MMRazor

Release 0.2.0

MMRazor Authors

Mar 08, 2022

开始你的第一步

CHAPTER

ONE

ENGLISH

CHAPTER TWO

简体中文

CHAPTER THREE

MMRAZOR.APIS

3.1 mmcls

`mmrazor.apis.mmcls.`**`set_random_seed`**(*seed*, *deterministic=False*)

Set random seed.

Parameters

- **`seed`** (*`int`*) –Seed to be used.
- **`deterministic`** (*`bool`*) –Whether to set the deterministic option for CUDNN backend, i.e., set `torch.backends.cudnn.deterministic` to True and `torch.backends.cudnn.benchmark` to False. Default: False.

`mmrazor.apis.mmcls.`**`train_model`**(*model*, *dataset*, *cfg*, *distributed=False*, *validate=False*, *timestamp=None*, *device='cuda'*, *meta=None*)

Copy from mmclassification and modify some codes.

This is an ugly implementation, and will be deprecated in the future. In the future, there will be only one train api and no longer distinguish between mmclassificaiton, mmsegmentation or mmdetection.

3.2 mmdet

3.3 mmseg

CHAPTER

FOUR

MMRAZOR.CORE

4.1 hooks

class mmrazor.core.hooks.**DistSamplerSeedHook**

Data-loading sampler for distributed training.

When distributed training, it is only useful in conjunction with `EpochBasedRunner`, while :obj:`IterBasedRunner` achieves the same purpose with `IterLoader`.

before_epoch(*runner*)

Executed in before_epoch stage.

class mmrazor.core.hooks.**DropPathProbHook**(*max_prob*, *interval=- 1*, *by_epoch=True*, ***kwargs*)

Set drop_path_prob periodically.

Parameters

- **max_prob** (*float*) –The max probability of dropping.
- **interval** (*int*) –The saving period. If `by_epoch=True`, interval indicates epochs, otherwise it indicates iterations. Default: -1, which means "never" .
- **by_epoch** (*bool*) –Saving checkpoints by epoch or by iteration. Default: True.

before_train_epoch(*runner*)

Executed in before_train_epoch stage.

class mmrazor.core.hooks.**SearchSubnetHook**(*interval=- 1*, *by_epoch=True*, *out_dir=None*, *max_keep_ckpts=- 1*, *save_last=True*, ***kwargs*)

Save checkpoints periodically.

Parameters

- **interval** (*int*) –The saving period. If `by_epoch=True`, interval indicates epochs, otherwise it indicates iterations. Default: -1, which means "never" .
- **by_epoch** (*bool*) –Saving checkpoints by epoch or by iteration. Default: True.

- **out_dir** (*str, optional*) –The directory to save checkpoints. If not specified, `runner.work_dir` will be used by default.
- **max_keep_ckpts** (*int, optional*) –The maximum checkpoints to keep. In some cases we want only the latest few checkpoints and would like to delete old ones to save the disk space. Default: -1, which means unlimited.
- **save_last** (*bool*) –Whether to force the last checkpoint to be saved regardless of interval.

after_train_epoch(*runner*)

Executed in after_train_epoch stage.

after_train_iter(*runner*)

Executed in after_train_iter stage.

before_run(*runner*)

Executed in before_run stage.

4.2 optimizer

`mmrazor.core.optimizer.`**build_optimizers**(*model*, *cfgs*)

Build multiple optimizers from configs. If *cfgs* contains several dicts for optimizers, then a dict for each constructed optimizers will be returned. If *cfgs* only contains one optimizer config, the constructed optimizer itself will be returned. For example, 1) Multiple optimizer configs: code-block:

```
optimizer_cfg = dict(
    model1=dict(type='SGD', lr=lr),
    model2=dict(type='SGD', lr=lr))
```

The return dict is `dict('model1': torch.optim.Optimizer, 'model2': torch.optim.Optimizer)` 2) Single optimizer config: .. code-block:

```
optimizer_cfg = dict(type='SGD', lr=lr)
```

The return is `torch.optim.Optimizer`. :param model: The model with parameters to be optimized. :type model: `nn.Module` :param cfgs: The config dict of the optimizer. :type cfgs: dict

Returns The initialized optimizers.

Return type dict[`torch.optim.Optimizer`] | `torch.optim.Optimizer`

4.3 runners

class `mmrazor.core.runners.`**`MultiLoaderEpochBasedRunner`**(*model*, *batch_processor=None*, *optimizer=None*, *work_dir=None*, *logger=None*, *meta=None*, *max_iters=None*, *max_epochs=None*)

Multi Dataloaders EpochBasedRunner.

There are three differences from EpochBaseRunner: 1) Support load data from multi dataloaders. 2) Support freeze some optimizer's lr update when runner has multi optimizers. 3) Add `search_subnet` api.

`register_lr_hook`(*lr_config*)

Resister a hook for setting learning rate.

Parameters **`lr_config`** (`dict`) –Config for setting learning rate.

`search_subnet`(*out_dir*, *filename_tmpl='epoch_{}.yaml'*, *create_symlink=True*)

Search the best subnet.

Parameters

- **`out_dir`** (`str`) –The directory that subnets are saved.
- **`filename_tmpl`** (`str, optional`) –The subnet filename template, which contains a placeholder for the epoch number. Defaults to 'epoch_{}.yaml'.
- **`create_symlink`** (`bool, optional`) –Whether to create a symlink "latest.yaml" to point to the latest subnet. Defaults to True.

