Framework for Systematic Comparison of Hashing-based vs. Retrieval-based methods for Cross-modal Retrieval

Shivangi Bithel, Dwijesh Gohil, Prabhat Kanaujia

May 22, 2021

List of Figures

1 Introduction

We live in a multi-modal world, where we can express ourselves using a variety of representation formats like text, audio, images, etc. They can be used to represent different events happening around us. To search for data across these different modalities, we have cross-modal retrieval systems. Cross-modal retrieval is a natural way of information retrieval, where the input data is given in one modality or a combination of modalities, and output data can be retrieved in another or set of different modalities. For example, given textual data, "A beautiful sunny day on the beach" we can retrieve data related to this similar event in the form of an image or a video.

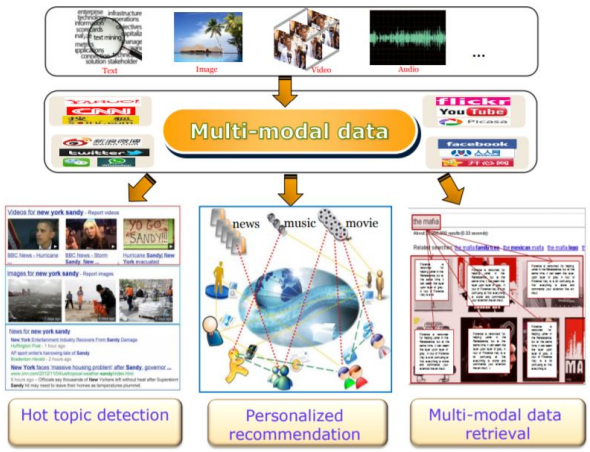

Figure 1: Multi-modal Data

Major Contributions of this work are briefly summarized below:

1. We are providing a systematic framework for comparing different cross-modal retrieval algorithms.

2. The framework evaluates two representative algorithms on the two commonly used datasets for cross-modal retrieval, namely Nuswide, XMediaNet and Wikipedia dataset.

2 Overview

Cross-Modal retrieval is an important problem in real-world applications like social media. Applications like Facebook, Twitter, Instagram get thousands of GBs of new data uploaded everyday which belongs to different modalities. To solve this problem, in the literature we have two kinds of algorithms, Hashing-based retrieval algorithms and Ranking-based retrieval algorithms. Hashing-based algorithms uses binary representation to speed up cross modal retrieval process. Data from different modalities is transformed to a common hamming space. We then perform cross modal similarity search in this common space by comparing the hamming distance. We use binary codes to represent data, which can lead to some loss in accuracy but it is time and space efficient. Retrieval-based ranking algorithms uses real-valued representation learning. Here, we learn a common representation space, where we can directly measure the similarity between data belonging to different modalities. These algorithms can be further categorized into supervised and unsupervised based on the methodology they use to learn the common space.

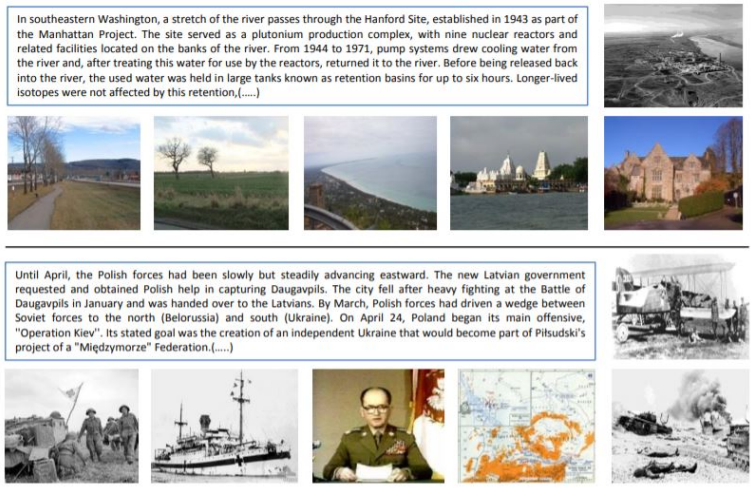

Figure 2: Cross-modal Retrieval

2.1 OCMFH

Online Collective Matrix Factorization Hashing is an unsupervised online learning method based on collective matrix factorization hashing (CMFH). It comes as an improvement over CMFH, and provides retrieval over large-scale data in a time and space efficient manner. As we have seen in example of Instagram, social media application, which already has millions of bytes of data, and every day a large chunk of new multimedia data is added to it. In such cases, training over complete data again and again is very time and space consuming. As a solution, OCMFH uses online learning method, which updates hash functions only on the new data chunk and also generates unified hash codes. It does not refer to the "orignal old data" while adaptively updating the hash codes. OCMFH also uses a zero mean normalization to overcome the mean varying problem in online hashing methods. It takes input, Image-Text pairs, hash-code length, parameters and iteration number and provides Hash codes, image and text projection matrices and their mean vectors. For querying, it uses feature vectors, projection matrices and mean vectors as input and compute the hamming distance between the generated hash codes, which is a bitwise operation and thus it is very fast and space efficient also.

