Fast Image Retrieval

CiSIPLab

Aug 29, 2023

GET STARTED

1 Installation **1**

2 Prepare datasets **3**
- 2.1 CIFAR10 . . . 3
- 2.2 ImageNet100 . . . 3
- 2.3 NUS-WIDE . . . 3
- 2.4 MS-COCO . . . 4
- 2.5 MIRFlickr-25000 . . . 4
- 2.6 Stanford Online Product . . . 4
- 2.7 References . . . 4

3 Train a model **5**
- 3.1 Use the config file . . . 7

4 Test a Model **9**

5 Inference trained model **11**

6 Custom dataset **13**

7 Custom data pipeline/ training method **17**

8 Custom backbone network **19**

9 Custom hashing architecture **21**

10 Custom loss **23**

11 W&B logging **25**

12 Backbone **27**

13 Loss (Method) **29**
- 13.1 Supervised . . . 29
- 13.2 Unsupervised . . . 29
- 13.3 Shallow (Non-Deep learning methods) . . . 30

14 Datasets **31**

15 Changelog **33**
- 15.1 0.1.2 . . . 33
- 15.2 0.1.1 . . . 33

15.3 0.1.0 . . . 33

16 Run on Arm (aarch64) 35
16.1 Create conda environment and install pytorch . . . 35
16.2 Requirement to build fairs . . . 35
16.3 Build faiss for aarch64 . . . 36
16.4 Install dependencies . . . 36
16.5 Done . . . 36

17 Indices and tables 37

CHAPTER

ONE

INSTALLATION

We recommend to start in a new conda environment or a new PyTorch docker container. We will only include the guide for conda environment at the monent.

1. Clone the repository.

```
git clone <url>
cd fast-image-retrieval
```

2. Create new conda environment. Deactivate your current if you are in any other environment(e.g. `base` env).

```
conda deactivate
conda create -n fast-image-retrieval python=3.8 -y
conda activate fast-image-retrieval
```

3. Install the dependencies

```
pip install -r requirements.txt
# faiss gpu version, or `faiss-cpu` for cpu-only version
conda install -c pytorch faiss-gpu -y
```

Operating Systems

Currently only Linux is tested. `faiss-gpu` is not available for macOS.

CHAPTER

TWO

PREPARE DATASETS

You have to download the datasets you needed to their path respectively. Datasets are stored in `data/` folder. For example, ImageNet100 is stored at `data/imagenet`. If `data/` folder is not yet created, you may create it in the project root by:

```
mkdir data
```

Warning: We do not own these datasets, please read and agree to the original authors' license before download and use.

2.1 CIFAR10

There is no need of extra downloading for CIFAR10 dataset. The dataset will be downloaded automatically at the first time of running.

2.2 ImageNet100

This is a subset/variant of original ImageNet dataset, we do not own the dataset. Please head to ImageNet official website to read and accept their term of uses and license before download the dataset here (updated Aug 2022). This is a subset with most common 100 classes.

Download and extract the archive at `data/` directory:

```
# at <project_root>/data/
tar -xzf imagenet.tar.gz
```

2.3 NUS-WIDE

This is a subset/variant of original NUS-WIDE dataset, we do not own the dataset. Please head to NUS-WIDE official website to read and accept their term of uses and license before download the dataset here or here (updated Feb 2023). This is a subset with 21 most common classes.

Download and extract the archive at `data/` directory:

```
# at <project_root>/data/
tar -xzf nuswide_v2_256.tar.gz
```

2.4 MS-COCO

This is a subset/variant of original MS-COCO dataset, we do not own the dataset. Please head to MS-COCO official website to read and accept their term of uses and license before download the dataset here (updated Aug 2022).

Download and extract the archive at `data/` directory:

```
# at <project_root>/data/
tar -xzf coco.tar.gz
```

2.5 MIRFlickr-25000

This is a subset/variant of original MIRFLICKR dataset, we do not own the dataset. Please head to MIRFLICKR official website to read and accept their term of uses and license before download the dataset here (updated Aug 2022).