`train`(*data_loader*, ***kwargs*)

Rewrite the `train` of `EpochBasedRunner`.

class `mmrazor.core.runners.`**`MultiLoaderIterBasedRunner`**(*model*, *batch_processor=None*, *optimizer=None*, *work_dir=None*, *logger=None*, *meta=None*, *max_iters=None*, *max_epochs=None*)

Multi Dataloaders IterBasedRunner.

There are three differences from IterBasedRunner 1) Support load data from multi dataloaders. 2) Support freeze some optimizer's lr update when runner has multi optimizers. 3) Add `search_subnet` api.

`register_lr_hook`(*lr_config*)

Resister a hook for setting learning rate.

Parameters **`lr_config`** (`dict`) –Config for setting learning rate.

`run`(*data_loaders*, *workflow*, *max_iters=None*, ***kwargs*)

Start running.

Parameters

- **data_loaders** (list[`DataLoader`]) –Dataloaders for training and validation.
- **workflow** (*list[tuple]*) –A list of (phase, iters) to specify the running order and iterations. E.g, [('train' , 10000), ('val' , 1000)] means running 10000 iterations for training and 1000 iterations for validation, iteratively.
- **max_iters** (*int*) –Specify the max iters.

search_subnet (*out_dir*, *filename_tmpl='epoch_{}.yaml'*, *create_symlink=True*)

Search the best subnet.

Parameters

- **out_dir** (*str*) –The directory that subnets are saved.
- **filename_tmpl** (*str, optional*) –The subnet filename template, which contains a placeholder for the epoch number. Defaults to 'epoch_{}.yaml' .
- **create_symlink** (*bool, optional*) –Whether to create a symlink "latest.yaml" to point to the latest subnet. Defaults to True.

4.4 searcher

class mmrazor.core.searcher.**EvolutionSearcher** (*algorithm*, *dataloader*, *test_fn*, *work_dir*, *logger*, *candidate_pool_size=50*, *candidate_top_k=10*, *constraints={'flops': 330000000.0}*, *metrics=None*, *metric_options=None*, *score_key='accuracy_top-1'*, *max_epoch=20*, *num_mutation=25*, *num_crossover=25*, *mutate_prob=0.1*, *resume_from=None*, ***search_kwargs*)

Implement of evolution search.

Parameters

- **algorithm** (`torch.nn.Module`) –Algorithm to be used.
- **dataloader** (*nn.Dataloader*) –Pytorch data loader.
- **test_fn** (*function*) –Test api to used for evaluation.
- **work_dir** (*str*) –Working direction is to save search result and log.
- **logger** (*logging.Logger*) –To log info in search stage.
- **candidate_pool_size** (*int*) –The length of candidate pool.
- **candidate_top_k** (*int*) –Specify top k candidates based on scores.

- **constraints** (*dict*) –Constraints to be used for screening candidates.
- **metrics** (*str*) –Metrics to be used for evaluating candidates.
- **metric_options** (*str*) –Options to be used for metrics.
- **score_key** (*str*) –To be used for specifying one metric from evaluation results.
- **max_epoch** (*int*) –Specify max epoch to end evolution search.
- **num_mutation** (*int*) –The number of candidates got by mutation.
- **num_crossover** (*int*) –The number of candidates got by crossover.
- **mutate_prob** (*float*) –The probability of mutation.
- **resume_from** (*str*) –Specify the path of saved .pkl file for resuming searching

check_constraints()

Check whether is beyond constraints.

Returns The result of checking.

Return type bool

search()

Execute the pipeline of evolution search.

update_top_k()

Update top k candidates.

class mmrazor.core.searcher.**GreedySearcher**(*algorithm*, *dataloader*, *target_flops*, *test_fn*, *work_dir*, *logger*, *max_channel_bins*, *min_channel_bins=1*, *metrics='accuracy'*, *metric_options=None*, *score_key='accuracy_top-1'*, *resume_from=None*, ***search_kwargs*)

Search with the greedy algorithm.

We start with the largest model and compare the network accuracy among the architectures where each layer is slimmed by one channel bin. We then greedily slim the layer with minimal performance drop. During the iterative slimming, we obtain optimized channel configurations under different resource constraints. We stop until reaching the strictest constraint (e.g., 200M FLOPs).

Parameters

- **algorithm** (torch.nn.Module) –Specific implemented algorithm based specific task in mmRazor, eg: AutoSlim.
- **dataloader** (torch.nn.Dataloader) –Pytorch data loader.
- **target_flops** (*list*) –The target flops of the searched models.
- **test_fn** (*callable*) –test a model with samples from a dataloader, and return the test results.

- **work_dir** (*str*) –Output result file.
- **logger** (*logging.Logger*) –To log info in search stage.
- **max_channel_bins** (*int*) –The maximum number of channel bins in each layer. Note that each layer is slimmed by one channel bin.
- **min_channel_bins** (*int*) –The minimum number of channel bins in each layer. Default to 1.
- **metrics** (*str | list[str]*) –Metrics to be evaluated. Default value is `accuracy`
- **metric_options** (*dict, optional*) –Options for calculating metrics. Allowed keys are ‘topk’, ‘thrs’ and ‘average_mode’. Defaults to None.
- **score_key** (*str*) –The metric to judge the performance of a model. Defaults to *accuracy_top-1*.
- **resume_from** (*str, optional*) –Specify the path of saved .pkl file for resuming searching. Defaults to None.

search()

Greedy Slimming.

4.5 utils

`mmrazor.core.utils.`**broadcast_object_list**(*object_list*, *src=0*)

Broadcasts picklable objects in `object_list` to the whole group.

Note that all objects in `object_list` must be picklable in order to be broadcasted.