2.2 ACMR

Adversarial Cross-Modal Retrieval is a ranking-based retrieval algorithm, which learns a common subspace to project data from different modalites. The goal of this algorithm is to find data item projections in a common feature representation subspace. ACMR combines the concept of supervised representation learning for cross-modal retrieval and adversarial learning. For input, ACMR takes image and text features,

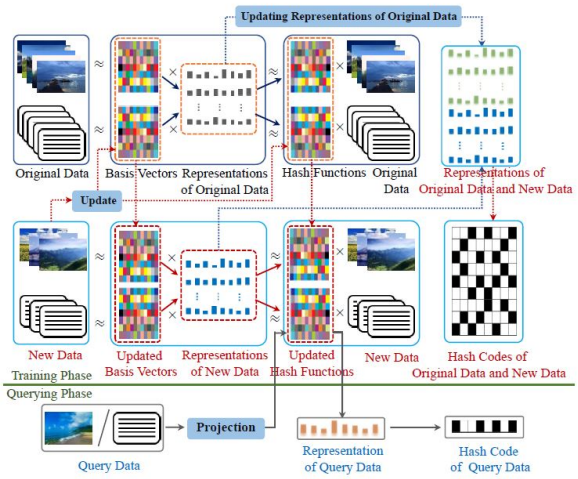

Figure 3: Framework for Online Collective Matrix Factorization Hashing

labels, hyperparameters, and samples in minibatch for each modality, and generates learned representation in common subspace. ACMR uses two processes, first is feature projector, which generates modality-invariant representation for items from different modalities in the common subspace. Its objective is to confuse the second process, which is a modality classifier. Modality classifier tries to differentiate between modalities and thus guide the feature projector to learn better representations which are invariant to the modality. The feature projector jointly performs label prediction so that underlying cross-modal semantic structure of the data can be preserved. Thus the feature representations are both invariant across the modalities and discriminative within a modality. For querying, we project input features in one modality and query features in other modality and calculate the distance between the two features. The features with higher similarity will have minimum distance even though they belong to different modalities.

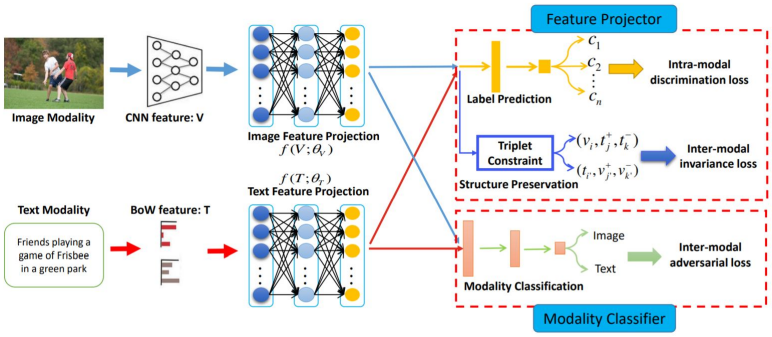

Figure 4: Flowchart of ACMR method

3 Dataset

3.1 Xmedianet

Xmedianet dataset can be downloaded from here. In the following experiments, we use only image and text features, though Xmedianet dataset comes with features of other modality as well(e.g. audio, video, etc.). Xmedianet dataset has 200 categories, 8000 test samples, and 32,000 train samples. We use this dataset without any pre-processing or filtering process.

3.2 NUS-WIDE

NUS-WIDE dataset can be downloaded from here. It is a multi-label dataset with image and text modality features. It has 81 ground truth categories but we use top-20 most frequent categories. We exclude any sample which does not have any ground truth category assigned. After such pre-processing, it has 83,644 train samples, 4402 test samples.
Another variant for the same dataset is being used in ACMR algorithm, which is NUS-WIDE 10K. This dataset can be downloaded from here. It uses 8000 training samples, 2000 testing samples and one sample only has one associated label, from the available 10 labels. Image feature vectors are 4096d, extracted using FC7 layer of VGGNet. Text samples are represented using 1000d Bag-of-words representation.

3.3 Wiki

Wikipedia dataset can be downloaded from here. In the ACMR algorithm, we are using this wikipedia dataset, which has 1300 training image-text pairs and 1566 testing image-text pairs. These are publically available 128d SIFT features for images and 10d LDA features. Only one label out of available 10 labels are assigned to the image-text pairs.