Download and extract the archive at `data/` directory:

```
# at <project_root>/data/
tar -xzf mirflickr.tar.gz
```

2.6 Stanford Online Product

This is a subset/variant of original Stanford Online Product dataset, we do not own the dataset. Please head to Stanford Online Product official website to read and accept their term of uses and license before download the dataset here (updated Aug 2022).

Download and extract the archive at `data/` directory:

```
# at <project_root>/data/
tar -xzf sop.tar.gz
```

2.7 References

We thank the following repository for the processed datasets:

1. https://github.com/TreezzZ/DSDH_PyTorch (imagenet100,nuswide)
2. https://github.com/thuml/HashNet/tree/master/pytorch (coco)

CHAPTER

THREE

TRAIN A MODEL

To train a model, the config(`--config`), dataset(`--ds`), and other parameters(see the usage below) has to be specified, or otherwise the default value will be used. We have included config file for most supported model in `configs/templates`.

```
usage: train.py [-h] [--config CONFIG] [--backbone BACKBONE]
                [--ds {imagenet100,nuswide,cifar10,imagenet50a,imagenet50b,cars,cifar10_
↪II,landmark,landmark200,landmark500,gldv2delgembed,roxford5kdelgembed,descriptor,sop,
↪sop_instance,food101,nuswide,coco,mirflickr}]
                [--dfolder DFOLDER] [--c10-ep {1,2}] [--ds-reset] [--usedb] [--train-
↪ratio TRAIN_RATIO] [--nbit NBIT] [--bs BS] [--maxbs MAXBS] [--epochs EPOCHS]
                [--arch ARCH] [--gpu-transform] [--gpu-mean-transform] [--no-aug] [--
↪resize-size RESIZE_SIZE] [--crop-size CROP_SIZE] [--R R]
                [--distance-func {hamming,cosine,euclidean}] [--zero-mean-eval] [--num-
↪worker NUM_WORKER] [--rand-aug]
                [--loss {greedyhash,jmlh,dpn,orthocos,ce,bihalf-supervised,orthoarc,sdhc,
↪csq,adsh,hashnet,dbdh,dpsh,mihash,sdh,dfh,dtsh,greedyhash-unsupervised,bihalf,ssdh,tbh,
↪itq,pca,lsh,sh,imh,cibhash}]
                [--tag TAG] [--seed SEED] [--optim {sgd,adam,rmsprop,adan}] [--loss-
↪params LOSS_PARAMS] [--device DEVICE] [--eval EVAL] [--lr LR] [--wd WD]
                [--step-size STEP_SIZE] [--lr-decay-rate LR_DECAY_RATE] [--scheduler␣
↪SCHEDULER] [--backbone-lr-scale BACKBONE_LR_SCALE] [--resume]
                [--resume-dir RESUME_DIR] [--enable-checkpoint] [--save-model] [--save-
↪best-model-only] [--discard-hash-outputs] [--load-from LOAD_FROM]
                [--benchmark] [--disable-tqdm] [--hash-bias] [--shuffle-database] [--
↪workers WORKERS] [--train-skip-preprocess] [--db-skip-preprocess]
                [--test-skip-preprocess] [--dataset-name-suffix DATASET_NAME_SUFFIX] [--
↪accimage] [--pin-memory] [--wandb]

optional arguments:
  -h, --help            show this help message and exit
  --config CONFIG       configuration file *.yml
  --backbone BACKBONE   the backbone feature extractor
  --ds {imagenet100,nuswide,cifar10,imagenet50a,imagenet50b,cars,cifar10_II,landmark,
↪landmark200,landmark500,gldv2delgembed,roxford5kdelgembed,descriptor,sop,sop_instance,
↪food101,nuswide,coco,mirflickr}
                        dataset
  --dfolder DFOLDER     data folder
  --c10-ep {1,2}        cifar10 evaluation protocol
  --ds-reset            whether to reset cifar10 txt
  --usedb               make all database images as training data
```

(continues on next page)