Parameters

- **object_list** (*List[Any]*) –List of input objects to broadcast. Each object must be picklable. Only objects on the src rank will be broadcast, but each rank must provide lists of equal sizes.
- **src** (*int*) –Source rank from which to broadcast `object_list`.

`mmrazor.core.utils.`**set_lr**(*runner*, *lr_groups*, *freeze_optimizers=[]*)

Set specified learning rate in optimizer.

CHAPTER

FIVE

MMRAZOR.DATASETS

5.1 datasets

CHAPTER

SIX

MMRAZOR.MODELS

6.1 algorithms

class mmrazor.models.algorithms.**AlignMethodDistill**(***kwargs*)

class mmrazor.models.algorithms.**AutoSlim**(*num_sample_training=4*, *input_shape=(3, 224, 224)*, *bn_training_mode=False*, ***kwargs*)

AutoSlim: A one-shot architecture search for channel numbers.

Please refer to the *paper <https://arxiv.org/abs/1903.11728>* for details.

Parameters

- **num_sample_training** (*int*) –In each iteration we train the model at smallest width, largest width and (*num_sample_training* – 2) random widths. It should be no less than 2. Defaults to 4
- **input_shape** (*tuple*) –Input shape used for calculation the flops of the supernet.
- **bn_training_mode** (*bool*) –Whether set bn to training mode when model is set to eval mode. Note that in slimmable networks, accumulating different numbers of channels results in different feature means and variances, which further leads to inaccurate statistics of shared BN. Set `bn_training_mode` to True to use the feature means and variances in a batch.

get_subnet_flops()

A hacky way to get flops information of a subnet.

train(*mode=True*)

Overwrite the train method in `nn.Module` to set `nn.BatchNorm` to training mode when model is set to eval mode when `self.bn_training_mode` is `True`.

Parameters **mode** (*bool*) –whether to set training mode (`True`) or evaluation mode (`False`). Default: `True`.

train_step(*data*, *optimizer*)

Train step function.

This function implements the standard training iteration for autoslim pretraining and retraining.

Parameters

- **data** (*dict*) –Input data from dataloader.
- **optimizer** (torch.optim.Optimizer) –The optimizer to accumulate gradient

class mmrazor.models.algorithms.**Darts**(*unroll*, ***kwargs*)

train_step(*data*, *optimizer*)

The iteration step during training.

This method defines an iteration step during training, except for the back propagation and optimizer updating, which are done in an optimizer hook. Note that in some complicated cases or models, the whole process including back propagation and optimizer updating are also defined in this method, such as GAN.

Parameters

- **data** (*dict*) –The output of dataloader.
- **optimizer** (torch.optim.Optimizer | dict) –The optimizer of runner is passed to `train_step()`. This argument is unused and reserved.

Returns

It should contain at least 3 keys: loss, log_vars, `num_samples`. `loss` is a tensor for back propagation, which can be a weighted sum of multiple losses. `log_vars` contains all the variables to be sent to the logger. `num_samples` indicates the batch size (when the model is DDP, it means the batch size on each GPU), which is used for averaging the logs.

Return type dict

class mmrazor.models.algorithms.**DetNAS**(***kwargs*)

Implementation of DetNAS

class mmrazor.models.algorithms.**GeneralDistill**(*with_student_loss=True*, *with_teacher_loss=False*, ***kwargs*)

General Distillation Algorithm.

Parameters

- **with_student_loss** (*bool*) –Whether to use student loss. Defaults to True.
- **with_teacher_loss** (*bool*) –Whether to use teacher loss. Defaults to False.

class mmrazor.models.algorithms.**SPOS**(*input_shape=(3, 224, 224)*, *bn_training_mode=False*, ***kwargs*)

Implementation of SPOS

get_subnet_flops()

Get subnet’s flops based on the complexity information of supernet.

train(*mode=True*)

Overwrite the train method in *nn.Module* to set *nn.BatchNorm* to training mode when model is set to eval mode when *self.bn_training_mode* is *True*.

Parameters **mode** (`bool`) –whether to set training mode (*True*) or evaluation mode (*False*). Default: *True*.

train_step(*data*, *optimizer*)

The iteration step during training.

In retraining stage, to train subnet like common model. In pre-training stage, First to sample a subnet from supernet, then to train the subnet.

6.2 architectures

class `mmrazor.models.architectures.`**MMClsArchitecture**(***kwargs*)

Architecture based on MMCls.

cal_pseudo_loss(*pseudo_img*)

Used for executing `forward` with pseudo_img.

forward_dummy(*img*)

Used for calculating network flops.

class `mmrazor.models.architectures.`**MMDetArchitecture**(***kwargs*)

Architecture based on MMDet.

cal_pseudo_loss(*pseudo_img*)

Used for executing `forward` with pseudo_img.

class `mmrazor.models.architectures.`**MMSegArchitecture**(***kwargs*)

Architecture based on MMSeg.

6.3 distillers

class `mmrazor.models.distillers.`**SelfDistiller**(*components*, ***kwargs*)

Transfer knowledge inside a single model.

Parameters **components** (`dict`) –The details of the distillation. It usually includes the module names of the teacher and the student, and the losses used in the distillation.

compute_distill_loss(*data*)

Compute the distillation loss.

exec_student_forward(*student*, *data*)

Forward computation of the student.

Parameters

- **student** (torch.nn.Module) –The student model to be used in the distillation.
- **data** (*dict*) –The output of dataloader.

exec_teacher_forward(*teacher*, *data*)

Forward computation of the teacher.

Parameters

- **teacher** (torch.nn.Module) –The teacher model to be used in the distillation.
- **data** (*dict*) –The output of dataloader.

prepare_from_student(*student*)

Registers a global forward hook for each teacher module and student module to be used in the distillation.

