tensorcv documentation

Release 0.1

Qian Ge

Sep 12, 2018

Contents

CHAPTER 1

tensorcv

1.1 tensorcv package

1.1.1 Subpackages

tensorcv.callbacks package

Submodules

tensorcv.callbacks.base module

class tensorcv.callbacks.base.**Callback**

Bases: object

base class for callbacks

after_epoch()

after_run(*rct*, *val*)

after_train()

before_epoch()

before_inference()

before_run(*rct*)

before_train()

epochs_completed

global_step

setup_graph(*trainer*)

trigger()

trigger_epoch()

trigger_step()

class tensorcv.callbacks.base.**ProxyCallback**(*cb*)
Bases: *tensorcv.callbacks.base.Callback*

tensorcv.callbacks.debug module

class tensorcv.callbacks.debug.**CheckScalar**(*tensors*, *periodic=1*)
Bases: *tensorcv.callbacks.base.Callback*

print scalar tensor values during training .. attribute:: _tensors

_names

__init__(*tensors*, *periodic=1*)
init CheckScalar object :param tensors: list[string] A tensor name or list of tensor names

tensorcv.callbacks.group module

class tensorcv.callbacks.group.**Callbacks**(*cbs*)
Bases: *tensorcv.callbacks.base.Callback*

group all the callback

get_hooks()

tensorcv.callbacks.hooks module

class tensorcv.callbacks.hooks.**Callback2Hook**(*cb*)
Bases: tensorflow.python.training.session_run_hook.SessionRunHook

after_run(*rct*, *val*)
Called after each call to run().

The *run_values* argument contains results of requested ops/tensors by *before_run()*.

The *run_context* argument is the same one send to *before_run* call. *run_context.request_stop()* can be called to stop the iteration.

If *session.run()* raises any exceptions then *after_run()* is not called.

Parameters

- **run_context** – A *SessionRunContext* object.
- **run_values** – A SessionRunValues object.

before_run(*rct*)
Called before each call to run().

You can return from this call a *SessionRunArgs* object indicating ops or tensors to add to the upcoming *run()* call. These ops/tensors will be run together with the ops/tensors originally passed to the original run() call. The run args you return can also contain feeds to be added to the run() call.

The *run_context* argument is a *SessionRunContext* that provides information about the upcoming *run()* call: the originally requested op/tensors, the TensorFlow Session.

At this point graph is finalized and you can not add ops.

Parameters **run_context** – A *SessionRunContext* object.

Returns None or a *SessionRunArgs* object.

class tensorcv.callbacks.hooks.**Infer2Hook**(*inferencer*)
Bases: tensorflow.python.training.session_run_hook.SessionRunHook

after_run(*rct*, *val*)
Called after each call to run().

The *run_values* argument contains results of requested ops/tensors by *before_run()*.

The *run_context* argument is the same one send to *before_run* call. *run_context.request_stop()* can be called to stop the iteration.

If *session.run()* raises any exceptions then *after_run()* is not called.

Parameters

- **run_context** – A *SessionRunContext* object.
- **run_values** – A SessionRunValues object.

before_run(*rct*)
Called before each call to run().

You can return from this call a *SessionRunArgs* object indicating ops or tensors to add to the upcoming *run()* call. These ops/tensors will be run together with the ops/tensors originally passed to the original run() call. The run args you return can also contain feeds to be added to the run() call.

The *run_context* argument is a *SessionRunContext* that provides information about the upcoming *run()* call: the originally requested op/tensors, the TensorFlow Session.

At this point graph is finalized and you can not add ops.

Parameters **run_context** – A *SessionRunContext* object.

Returns None or a *SessionRunArgs* object.

class tensorcv.callbacks.hooks.**Prediction2Hook**(*prediction*)
Bases: tensorflow.python.training.session_run_hook.SessionRunHook

after_run(*rct*, *val*)
Called after each call to run().

The *run_values* argument contains results of requested ops/tensors by *before_run()*.

The *run_context* argument is the same one send to *before_run* call. *run_context.request_stop()* can be called to stop the iteration.

If *session.run()* raises any exceptions then *after_run()* is not called.

Parameters

- **run_context** – A *SessionRunContext* object.
- **run_values** – A SessionRunValues object.

before_run(*rct*)
Called before each call to run().

You can return from this call a *SessionRunArgs* object indicating ops or tensors to add to the upcoming *run()* call. These ops/tensors will be run together with the ops/tensors originally passed to the original run() call. The run args you return can also contain feeds to be added to the run() call.