(continued from previous page)

```
  --train-ratio TRAIN_RATIO
                        training ratio (useful when usedb is activated)
  --nbit NBIT           number of bits for hash codes
  --bs BS               batch size
  --maxbs MAXBS         maximum batch size for testing, by default it is max(bs * 4,␣
↪maxbs)
  --epochs EPOCHS       training epochs
  --arch ARCH           architecture for the hash function
  --gpu-transform
  --gpu-mean-transform
  --no-aug              whether to skip augmentation
  --resize-size RESIZE_SIZE
                        Image Resize size before crop
  --crop-size CROP_SIZE
                        Image Crop size. Final image size.
  --R R                 if 0, using default R for specific dataset; -1 for mAP@All
  --distance-func {hamming,cosine,euclidean}
  --zero-mean-eval
  --num-worker NUM_WORKER
                        number of worker for dataloader
  --rand-aug            use random augmentation
  --loss {greedyhash,jmlh,dpn,orthocos,ce,bihalf-supervised,orthoarc,sdhc,csq,adsh,
↪hashnet,dbdh,dpsh,mihash,sdh,dfh,dtsh,greedyhash-unsupervised,bihalf,ssdh,tbh,itq,pca,
↪lsh,sh,imh,cibhash}
  --tag TAG
  --seed SEED
  --optim {sgd,adam,rmsprop,adan}
  --loss-params LOSS_PARAMS
  --device DEVICE       torch.device('?') cpu, cuda:x
  --eval EVAL           total evaluations throughout the training
  --lr LR               learning rate
  --wd WD               weight decay
  --step-size STEP_SIZE
                        relative step size (0~1)
  --lr-decay-rate LR_DECAY_RATE
                        decay rate for lr
  --scheduler SCHEDULER
                        LR Scheduler
  --backbone-lr-scale BACKBONE_LR_SCALE
                        Scale the learning rate of CNN backbone
  --resume
  --resume-dir RESUME_DIR
                        resume dir
  --enable-checkpoint
  --save-model
  --save-best-model-only
  --discard-hash-outputs
  --load-from LOAD_FROM
                        whether to load from a model
  --benchmark           Benchmark mode, determinitic, and no loss
  --disable-tqdm        disable tqdm for less verbose stderr
  --hash-bias           add bias to hash_fc
```

(continues on next page)

(continued from previous page)

```
  --shuffle-database    shuffle database during mAP evaluation
  --workers WORKERS     number of workers
  --train-skip-preprocess
  --db-skip-preprocess
  --test-skip-preprocess
  --dataset-name-suffix DATASET_NAME_SUFFIX
  --accimage            use accimage as backend
  --pin-memory          pin memory
  --wandb               enable wandb logging
```

3.1 Use the config file

You can create a new config file from the provided template. To specify argument(s), you may add `argument_key: argument_value` in the yaml config file. For loss parameters, you may add under the `loss_param` list. For example:

```yaml
loss: orthocos
loss_param:
  loss_param1: value
  loss_param2: value
new_argument: new_argument_value
```

Warning: Missing of loss params will cause error.

To start the training, you may run the training script at `train.py` with specify the config file and any additional arguments or otherwise the default value will be used. Do note that, the argument specify on the command line will override the value in config file.

```
python train.py --config configs/templates/orthocos.yaml --ds cifar10 --nbit 64 --epoch␣
↪100
```

Note: The priority of arguments is as follow: cli arguments > config file > default value.

Differ from the typical argument, to override the loss parameters, one must specify the argument `--loss-param` with their specific value seperated by colon and semicolon `"key1:value1;key2:value2"` to override the loss parameters(usually the hyper-parameters). For example:

```
--loss-param "m:0.4;s:8.0;cent_init:B"
```

Note: Some dataloader problem might occur when `--num-worker` is not set to 0 in Windows and macOS BigSur.

When the training task is launched, a log directory will be created. All the trained model(if `--save-model` specified), training loss, database and query codes, checkpoints(if enabled), etc will be stored in this directory. The directory will be created under `logs` directory with `<loss>_<backbone>_<arch>_<nbit>_<ds>_<epoch>_<lr>_<optim>/<order>_<tag>_<seed>` format, for example:

```
"logs/orthocos_alexnet_orthohash_64_cifar10_100_0.0001_adam/002_test_19084"
```

CHAPTER

FOUR

TEST A MODEL

Will be introduced soon.