4 Framework

We provide a framework for end-to-end experiments and comparisons across models. The framework consists of 3 primary components:

4.1 Dataset

```
def __init__(self, directories,
                      loader,
                      preprocess=None,
                      preprocess_params=None,
                      normalize=None,
                      normalization_params=None,
                      read_directories=(True,True,True),
                      summarize=None
                      ):
```

Listing 1: Dataset initializer

Dataset class provides a wrapper for loading, preprocessing, normalizing and summarizing train, validation and test datasets. The format for storing, preprocessing, normalizing and summarizing the dataset is user-defined, as per the corresponding functions.
This modular approach allows the user to change a particular aspect of the dataset, while keeping others fixed, thus allowing for effective comparisons.

4.2 Model

```
def __init__(self,
             training_function,
             hyperparams,
             dataset_obj,
             params=None,
             params_verification=None,
             prediction_function=None,
             evaluation_metrics=None,
             is_neural=None
             ):
```

Listing 2: Model initializer

Model class also has a modular structure.

- The parameter - training_function, takes as an input a function. This function is core to the model class object. It has to be provided by the user and should point to a function which performs the training procedure for the concerned model.
- Hyperparams, params are user-defined objects and it is up to the user to ensure consistent-use in the training and predictions functions provided by them.
- dataset_obj is an object of Dataset class, as defined above

- evaluation_metrics are currently fixed and limited to Precision, Recall and MAP

This framework allows the user to change certain aspects of the model, such as hyperparameters, easily and run the experiments end-to-end. Result can be exported to a file and using the Comparator class defined later, a broad range of measures can be compared across multiple models.

An example of an experimentation routine (for OCMFH algorithm is detailed below):

```
dataset_filepath = '/mnt/f/mtp/dataset/dataset/xmedianet/'
#dataset_filepath = '/mnt/f/mtp/dataset/dataset/nus_wide/NUS-WIDE/'
data = Dataset((dataset_filepath, dataset_filepath, dataset_filepath),
        loaderXmedianet, read_directories=(True, False, True), summarize=
    summarizeDataset)
data.load_data()
hyperparams = {'bits':64, 'lambda_':0.5, 'mu':100, 'gamma':0.001, 'iter'
    :100, 'cmfhiter':100}
params = Parameters({'PI':None, 'PT':None, 'HH':None, 'B_Tr' : None, '
    B_Ir' : None, 'B_Te' : None, 'B_Ie' : None})
model = Model(  train,
        hyperparams,
        data,
        params,
        None, #params_verification
        predict, #prediction_function
        None, #evaluation_metrics
        False) #isneural
model.train_model()
model.predict('test')
model.evaluate(data.get_train_labels(), data.get_test_labels())
model.save_stats('ocmfh_xmedianet_stats_{}bits.npy'.format(hyperparams['
    bits']))
```

Listing 3: Model training

Stats for a particular experiment can be exported to a file, and such files can be used to compare models on various metrics.

4.3 Comparator

Comparator class takes a list of filenames as an input and loads and stores the corresponding stats. Following this, 2 kinds of plots are provided out-of-the-box:

- bar plots
- line plots

There are some restrictions regarding the structure of the stats_file, for these curves. Please refer to the comments and documentation for more info. Custom functions can be added on top of the comparator class, if needed, making the framework extensible. For examples of what kinds of plots one can expect, please refer to the next section. Sample code is also provided along with the Comparator class, in compare.py, to demonstrate how graphs can be plotted for multiple files and statistics.

5 Results

5.1 Loss History

Following are the loss vs iteration graphs. First plot compares ACMR[WYX+17] and OCMFH[WWA+20] on nuswide dataset. Remaining graphs are for OCMFH[WWA+20] algorithm for varying bit length on both nuswide and xmedianet dataset.