Parameters **student** (torch.nn.Module) –The student model to be used in the distillation.

reset_outputs(*outputs*)

Reset the teacher' s outputs or student' s outputs.

student_forward_output_hook(*module*, *inputs*, *outputs*)

Save the output.

Parameters

- **module** (torch.nn.Module) –the module of register hook
- **inputs** (*tuple*) –input of module
- **outputs** (*tuple*) –out of module

teacher_forward_output_hook(*module*, *inputs*, *outputs*)

Save the output.

Parameters

- **module** (torch.nn.Module) –the module of register hook
- **inputs** (*tuple*) –input of module
- **outputs** (*tuple*) –out of module

class mmrazor.models.distillers.**SingleTeacherDistiller**(*teacher*, *teacher_trainable=False*, *teacher_norm_eval=True*, *components=()*, ***kwargs*)

Distiller with single teacher.

Parameters

- **teacher** (*dict*) –The config dict for teacher.
- **teacher_trainable** (*bool*) –Whether the teacher is trainable. Default: False.

- **teacher_norm_eval** (*bool*) –Whether to set teacher' s norm layers to eval mode, namely, freeze running stats (mean and var). Note: Effect on Batch Norm and its variants only. Default: True.
- **components** (*dict*) –The details of the distillation. It usually includes the module names of the teacher and the student, and the losses used in the distillation.

build_align_module(*cfg*)

Build `align_module` from the *cfg*.

`align_module` is needed when the number of channels output by the teacher module is not equal to that of the student module, or for some other reasons.

Parameters **cfg** (*dict*) –The config dict for `align_module`.

build_teacher(*cfg*)

Build a model from the *cfg*.

compute_distill_loss(*data=None*)

Compute the distillation loss.

exec_student_forward(*student*, *data*)

Execute the teacher' s forward function.

After this function, the student' s featuremaps will be saved in `student_outputs`.

exec_teacher_forward(*data*)

Execute the teacher' s forward function.

After this function, the teacher' s featuremaps will be saved in `teacher_outputs`.

get_teacher_outputs(*teacher_module_name*)

Get the outputs according module name.

prepare_from_student(*student*)

Registers a global forward hook for each teacher module and student module to be used in the distillation.

Parameters **student** (`torch.nn.Module`) –The student model to be used in the distillation.

reset_outputs(*outputs*)

Reset the teacher' s outputs or student' s outputs.

student_forward_output_hook(*module*, *inputs*, *outputs*)

Save the module' s forward output.

Parameters

- **module** (`torch.nn.Module`) –The module to register hook.
- **inputs** (*tuple*) –The input of the module.
- **outputs** (*tuple*) –The output of the module.

teacher_forward_output_hook (*module*, *inputs*, *outputs*)

Save the module' s forward output.

Parameters

- **module** (`torch.nn.Module`) –The module to register hook.
- **inputs** (*tuple*) –The input of the module.
- **outputs** (*tuple*) –The output of the module.

train (*mode=True*)

Set distiller' s forward mode.

6.4 losses

class mmrazor.models.losses.**ChannelWiseDivergence** (*tau=1.0*, *loss_weight=1.0*)

PyTorch version of `Channel-wise Distillation for Semantic Segmentation.

<https://arxiv.org/abs/2011.13256>`_.

Parameters

- **tau** (*float*) –Temperature coefficient. Defaults to 1.0.
- **loss_weight** (*float*) –Weight of loss. Defaults to 1.0.

forward (*preds_S*, *preds_T*)

Forward computation.

Parameters

- **preds_S** (*torch.Tensor*) –The student model prediction with shape (N, C, H, W).
- **preds_T** (*torch.Tensor*) –The teacher model prediction with shape (N, C, H, W).

Returns The calculated loss value.

Return type torch.Tensor

class mmrazor.models.losses.**KLDivergence** (*tau=1.0*, *reduction='batchmean'*, *loss_weight=1.0*)

A measure of how one probability distribution Q is different from a second, reference probability distribution P.

Parameters

- **tau** (*float*) –Temperature coefficient. Defaults to 1.0.
- **reduction** (*str*) –Specifies the reduction to apply to the loss: `'none'` | `'batchmean'` | `'sum'` | `'mean'`. `'none'`: no reduction will be applied, `'batchmean'`: the sum of the output will be divided by

  the batchsize,

`'sum'`: the output will be summed, `'mean'`: the output will be divided by the number of elements in the output.

Default: `'batchmean'`

- **`loss_weight`** (*`float`*) –Weight of loss. Defaults to 1.0.

`forward`(*preds_S*, *preds_T*)

Forward computation.

Parameters

- **`preds_S`** (*`torch.Tensor`*) –The student model prediction with shape (N, C, H, W) or shape (N, C).
- **`preds_T`** (*`torch.Tensor`*) –The teacher model prediction with shape (N, C, H, W) or shape (N, C).

Returns The calculated loss value.

Return type torch.Tensor

`class` `mmrazor.models.losses.`**`WSLD`**(*tau=1.0*, *loss_weight=1.0*, *num_classes=1000*)

PyTorch version of Rethinking Soft Labels for Knowledge Distillation: A Bias-Variance Tradeoff Perspective.

Parameters

- **`tau`** (*`float`*) –Temperature coefficient. Defaults to 1.0.
- **`loss_weight`** (*`float`*) –Weight of loss. Defaults to 1.0.
- **`num_classes`** (*`int`*) –Defaults to 1000.

`forward`(*student*, *teacher*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

6.5 mutables

class `mmrazor.models.mutables.`**`DifferentiableEdge`**(*with_arch_param*, ***kwargs*)

Differentiable Edge.