CHAPTER

FIVE

INFERENCE TRAINED MODEL

We provide simple Web Interface for Inference that can be a good showcase or demo for trained model. This interface allows making simple inference on trained image retrieval model. By uploading a query image, it will return top k the nearest image from the database index. This web interface is created with Flask framework and is very easy to use.

To start with this inference

1. Run the following command on the root of project:

```
python inference.py --dir logs/orthocos_alexnet_orthohash_128_imagenet100_100_0.
↪0001_adam/002_test_0 --device cuda:0 --k 10 --ip 0.0.0.0 --port 8000
```

   Note that `--dir` is required field, `--device`, `--k`, `--ip` and `--port` is optional.

   Note: First run will be slow, this is because we had to create index for all images in database, and is recommend to run with GPU. A decent GPU took less than 10 minutes. After that, the index will be stored in the log directory for future use.

2. Head to `http://<your ip address>:<port>`. By default, it is at http://localhost:8000/
3. Drag an image to the white box area and click on `Upload File`.
4. Here you go.


CISiP-FIRe
localhost:8000
Image Retrieval - CISiP-FIRe
Choose or drag image
UPLOAD FILE
Distance: 0
Distance: 0
Distance: 1
Distance: 1
Distance: 1
Distance: 1
Distance: 1
Distance: 1
Distance: 1
Distance: 1

CHAPTER

SIX

CUSTOM DATASET

In `utils/datasets.py`, create a Dataset class for your custom dataset. Make sure return `image, class_id, and index_id` for each item.

```
from torch.utils.data.dataset import Dataset


class ImageDataset(Dataset):
    def __init__(self, ...):
        self.data = ..
        pass
    def __getitem__(self, index):
        return image, class_id, index
    def __len__(self):
        return len(data)
```

In the same file, create a dataset-specific method for your custom dataset. For example:

```
def new_dataset(**kwargs):
    transform = kwargs['transform']
    filename = kwargs['filename']
    suffix = kwargs.get('dataset_name_suffix', '')

    d = ImageDataset(f'data/imagenet{suffix}', transform=transform, filename=filename,␣
↪ratio=kwargs.get('ratio', 1))
    return d
```

In `configs.py:289`, specify your dataset name and link it to the dataset method you have created, for example:

```
datafunc = {
    'imagenet100': datasets.imagenet100,
    .
    .
    'new_dataset': datasets.new_dataset  # new dataset
}[dataset_name]
```

In the same `configs.py` file, at line `configs.py:53` specify the image size to resize. For example:

```
r = {
    'imagenet100': 256,
    .
    .
```

(continues on next page)

(continued from previous page)

```
    'new_dataset': 512 # new dataset resize size
}
```

At line `configs.py:80`, specify the crop size. For example:

```
r = {
    'imagenet100': 224,
    .
    .
    'new_dataset': 486 # new dataset crop size
}
```

Note: The image is first resize then only crop. However, it could depends on how you specify the augmentation for your dataset, which will be introduct later in this section.

At line `configs.py:108`, specify the number of class for your dataset. For example:

```
r = {
    'imagenet100': 100,
    .
    .
    'new_dataset': 10 # new dataset number of class
}
```

At line `configs.py:129`, specify the default number of image to retrieve per query (R). For example:

```
r = {
    'imagenet100': 100,
    .
    .
    'new_dataset': 10 # new dataset default R
}
```

At line `configs.py:281`, add the dataset name to the conditon as follow:

```
if dataset_name in ['imagenet100', 'nuswide', 'new_dataset']:
```

In `constants.py:datasets`, add the dataset name, for example:

```
datasets = {
    'class': ['imagenet100', 'new_dataset'],
    'multiclass': ['nuswide'],
}
```

Lastly, in the another file at line `utils/augmentations.py:19`, in the method `get_train_transform`, specify the augmentation. For example:

```
'nuswide': [
    transforms.Resize(resize),
    transforms.RandomCrop(crop),
    transforms.RandomHorizontalFlip()
],
```

(continues on next page)

(continued from previous page)

```
'new_dataset': [
    transforms.RandomResizedCrop(crop),
    transforms.RandomHorizontalFlip()
],
```

CHAPTER

SEVEN

CUSTOM DATA PIPELINE/ TRAINING METHOD

Will be introduced soon.