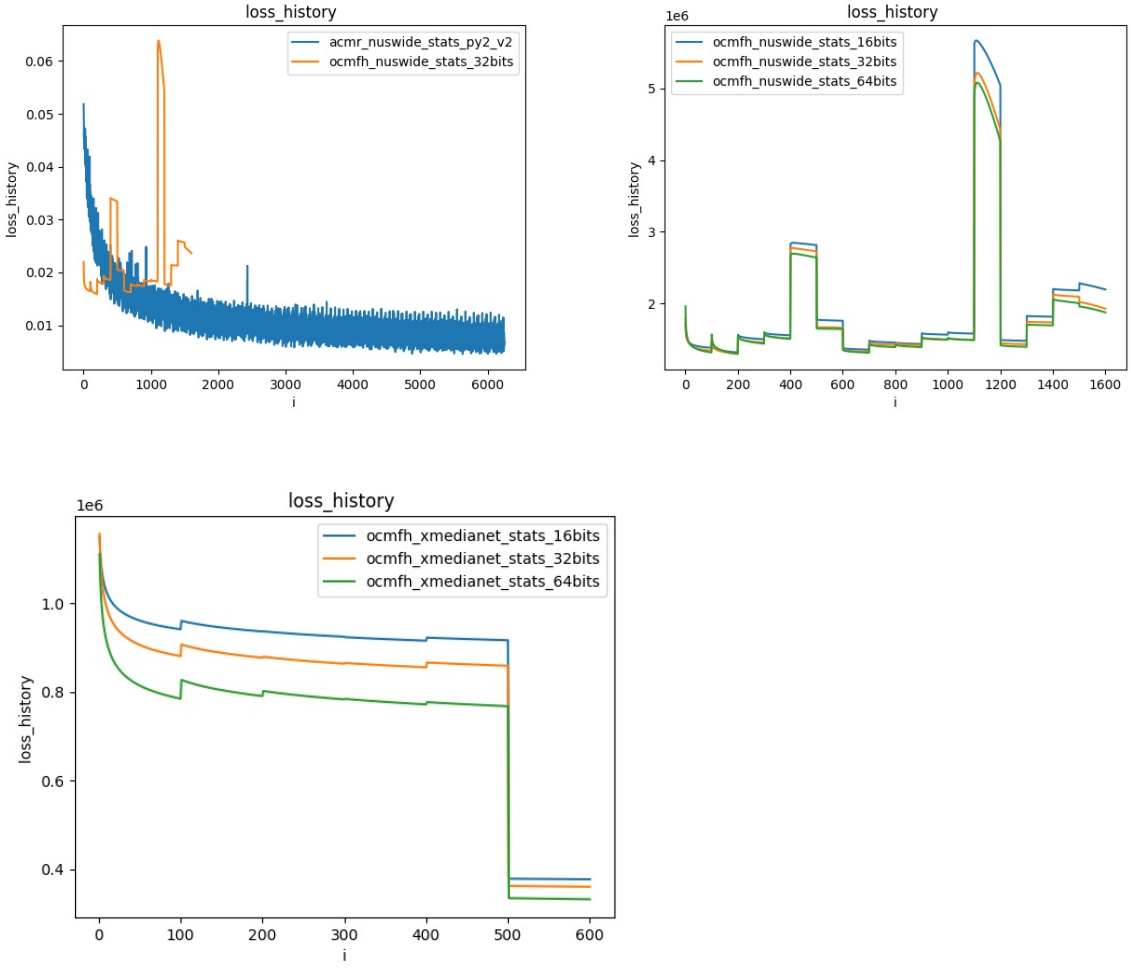

Figure 5: Loss vs iteration graph for inter and intra method comparison

5.2 Mean Average Precision

Following are the mean average precision vs i (mAP @ i) graph. First two graph are for OCMFH[WWA+20] on nuswide dataset for varying bit length. One graph for image to text(itot) and one graph for text to image(ttoi). Remaining two graphs compares ACMR[WYX+17] and OCMFH[WWA+20] on wiki and nuswide dataset respectively for both itot and ttoi task.