Search the best module from choices by learnable parameters.

Parameters **`with_arch_param`** (*`bool`*) –whether build learable architecture parameters.

`build_arch_param`()

build learnable architecture parameters.

`compute_arch_probs`(*arch_param*)

compute chosen probs according architecture parameters.

`forward`(*prev_inputs*, *arch_param=None*)

forward function.

In some algorithms, there are several `MutableModule` share the same architecture parameters. So the architecture parameters are passed in as args.

Parameters

- **`prev_inputs`** (*`list[torch.Tensor]`*) –each choice’ s inputs.
- **`arch_param`** (*`torch.nn.Parameter`*) –architecture parameters.

class `mmrazor.models.mutables.`**`DifferentiableOP`**(*with_arch_param*, ***kwargs*)

Differentiable OP.

Search the best module from choices by learnable parameters.

Parameters **`with_arch_param`** (*`bool`*) –whether build learable architecture parameters.

`build_arch_param`()

build learnable architecture parameters.

`compute_arch_probs`(*arch_param*)

compute chosen probs according architecture parameters.

`forward`(*x*, *arch_param=None*)

forward function.

In some algorithms, there are several `MutableModule` share the same architecture parameters. So the architecture parameters are passed in as args.

Parameters

- **`prev_inputs`** (*`list[torch.Tensor]`*) –each choice’ s inputs.
- **`arch_param`** (*`torch.nn.Parameter`*) –architecture parameters.

class mmrazor.models.mutables.**GumbelEdge**(***kwargs*)

Gumbel Edge.

Search the best module from choices by gumbel trick.

compute_arch_probs(*arch_param*)

compute chosen probs by gumbel trick.

set_temperature(*temperature*)

Modify the temperature.

class mmrazor.models.mutables.**GumbelOP**(*tau=1.0*, *hard=True*, ***kwargs*)

Gumbel OP.

Search the best module from choices by gumbel trick.

compute_arch_probs(*arch_param*)

compute chosen probs by gumbel trick.

set_temperature(*temperature*)

Modify the temperature.

class mmrazor.models.mutables.**MutableEdge**(*choices*, ***kwargs*)

Mutable Edge. In some NAS algorithms (Darts, AutoDeeplab, etc.), the connections between modules are searchable, such as the connections between a node and its previous nodes in Darts. `MutableEdge` has N modules to process N inputs respectively.

Parameters **choices** (*torch.nn.ModuleDict*) –Unlike `MutableOP`, there are already created modules in choices.

build_choices(*cfg*)

MutableEdge’s choices is already built.

class mmrazor.models.mutables.**MutableModule**(*space_id*, *num_chosen=1*, *init_cfg=None*, ***kwargs*)

Base class for `MUTABLES`. Searchable module for building searchable architecture in NAS. It mainly consists of module and mask, and achieving searchable function by handling mask.

Parameters

- **space_id** (*str*) –Used to index `Placeholder`, it is one and only index for each `Placeholder`.
- **num_chosen** (*str*) –The number of chosen `OPS` in the `MUTABLES`.
- **init_cfg** (*dict*) –Init config for `BaseModule`.

build_choice_mask()

Generate the choice mask for the choices of `MUTABLES`.

Returns Init choice mask. Its elements’ type is bool.

Return type torch.Tensor

abstract build_choices(*cfg*)

Build all chosen `OPS` used to combine `MUTABLES`, and the choices will be sampled.

Parameters **cfg** (*dict*) –The config for the choices.

build_space_mask()

Generate the space mask for the search spaces of `MUTATORS`.

Returns Init choice mask. Its elements' type is float.

Return type torch.Tensor

property choice_modules

The choices' modules.

Returns The values of the choices.

Return type tuple

property choice_names

The choices' names.

Returns The keys of the choices.

Return type tuple

export(*chosen*)

Delete not chosen `OPS` in the choices.

Parameters **chosen** (*list[str]*) –Names of chosen `OPS`.

abstract forward(*x*)

Forward computation.

Parameters **x** (*tensor | tuple[tensor]*) –x could be a Torch.tensor or a tuple of Torch.tensor, containing input data for forward computation.

property num_choices

The number of the choices.

Returns the length of the choices.

Return type int

set_choice_mask(*mask*)

Use the mask to update the choice mask.

Parameters **mask** (*torch.Tensor*) –Choice mask specified to update the choice mask.

class mmrazor.models.mutables.**MutableOP**(*choices*, *choice_args*, ***kwargs*)

An important type of `MUTABLES`, inherits from `MutableModule`.

Parameters

- **choices** (*dict*) –The configs for the choices, the chosen `OPS` used to combine `MUTABLES`.

- **choice_args** (*dict*) –The args used to set chosen `OPS`.

build_choices(*cfgs*, *choice_args*)

Build all chosen `OPS` used to combine `MUTABLES`, and the choices will be sampled.

Parameters

- **cfgs** (*dict*) –The configs for the choices.
- **choice_args** (*dict*) –The args used to set chosen `OPS`.

Returns Consists of chosen `OPS` in the arg *cfgs*.

Return type torch.nn.ModuleDict

class mmrazor.models.mutables.**OneShotOP**(***kwargs*)

A type of `MUTABLES` for the one-shot NAS.

forward(*x*)

Forward computation for chosen `OPS`, in one-shot NAS, the number of chosen `OPS` can only be one.

Parameters x (*tensor | tuple[tensor]*) –x could be a Torch.tensor or a tuple of Torch.tensor, containing input data for forward computation.

Returns The result of forward.

