The utilization of Kernel Density Estimation (KDE) in feature extraction showcases a classical approach, yet exploring some alternative density estimation techniques could yield significant benefits. For instance, vector quantization methods central to Generative Adversarial Networks (GANs) offer a way to discretize the input space, potentially leading to more distinct feature representation. This method, through its ability to create "prototypical" representations of data, could enhance the distinctiveness of features extracted for each class in UCC.

Another alternative is the use of Variational Autoencoders (VAEs), which through their probabilistic nature, could provide a more nuanced understanding of the data distribution and improve the generalization of the classifier.

Unsupervised Feature Extraction (Embeddings)

Contrastive learning methods, which have shown great promise in unsupervised representation learning, could be an alternative to enhance feature extraction capabilities. By enforcing that representations of different augmentations of the same image are closer in the feature space than representations of different images, the model could learn more generalizable and discriminative features. This principle could be especially relevant for UCC classification, where distinguishing between complex and subtle variations is crucial.

Alternatively, Integrating masked auto-encoders, as suggested by recent advancements in self-supervised learning, such as BERT for text and MAE for images, represents an opportunity to push the boundaries of feature extraction. This approach could compel the model to reconstruct missing patches of the input image, thereby potentially focusing on the most salient features necessary for reconstruction. This aligns with the paper's goal of creating a strong feature extractor that can support a robust UCC classifier.

Signal Dilution in MIL from Power Set Explosion

The phenomenon of signal dilution in MIL is a critical challenge that arises when the informative signal is spread thinly across a large number of instances.

To combat this, one might consider adaptive bagging strategies that focus on creating bags that are more likely to contain informative instances.

Alternatively, Graph Neural Networks (GNNs) could offer a novel perspective on handling MIL problems. By treating bags as graphs, where instances are nodes and relationships between instances are edges, GNNs can leverage the relational information often present in MIL datasets. This approach could be particularly advantageous when the instances within a bag are not independent of each other, allowing the model to make predictions based on the collective properties of all instances within a bag.