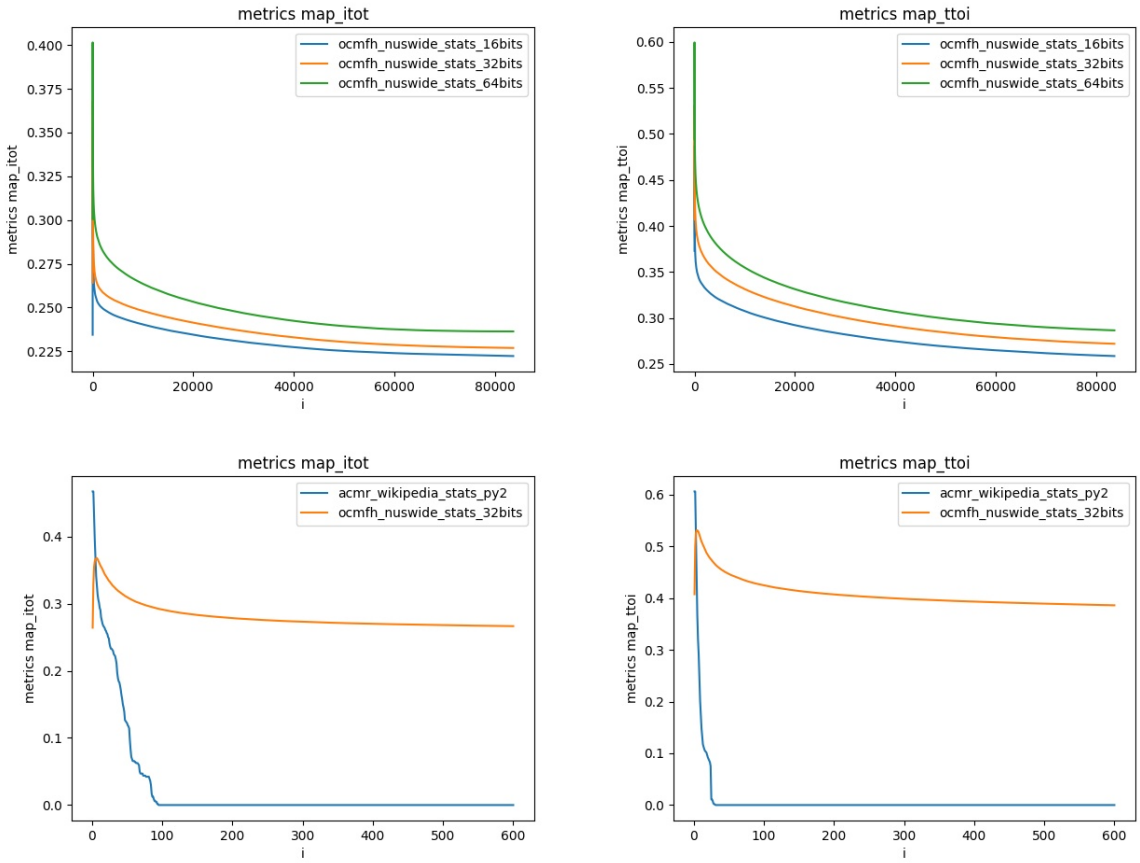

Figure 6: mAP metric for intra and inter method comparison on various datasets

5.3 Precision VS Recall

Following are the precision vs recall curves. All of them are for OCMFH[WWA+20] on both datasets(nuswide and xmedianet) for both tasks(itot and ttoi) with varying bit vector length.

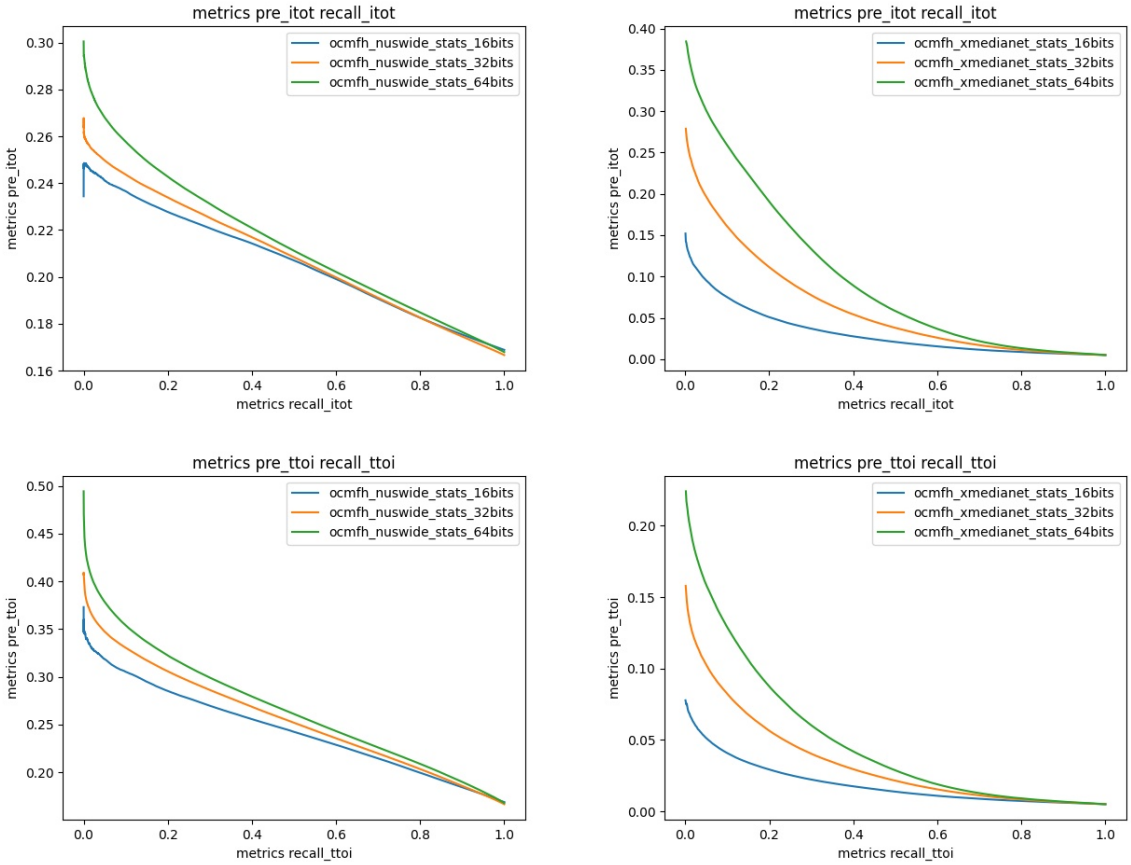

Figure 7: Precision vs Recall curve for OCMFH on various datasets

5.4 Precision

Following are the precision @ i curves. Graphs in the first column are for OCMFH[WWA$^+$20] on two different datasets with fixed bit vector length(16). Graphs in the second column compares the precision value on varying bit vector length for OCMFH[WWA$^+$20] on nuswide dataset.