Return type torch.Tensor

6.6 mutators

class mmrazor.models.mutators.**DartsMutator**(*ignore_choices=('zero')*, ***kwargs*)

class mmrazor.models.mutators.**DifferentiableMutator**(***kwargs*)

A mutator for the differentiable NAS, which mainly provide some core functions of changing the structure of `ARCHITECTURES`.

build_arch_params(*supernet*)

This function will build many arch params, which are generally used in diffrentiale search algorithms, such as Darts' series. Each space_id corresponds to an arch param, so the Mutable with the same space_id share the same arch param.

Parameters supernet (`torch.nn.Module`) –The architecture to be used in your algorithm.

Returns

the arch params are got after traversing the supernet.

Return type torch.nn.ParameterDict

modify_supernet_forward(*supernet*)

Modify the supernet's default value in forward. Traverse all child modules of the model, modify the supernet's default value in :func:'forward' of each Space.

Parameters **`supernet`** (`torch.nn.Module`) –The architecture to be used in your algorithm.

`prepare_from_supernet` (*supernet*)

Inherit from `BaseMutator`' s, execute some customized functions exclude implementing origin `prepare_from_supernet`.

Parameters **`supernet`** (`torch.nn.Module`) –The architecture to be used in your algorithm.

`class` `mmrazor.models.mutators.`**`OneShotMutator`** (***kwargs*)

A mutator for the one-shot NAS, which mainly provide some core functions of changing the structure of `ARCHITECTURES`.

`static crossover` (*subnet_dict1*, *subnet_dict2*)

Crossover used in evolution search.

Parameters

- **`subnet_dict1`** (*`dict`*) –Record the information to build the subnet from the supernet, its keys are the properties `space_id` of placeholders in the mutator' s search spaces, its values are masks.
- **`subnet_dict2`** (*`dict`*) –Record the information to build the subnet from the supernet, its keys are the properties `space_id` of placeholders in the mutator' s search spaces, its values are masks.

Returns A new subnet_dict after crossover.

Return type dict

`static get_random_mask` (*space_info*, *searching*)

Generate random mask for randomly sampling.

Parameters

- **`space_info`** (*`dict`*) –Record the information of the space need to sample.
- **`searching`** (*`bool`*) –Whether is in search stage.

Returns Random mask generated.

Return type torch.Tensor

`mutation` (*subnet_dict*, *prob=0.1*)

Mutation used in evolution search.

Parameters

- **`subnet_dict`** (*`dict`*) –Record the information to build the subnet from the supernet, its keys are the properties `space_id` of placeholders in the mutator' s search spaces, its values are masks.
- **`prob`** (*`float`*) –The probability of mutation.

Returns A new subnet_dict after mutation.

Return type dict

static reset_in_subnet(*m*, *in_subnet=True*)

Reset the module's attribution.

Parameters

- **m** (`torch.nn.Module`) –The module in the supernet.
- **in_subnet** (*bool*) –If the module in subnet, set `in_subnet` to True, otherwise set to False.

sample_subnet(*searching=False*)

Random sample subnet by random mask.

Parameters **searching** (*bool*) –Whether is in search stage.

Returns

Record the information to build the subnet from the supernet, its keys are the properties `space_id` of placeholders in the mutator's search spaces, its values are random mask generated.

Return type dict

set_chosen_subnet(*subnet_dict*)

Set chosen subnet in the search_spaces after searching stage.

Parameters **subnet_dict** (*dict*) –Record the information to build the subnet from the supernet, its keys are the properties `space_id` of placeholders in the mutator's search spaces, its values are masks.

set_subnet(*subnet_dict*)

Setting subnet in the supernet based on the result of `sample_subnet` by changing the flag: `in_subnet`, which is easy to implement some operations for subnet, such as `forward`, calculate flops and so on.

Parameters **subnet_dict** (*dict*) –Record the information to build the subnet from the supernet, its keys are the properties `space_id` of placeholders in the mutator's search spaces, its values are masks.

6.7 ops

class `mmrazor.models.ops.`**DartsDilConv**(*kernel_size*, *use_drop_path=False*, *norm_cfg={'type': 'BN'}*, ***kwargs*)

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class `mmrazor.models.ops.`**`DartsPoolBN`**(*pool_type*, *kernel_size=3*, *norm_cfg={'type': 'BN'}*, *use_drop_path=False*, ***kwargs*)

`forward`(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class `mmrazor.models.ops.`**`DartsSepConv`**(*kernel_size*, *use_drop_path=False*, *norm_cfg={'type': 'BN'}*, ***kwargs*)

`forward`(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class `mmrazor.models.ops.`**`DartsSkipConnect`**(*use_drop_path=False*, *norm_cfg={'type': 'BN'}*, ***kwargs*)

Reduce feature map size by factorized pointwise (stride=2).

`forward`(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class mmrazor.models.ops.**DartsZero**(***kwargs*)

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class mmrazor.models.ops.**Identity**(*conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg=None*, ***kwargs*)

Base class for searchable operations.

Parameters

- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type=’BN’).
- **act_cfg** (*dict*) –Config dict for activation layer. Default: None.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class mmrazor.models.ops.**MBBlock**(*kernel_size*, *expand_ratio*, *se_cfg=None*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *drop_path_rate=0.0*, *with_cp=False*, ***kwargs*)

Mobilenet block for Searchable backbone.

Parameters

- **kernel_size** (*int*) –Size of the convolving kernel.
- **expand_ratio** (*int*) –The input channels’ expand factor of the depthwise convolution.

- **se_cfg** (*dict, optional*) –Config dict for se layer. Defaults to None, which means no se layer.
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type=' BN').
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type=' ReLU').
- **drop_path_rate** (*float*) –stochastic depth rate. Defaults to 0.
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.