CHAPTER

EIGHT

CUSTOM BACKBONE NETWORK

In this framework, we have implemented 3 popular backbone network, which are AlexNet, VGG, and ResNet. We also leave the option to add other backbone network. This tutorial will give an example to implement a custom backbone network.

First, create a new python file `models/backbone/new_backbone.py`. Inside, create a new Backbone network extending the base class `BaseBackbone`. For example,

```
class NewAlexNetBackbone(BaseBackbone):
    def __init__(self, nbit, nclass, pretrained=False, freeze_weight=False, **kwargs):
        super(NewAlexNetBackbone, self).__init__()

        model = alexnet(pretrained=pretrained)
        self.features = model.features
        self.avgpool = model.avgpool
        fc = []
        for i in range(6):
            fc.append(model.classifier[i])
        self.fc = nn.Sequential(*fc)
        self.classifier = model.classifier[-1]

        self.in_features = model.classifier[6].in_features
        self.nbit = nbit
        self.nclass = nclass

        if freeze_weight:
            for param in self.features.parameters():
                param.requires_grad_(False)
            for param in self.fc.parameters():
                param.requires_grad_(False)

    def get_features_params(self): # this is required method
        return list(self.features.parameters()) + list(self.fc.parameters()) + list(self.
↪classifier.parameters())

    def get_hash_params(self): # this is required method
        raise NotImplementedError('no hash layer in backbone')

    def train(self, mode=True):
        super(NewAlexNetBackbone, self).train(mode)

        # all dropout set to eval
```

(continues on next page)

(continued from previous page)

```
        for mod in self.modules():
            if isinstance(mod, nn.Dropout):
                mod.eval()

    def forward(self, x): # this is required method
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

Secondly, in `models.architectures.helper.get_backbone` add the definition(s) of your new custom backbone, for example:

```
if backbone == 'alexnet':
    return AlexNetBackbone(nbit=nbit, nclass=nclass, pretrained=pretrained,
                           freeze_weight=freeze_weight, **kwargs)
if backbone == 'new_alexnet':
    return NewAlexNetBackbone(nbit=nbit, nclass=nclass, pretrained=pretrained,
                              freeze_weight=freeze_weight, **kwargs)
```

CHAPTER

NINE

CUSTOM HASHING ARCHITECTURE

New method might have a different architecture, our framework also allow custom architecture to be added. This tutorial will provide an example on how to add a new custom architecture.

Firstly, create a new python file at `models/architectures/arch_new.py`. Create a new architecture class extend the base class `BaseArch`. By extending this base class, the following attributes are accessible in the new architecture:

```
self.backbone_name = config['arch_kwargs']['backbone']
self.nbit = config['arch_kwargs']['nbit']
self.nclass = config['arch_kwargs']['nclass']
self.pretrained = config['arch_kwargs'].get('pretrained', True)
self.freeze_weight = config['arch_kwargs'].get('freeze_weight', False)
self.bias = config['arch_kwargs'].get('bias', False)
self.config = config
self.hash_kwargs = config['loss_param']
```

The new class should created with annotation `@register_network('new_arch_name')` and extending the base class `BaseArch` like the example below:

```
@register_network('dpn')
class ArchDPN(BaseArch):
    """Arch DPN, CSQ"""
    def __init__(self, config, **kwargs):
        super(ArchDPN, self).__init__(config, **kwargs)

        self.bias = config['arch_kwargs'].get('bias', False)

        self.backbone = get_backbone(backbone=self.backbone_name,
                                     nbit=self.nbit,
                                     nclass=self.nclass,
                                     pretrained=self.pretrained,
                                     freeze_weight=self.freeze_weight,
                                     **kwargs)
        self.ce_fc = nn.Linear(self.backbone.in_features, self.nclass)
        self.hash_fc = nn.Linear(self.backbone.in_features, self.nbit, bias=self.bias)

    def get_features_params(self): # this is a required method
        return self.backbone.get_features_params()

    def get_hash_params(self): # this is a required method
        return list(self.ce_fc.parameters()) + list(self.hash_fc.parameters())
```

(continues on next page)

(continued from previous page)