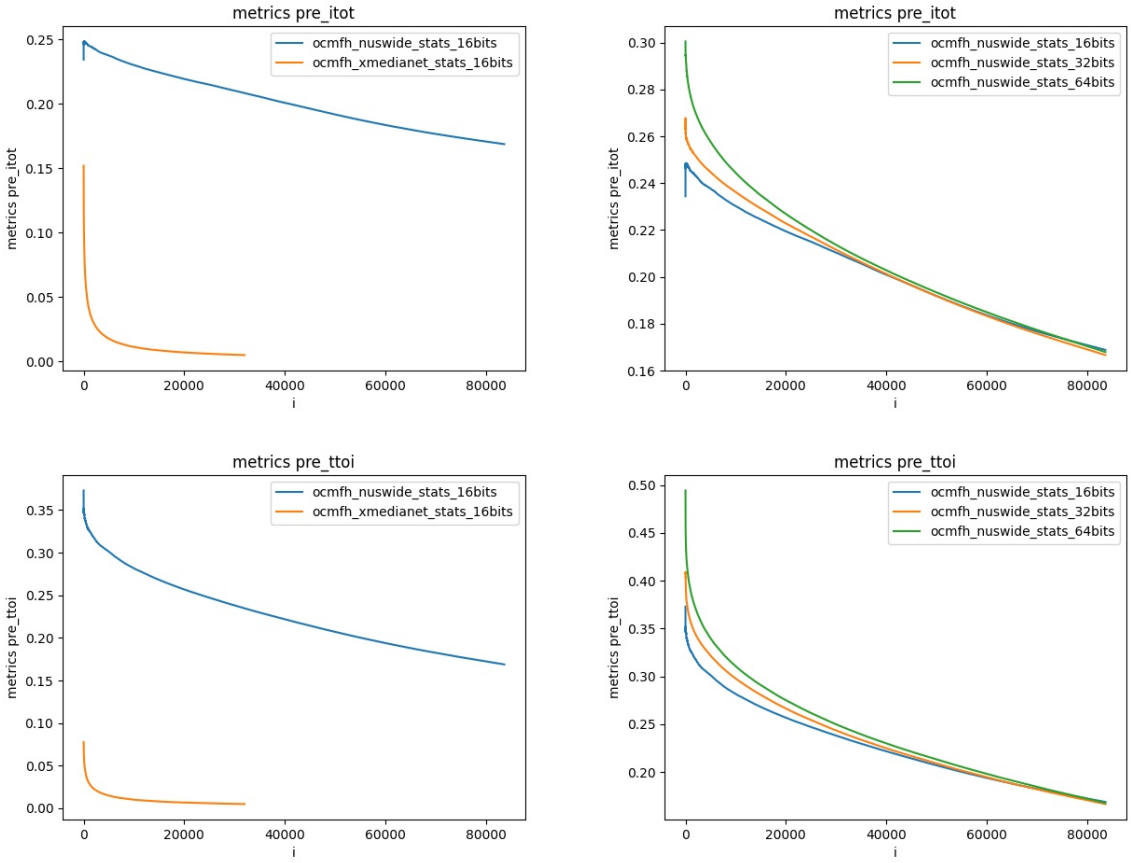

Figure 8: Precision metric for OCMFH on various datasets

5.5 Recall

Following are the recall @ i curves. Plots are shown for both ACMR[WYX+17] and OCMFH[WWA+20] on various datasets(nuswide, xmedianet, and wiki) with varying bit vector length.