Returns The output tensor.

Return type Tensor

forward(*x*)

Forward function.

Parameters **x** (*torch.Tensor*) –The input tensor.

Returns The output tensor.

Return type torch.Tensor

class mmrazor.models.ops.**ShuffleBlock**(*kernel_size*, *conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *with_cp=False*, ***kwargs*)

InvertedResidual block for Searchable ShuffleNetV2 backbone.

Parameters

- **kernel_size** (*int*) –Size of the convolving kernel.
- **stride** (*int*) –Stride of the convolution layer. Default: 1
- **conv_cfg** (*dict, optional*) –Config dict for convolution layer. Default: None, which means using conv2d.
- **norm_cfg** (*dict*) –Config dict for normalization layer. Default: dict(type=' BN').
- **act_cfg** (*dict*) –Config dict for activation layer. Default: dict(type=' ReLU').
- **with_cp** (*bool*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Default: False.

Returns The output tensor.

Return type Tensor

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

class `mmrazor.models.ops.`**`ShuffleXception`**(*conv_cfg=None*, *norm_cfg={'type': 'BN'}*, *act_cfg={'type': 'ReLU'}*, *with_cp=False*, ***kwargs*)

Xception block for ShuffleNetV2 backbone.

Parameters

- **`conv_cfg`** (*`dict, optional`*) –Config dict for convolution layer. Defaults to None, which means using conv2d.
- **`norm_cfg`** (*`dict`*) –Config dict for normalization layer. Defaults to dict(type=' BN').
- **`act_cfg`** (*`dict`*) –Config dict for activation layer. Defaults to dict(type=' ReLU').
- **`with_cp`** (*`bool`*) –Use checkpoint or not. Using checkpoint will save some memory while slowing down the training speed. Defaults to False.

Returns The output tensor.

Return type Tensor

`forward`(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

6.8 pruners

class `mmrazor.models.pruners.`**`RatioPruner`**(*ratios*, ***kwargs*)

A random ratio pruner.

Each layer can adjust its own width ratio randomly and independently.

Parameters **`ratios`** (*`list | tuple`*) –Width ratio of each layer can be chosen from *ratios* randomly. The width ratio is the ratio between the number of reserved channels and that of all channels in a layer. For example, if *ratios* is [0.25, 0.5], there are 2 cases for us to choose from when we sample from a layer with 12 channels. One is sampling the very first 3 channels in this layer, another is sampling the very first 6 channels in this layer. Default to None.

convert_switchable_bn(*module*, *num_bns*)

Convert normal `nn.BatchNorm2d` to `SwitchableBatchNorm2d`.

Parameters

- **module** (`torch.nn.Module`) –The module to be converted.
- **num_bns** (*int*) –The number of `nn.BatchNorm2d` in a `SwitchableBatchNorm2d`.

Returns

The converted module. Each `nn.BatchNorm2d` in this module has been converted to a `SwitchableBatchNorm2d`.

Return type `torch.nn.Module`

get_channel_mask(*out_mask*)

Randomly choose a width ratio of a layer from `ratios`

sample_subnet()

Random sample subnet by random mask.

Returns

Record the information to build the subnet from the supernet, its keys are the properties `space_id` in the pruner' s search spaces, and its values are corresponding sampled out_mask.

Return type dict

set_min_channel()

Set the number of channels each layer to minimum.

switch_subnet(*channel_cfg*, *subnet_ind=None*)

Switch the channel config of the supernet according to channel_cfg.

If we train more than one subnet together, we need to switch the channel_cfg from one to another during one training iteration.

Parameters

- **channel_cfg** (*dict*) –The channel config of a subnet. Key is space_id and value is a dict which includes out_channels (and in_channels if exists).
- **subnet_ind** (*int, optional*) –The index of the current subnet. If we replace normal BatchNorm2d with `SwitchableBatchNorm2d`, we should switch the index of `SwitchableBatchNorm2d` when switch subnet. Defaults to None.

class mmrazor.models.pruners.**StructurePruner**(*except_start_keys=['head.fc']*)

Base class for structure pruning. This class defines the basic functions of a structure pruner. Any pruner that inherits this class should at least define its own *sample_subnet* and *set_min_channel* functions. This part is being continuously optimized, and there may be major changes in the future.

Reference to https://github.com/jshilong/FisherPruning

Parameters **except_start_keys** (*List[str]*) –the module whose name start with a string in except_start_keys will not be prune.

add_pruning_attrs(*module*)

Add masks to a `nn.Module`.

build_channel_spaces(*name2module*)

Build channel search space.

Parameters **name2module** (*dict*) –A mapping between module_name and module.

Returns

The channel search space. The key is space_id and the value is the corresponding out_mask.

Return type dict

concat_backward_parser(*grad_fn*, *module2name*, *var2module*, *cur_path*, *result_paths*, *visited*)

Parse the backward of a concat operation.

Example

```
>>> conv = nn.Conv2d(3, 3, 3)
>>> pseudo_img = torch.rand(1, 3, 224, 224)
>>> out1 = conv(pseudo_img)
>>> out2 = conv(pseudo_img)
>>> out = torch.cat([out1, out2], dim=1)
>>> print(out.grad_fn.next_functions)
((<ThnnConv2DBackward object at 0x0000020E405F24C8>, 0),
(<ThnnConv2DBackward object at 0x0000020E405F2648>, 0))
>>> # the length of ``out.grad_fn.next_functions`` is two means
>>> # ``out`` is obtained by concatenating two tensors
```

conv_backward_parser(*grad_fn*, *module2name*, *var2module*, *cur_path*, *result_paths*, *visited*)

Parse the backward of a conv layer.