```
    def forward(self, x): # this is a required method
        x = self.backbone(x)
        u = self.ce_fc(x)
        v = self.hash_fc(x)
        return u, v
```

`get_features_params()` should return the list of trainable parameter from backbone network, while `get_hash_params` should return the list of trainable parameters in the hashing layers. `forward` method should be defined with your computation graph following your architecture.

Then, you should add the created architecture in the whitelist at `supported_model` dict in `constants.py:47`, for example:

```
supported_model = {
    'orthocos': ['orthohash', 'new_architecture'],
}
```

CHAPTER

TEN

CUSTOM LOSS

First, create a new python file `functions/loss/new_loss.py`. Create the loss class like the example below:

```
class NewLoss(nn.Module):
    def __init__(self, **kwargs):
        super(NewLoss, self).__init__()

        self.losses = {}  # required field

    def forward(self, x, h, b, labels, index, **kwargs):
        """
        x: features before hash layer
        h: output from hash FC
        b: binary code
        labels: not using (only use to obtain size)
        """

        loss1 = F.mse_loss(target_b, target_x)
        loss2 = ..
        loss = loss1+ loss2

        self.losses['mse'] = loss1  # to display and record the loss
        self.losses['loss2'] = loss2
        return loss
```

Note: The parameter for forward method will be different depends on type of the loss, the example above is for `unsupervised` method.

Method type	Loss params
supervised	self, logits, code_logits, labels, onehot=True
unsupervised	self, x, h, b, labels, index, **kwargs
pairwise	self, u, y, ind=None
adversarial	self, x, code_logits, rec_x, discs
shallow	self, x
contrastive	self, prob_i, prob_j, z_i, z_j

The parameter could also differ from the default, but that will be out of the scope of this example.

Secondly, at `scripts.train_helper.get_loss`, in `loss` dict, add the new loss name and link it to the loss class. For example:

```
loss = {
    'orthocos': OrthoCosLoss,
    'new_loss': NewLoss,
    ...
}
```

Then, at `losses` dict in `constants.py:28`, add the new loss name to the type of loss, for example, `new_loss` is a type of `supervised` loss/method:

```
losses = {
    'supervised': ['orthocos', ...,  'new_loss'],
    ...
}
```

Lastly, add the created loss and their supported architecture in the whitelist at `supported_model` dict in `constants.py:47`, for example:

```
supported_model = {
    'new_loss': ['orthohash', 'new_architecture'],
}
```

CHAPTER

ELEVEN

W&B LOGGING

To enable model logging using Weight and Bias, it can be enabled by the command line argument `--wandb`. You should also login your account first by `wandb login` if you haven't. Besides, you can also use `wandb init`.