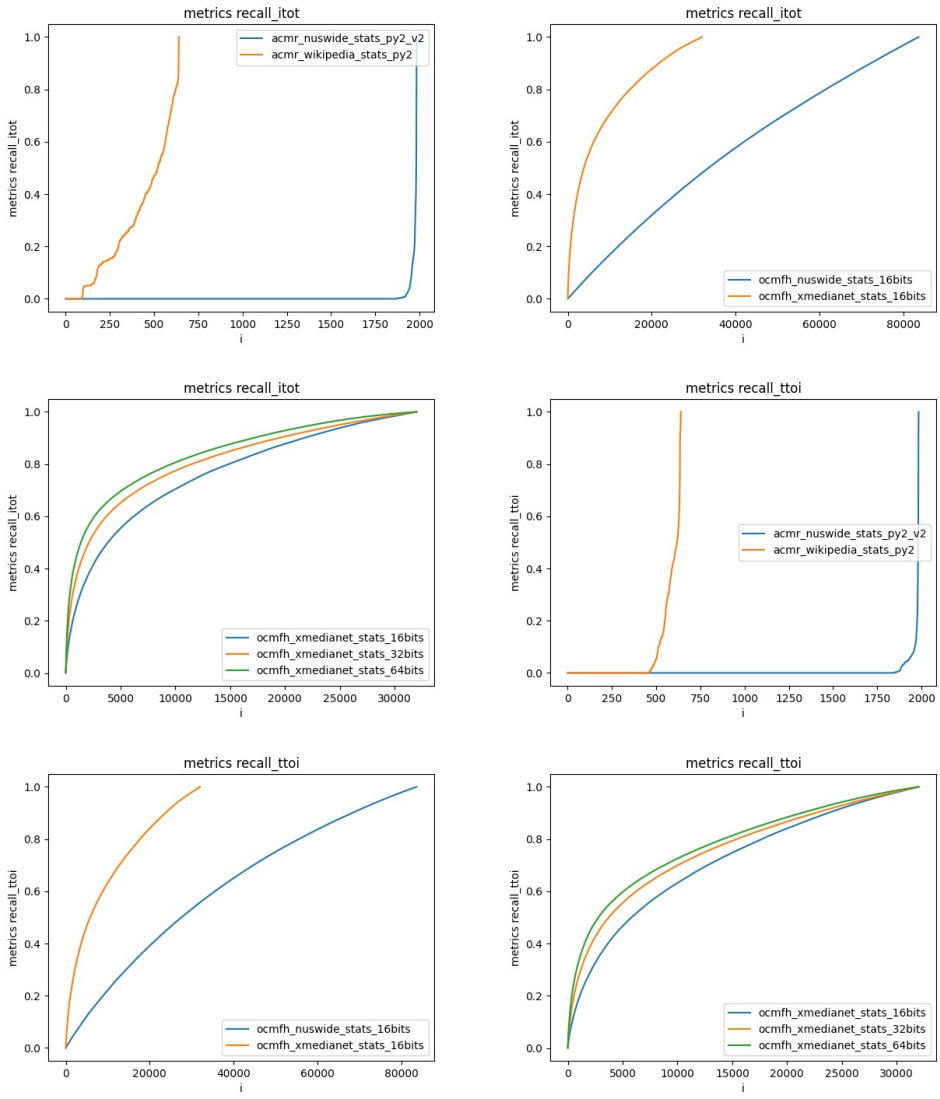

Figure 9: Recall metric for ACMR and OCMFH on various datasets

5.6 Time

Following are the bar plots for training and execution time in seconds. Plots in the first row are the prediction and training time of OCMFH[WWA$^+$20] on two datasets(nuswide and xmedianet) with fixed bit vector length(16). Plot in the second row compares OCMFH[WWA$^+$20] and ACMR training time on nuswide dataset.