Example

```
>>> conv = nn.Conv2d(3, 3, 3)
>>> pseudo_img = torch.rand(1, 3, 224, 224)
>>> out = conv(pseudo_img)
>>> print(out.grad_fn.next_functions)
((None, 0), (<AccumulateGrad object at 0x0000020E405CBD88>, 0),
(<AccumulateGrad object at 0x0000020E405CB588>, 0))
>>> # op.next_functions[0][0] is None means this ThnnConv2DBackward
>>> # op has no parents
>>> # op.next_functions[1][0].variable is the weight of this Conv2d
>>> # module
>>> # op.next_functions[2][0].variable is the bias of this Conv2d
>>> # module
```

deploy_subnet(*supernet*, *channel_cfg*)

Deploy subnet according *channel_cfg*.

export_subnet()

Generate subnet configs according to the in_mask and out_mask of a module.

find_make_group_parser(*node_name*, *name2module*)

Find the corresponding make_group_parser according to the `node_name`

find_node_parents(*paths*)

Find the parent node of a node.

A node in the `paths` can be a module name or a operation name such as *concat_140719322997152*. Note that the string of numbers following `concat` do not have a particular meaning. It just make the operation name unique.

Parameters **paths** (*list*) –The traced paths.

get_max_channel_bins(*max_channel_bins*)

Get the max number of channel bins of all the groups which can be pruned during searching.

Parameters **max_channel_bins** (*int*) –The max number of bins in each layer.

get_space_id(*module_name*)

Get the corresponding space_id of the module_name.

The modules who share the same space_id will share the same out_mask. If the module is the output module(there is no other `nn.Module` whose input is its output), this function will return None. As the output module can not be pruned. If the input of this module is the concatenation of the output of several `nn.Module`, this function will return a dict object. If this module is in one of the groups, this function will return the group name. As the modules in the same group should share the same space_id. Otherwise, this function will return the module_name as space_id.

Parameters **module_name** (*str*) –the name of a `nn.Module`.

Returns the corresponding space_id of the module_name.

Return type str or dict or None

linear_backward_parser(*grad_fn*, *module2name*, *var2module*, *cur_path*, *result_paths*, *visited*)

Parse the backward of a conv layer.

Example

```
>>> fc = nn.Linear(3, 3, bias=True)
>>> input = torch.rand(3, 3)
>>> out = fc(input)
>>> print(out.grad_fn.next_functions)
((<AccumulateGrad object at 0x0000020E405F75C8>, 0), (None, 0),
(<TBackward object at 0x0000020E405F7D48>, 0))
>>> # op.next_functions[0][0].variable is the bias of this Linear
>>> # module
>>> # op.next_functions[1][0] is None means this AddmmBackward op
>>> # has no parents
>>> # op.next_functions[2][0] is the TBackward op, and
>>> # op.next_functions[2][0].next_functions[0][0].variable is
>>> # the transpose of the weight of this Linear module
```

make_same_out_channel_groups(*node2parents*, *name2module*)

Modules have the same child should be in the same group.

static modify_conv_forward(*module*)

Modify the forward method of a conv layer.

static modify_fc_forward(*module*)

Modify the forward method of a linear layer.

prepare_from_supernet(*supernet*)

Prepare for pruning.

abstract sample_subnet()

Sample a subnet from the supernet.

Returns

Record the information to build the subnet from the supernet, its keys are the properties `space_id` in the pruner' s search spaces, and its values are corresponding sampled out_mask.

Return type dict

set_channel_bins(*channel_bins_dict*, *max_channel_bins*)

Set subnet according to the number of channel bins in a layer.

Parameters

- **channel_bins_dict** (*dict*) –The number of bins in each layer. Key is the space_id of each layer and value is the corresponding mask of channel bin.
- **max_channel_bins** (*int*) –The max number of bins in each layer.

set_max_channel()

Set the number of channels each layer to maximum.

abstract set_min_channel()

Set the number of channels each layer to minimum.

set_subnet(*subnet_dict*)

Modify the in_mask and out_mask of modules in supernet according to subnet_dict.

Parameters **subnet_dict** (*dict*) –the key is space_id and the value is the corresponding sampled out_mask.

trace_bn_conv_links(*grad_fn*, *module2name*, *var2module*, *bn_conv_links*, *visited*)

Get the convolutional layer placed before a bn layer in the model.

Example

```
>>> conv = nn.Conv2d(3, 3, 3)
>>> bn = nn.BatchNorm2d(3)
>>> pseudo_img = torch.rand(1, 3, 224, 224)
>>> out = bn(conv(pseudo_img))
>>> print(out.grad_fn.next_functions)
((<ThnnConv2DBackward object at 0x0000020E40639688>, 0),
(<AccumulateGrad object at 0x0000020E40639208>, 0),
(<AccumulateGrad object at 0x0000020E406398C8>, 0))
>>> # op.next_functions[0][0] is ThnnConv2DBackward means
>>> # the parent of this NativeBatchNormBackward op is
>>> # ThnnConv2DBackward
>>> # op.next_functions[1][0].variable is the weight of this bn
>>> # module
>>> # op.next_functions[2][0].variable is the bias of this bn
>>> # module
```

trace_non_pass_path(*grad_fn*, *module2name*, *var2module*, *cur_path*, *result_paths*, *visited*)

Trace the topology of all the `NON_PASS_MODULE`.

CHAPTER

SEVEN

MMRAZOR.UTILS

`mmrazor.utils.`**`setup_multi_processes`**(*cfg*)

Setup multi-processing environment variables.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

INDEX