```
python train.py ... --wandb
```

CHAPTER

TWELVE

BACKBONE

1. Alexnet
2. VGG{16}
3. ResNet{18,34,50,101,152}
4. ViT
5. Swim Transformer
6. ConvNext

CHAPTER

THIRTEEN

LOSS (METHOD)

13.1 Supervised

Method	Config Template	Loss Name	64bit ImageNet AlexNet (mAP@1K)
ADSH	*adsh.yaml*	adsh	0.645
BiHalf	*bihalf-supervised.yaml*	bihalf-supervised	0.684
Cross Entropy	*ce.yaml*	ce	0.434
CSQ	*csq.yaml*	csq	0.686
DFH	*dfh.yaml*	dfh	0.689
DPN	*dpn.yaml*	dpn	0.692
DPSH	*dpsh.yaml*	dpsh	0.599
DTSH	*dtsh.yaml*	dtsh	0.608
GreedyHash	*greedyhash.yaml*	greedyhash	0.667
HashNet	*hashnet.yaml*	hashnet	0.588
JMLH	*jmlh.yaml*	jmlh	0.664
OrthoCos(OrthoHash)	*orthocos.yaml*	orthocos	**0.701**
OrthoArc(OrthoHash)	*orthoarc.yaml*	orthoarc	0.698
SDH-C	*sdhc.yaml*	sdhc	0.639

13.2 Unsupervised

Method	Config Template	Loss Name	64bit ImageNet AlexNet (mAP@1K)
BiHalf	*bihalf.yaml*	bihalf	0.403
CIBHash	*cibhash.yaml*	cibhash	0.322
Greedy-Hash	*greedyhash-unsupervised.yaml*	greedyhash-unsupervised	0.407
SSDH	*ssdh.yaml*	ssdh	0.146
TBH	*tbh.yaml*	tbh	0.324

13.3 Shallow (Non-Deep learning methods)

Method	Config Template	Loss Name	64bit ImageNet AlexNet (mAP@1K)
IMH	*imh.yaml*	imh	0.467
ITQ	*itq.yaml*	itq	0.402
LsH	*lsh.yaml*	lsh	0.206
PCAHash	*pca.yaml*	pca	0.405
SH	*sh.yaml*	sh	0.350

Warning: Shallow methods only works with descriptor datasets. We will upload the descriptor datasets and

CHAPTER

FOURTEEN

DATASETS

Dataset	Name in framework
ImageNet100	imagenet100
NUS-WIDE	nuswide
MS-COCO	coco
MIRFLICKR/Flickr25k	mirflickr
Stanford Online Product	sop
Cars dataset	cars
CIFAR10	cifar10

CHAPTER

FIFTEEN

CHANGELOG

15.1 0.1.2

Add web inference interface. Now can do quick demo/testing on trained model. Fixed `wandb` being called when not enabled.

15.2 0.1.1

Add support for Weight and Bias logging

15.3 0.1.0

Initial release.

CHAPTER

SIXTEEN

RUN ON ARM (AARCH64)

These procedures were tested on Oracle ARM Ampere (aarch64) instance. We do not test on ARM device with CUDA device.

16.1 Create conda environment and install pytorch

```
conda craete -n fire python=3.9
conda activate fire
pip install numpy
pip install --pre torch torchvision torchaudio -f https://download.pytorch.org/whl/
↪nightly/cpu/torch_nightly.html
```

16.2 Requirement to build fairs

First, install prerequisites for building faiss.

```
sudo apt-get install libatlas-base-dev libatlas3-base
sudo apt-get install clang-8
sudo apt-get install swig
sudo apt-get install build-essential libblas-dev gfortran liblapack-dev
sudo apt -y install libopenblas-dev libblis-dev
sudo update-alternatives --set libblas.so.3-aarch64-linux-gnu \
    /usr/lib/aarch64-linux-gnu/blis-openmp/libblas.so.3
```

Then install latest cmake:

```
wget https://github.com/Kitware/CMake/releases/download/v3.22.1/cmake-3.22.1-linux-
↪aarch64.tar.gz
tar -xzf cmake-3.22.1-linux-aarch64.tar.gz
mv cmake-3.22.1-linux-aarch64 ~/cmake
alias cmake=~/cmake/bin/cmake
```

16.3 Build faiss for aarch64

```
# clone
git clone https://github.com/facebookresearch/faiss.git
cd faiss/
# build
cmake  -B build -DCMAKE_CXX_COMPILER=clang++-8 -DFAISS_ENABLE_GPU=OFF  -DPython_
↪EXECUTABLE=$(which python3) -DFAISS_OPT_LEVEL=generic -DCMAKE_BUILD_TYPE=Release -
↪DBUILD_TESTING=ON -DBLA_VENDOR=OpenBLAS
make -C build -j faiss
make -C build -j swigfaiss
# install python
(cd build/faiss/python && python setup.py install)
# install C headers (required for test)
sudo make -C build install
# testing
make -C build test
conda install pytest scipy numpy
PYTHONPATH="$(ls -d ./build/faiss/python/build/lib*/)" pytest tests/test_*.py
### note that test_binary might failed. But should be okay
make -C build demo_ivfpq_indexing
```

16.4 Install dependencies

Now install the dependencies for `cisip-FIRe`.

```
conda install scipy scikit-learn tqdm matplotlib pyyaml pandas seaborn flask psutil
sudo apt-get install ffmpeg libsm6 libxext6  -y
pip install pytorch_memlab opencv-python kornia wandb
```

16.5 Done

Now you can use the framework as usual.

CHAPTER

SEVENTEEN

INDICES AND TABLES

- genindex
- modindex
- search