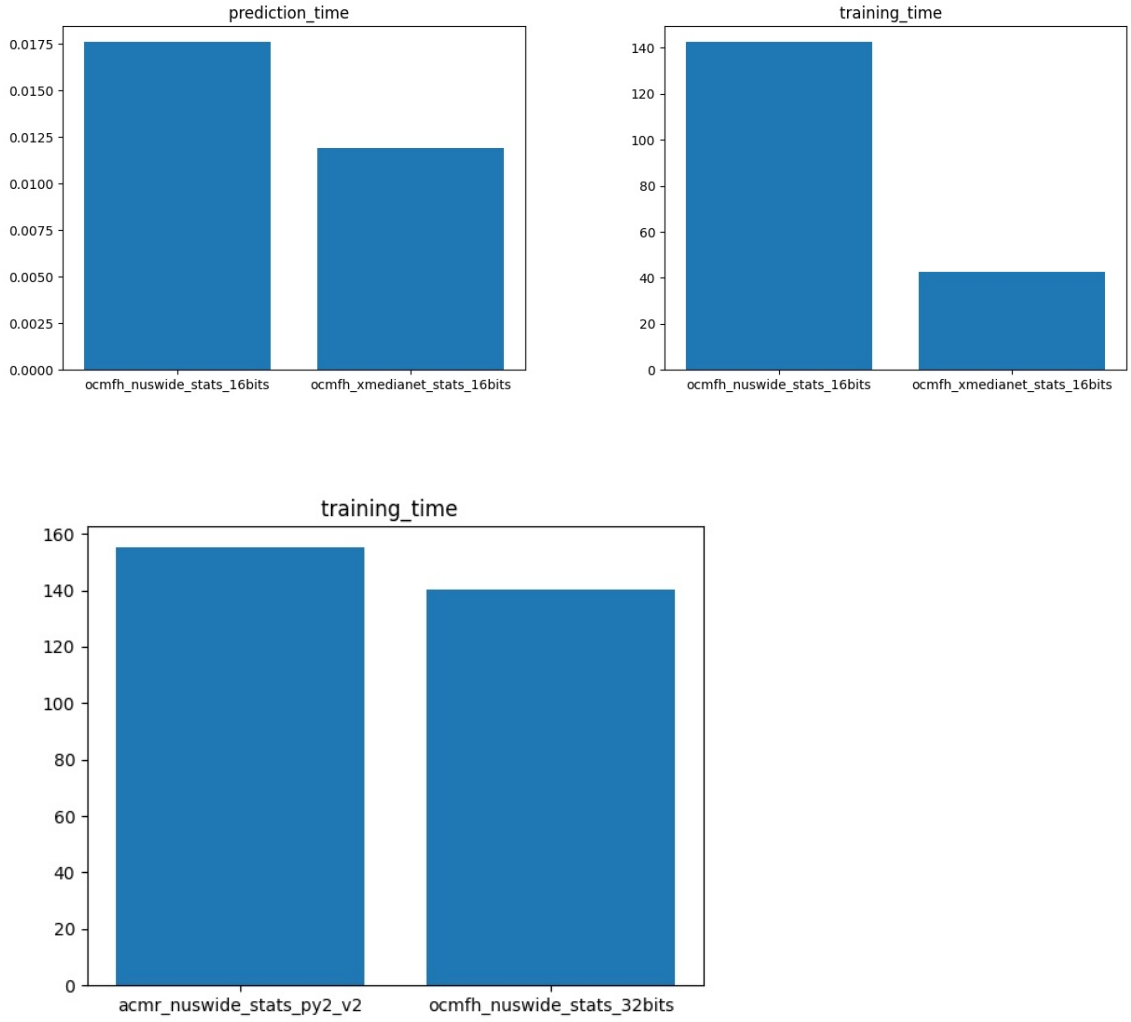

Figure 10: Training and prediction time for inter and intra method comparison on various datasets

5.7 Size

Following plots compares nuswide and xmedianet dataset statistics, e.g. number of classes and number of samples.

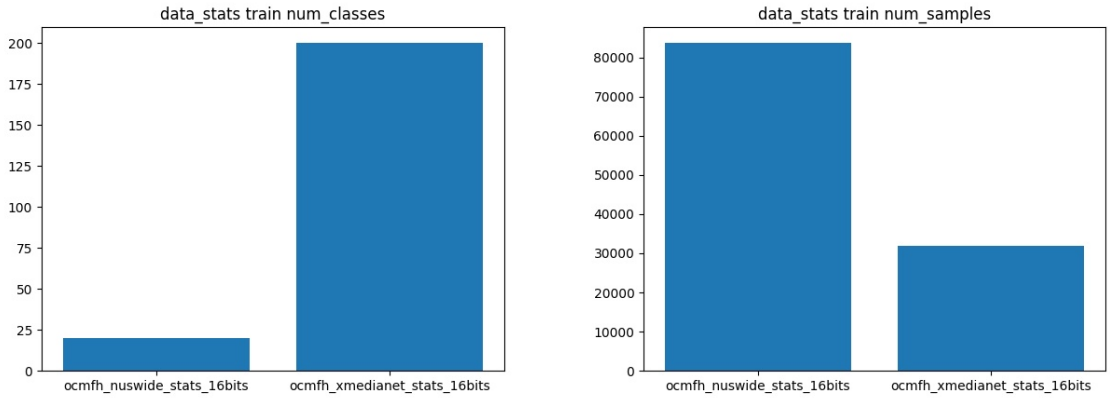

Figure 11: Dataset comparison in terms of #classes and #samples

6 Comparison (OCMFH)

In this section we compare the mAP score of OCMFH[WWA$^+$20] between our implementation and scores mentioned in the original paper. **Note:** The comparison is not fair. In the original paper, the authors use vgg16 features for images. Whereas, we use bag of word features for images. The train and test split sizes are also different.

	Task	32 bits	64 bits	128 bits
OCMFH[WWA$^+$20]	itot	0.8207	0.8455	0.8568
OCMFH(our)	itot	0.2269	0.2363	0.2329
OCMFH[WWA$^+$20]	ttoi	0.7675	0.7783	0.7863
OCMFH(our)	ttoi	0.2720	0.2869	0.2882

Table 1: mAP on NUS-WIDE dataset

Following can be the reasons behind such a large difference in the metric scores.

- They use only top-10 most frequent labels. We use top-20 most frequent labels.
- They use vgg16 features for image modality. We use simple bag-of-word features.

7 Links

- Xmedianet dataset , NUS-WIDE dataset
- Wiki , NUS-WIDE 10K
- Source Code
- Framework Documentation

References

[WWA+20] Di Wang, Quan Wang, Yaqiang An, Xinbo Gao, and Yumin Tian. Online collective matrix factorization hashing for large-scale cross-media retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1409–1418, New York, NY, USA, 2020. Association for Computing Machinery.

[WYX+17] Bokun Wang, Yang Yang, Xing Xu, Alan Hanjalic, and Heng Tao Shen. Adversarial cross-modal retrieval. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM '17, page 154–162, New York, NY, USA, 2017. Association for Computing Machinery.