The *run_context* argument is a *SessionRunContext* that provides information about the upcoming *run()* call: the originally requested op/tensors, the TensorFlow Session.

At this point graph is finalized and you can not add ops.

Parameters **run_context** – A *SessionRunContext* object.

Returns None or a *SessionRunArgs* object.

tensorcv.callbacks.inference module

class tensorcv.callbacks.inference.**FeedInference**(*inputs*, *periodic=1*, *inferencers=[]*, *extra_cbs=None*, *infer_batch_size=None*)

Bases: tensorcv.callbacks.inference.InferenceBase

default inferencer: inference_list = InferImages('generator/gen_image', prefix = 'gen')

class tensorcv.callbacks.inference.**GANInference**(*inputs=None*, *periodic=1*, *inferencers=None*, *extra_cbs=None*)

Bases: tensorcv.callbacks.inference.InferenceBase

class tensorcv.callbacks.inference.**FeedInferenceBatch**(*inputs*, *periodic=1*, *batch_count=10*, *inferencers=[]*, *extra_cbs=None*, *infer_batch_size=None*)

Bases: *tensorcv.callbacks.inference.FeedInference*

do not use all validation data

tensorcv.callbacks.inferencer module

class tensorcv.callbacks.inferencer.**InferencerBase**

Bases: *tensorcv.callbacks.base.Callback*

after_inference()

before_inference()

process before every inference

get_fetch(*val*)

put_fetch()

setup_inferencer()

class tensorcv.callbacks.inferencer.**InferImages**(*im_name*, *prefix=None*, *color=False*, *tanh=False*)

Bases: *tensorcv.callbacks.inferencer.InferencerBase*

class tensorcv.callbacks.inferencer.**InferScalars**(*scaler_names*, *summary_names=None*)

Bases: *tensorcv.callbacks.inferencer.InferencerBase*

class tensorcv.callbacks.inferencer.**InferOverlay**(*im_name*, *prefix=None*, *color=False*, *tanh=False*)

Bases: *tensorcv.callbacks.inferencer.InferImages*

class tensorcv.callbacks.inferencer.**InferMat**(*infer_save_name*, *mat_name*, *prefix=None*)

Bases: *tensorcv.callbacks.inferencer.InferImages*

tensorcv.callbacks.inputs module

class tensorcv.callbacks.inputs.**FeedInput**(*dataflow*, *placeholders*)
Bases: *tensorcv.callbacks.base.Callback*

input using feed

tensorcv.callbacks.monitors module

class tensorcv.callbacks.monitors.**TrainingMonitor**
Bases: *tensorcv.callbacks.base.Callback*

process_summary(*summary*)

class tensorcv.callbacks.monitors.**Monitors**(*mons*)
Bases: *tensorcv.callbacks.monitors.TrainingMonitor*

group monitors

class tensorcv.callbacks.monitors.**TFSummaryWriter**
Bases: *tensorcv.callbacks.monitors.TrainingMonitor*

process_summary(*summary*)

tensorcv.callbacks.saver module

class tensorcv.callbacks.saver.**ModelSaver**(*max_to_keep=5*, *keep_checkpoint_every_n_hours=0.5*, *periodic=1*, *checkpoint_dir=None*, *var_collections='variables'*)
Bases: *tensorcv.callbacks.base.Callback*

tensorcv.callbacks.summary module

class tensorcv.callbacks.summary.**TrainSummary**(*key=None*, *periodic=1*)
Bases: *tensorcv.callbacks.base.Callback*

tensorcv.callbacks.trigger module

class tensorcv.callbacks.trigger.**PeriodicTrigger**(*trigger_cb*, *every_k_steps=None*, *every_k_epochs=None*)
Bases: *tensorcv.callbacks.base.ProxyCallback*

may not need

Module contents

tensorcv.dataflow package

Subpackages

tensorcv.dataflow.dataset package

Submodules

tensorcv.dataflow.dataset.BSDS500 module

class tensorcv.dataflow.dataset.BSDS500.**BSDS500**(*name*, *data_dir=''*, *shuffle=True*, *normalize=None*, *is_mask=False*, *normalize_fnc=<function identity>*, *resize=None*)

Bases: *tensorcv.dataflow.image.ImageFromFile*

class tensorcv.dataflow.dataset.BSDS500.**BSDS500HED**(*name*, *data_dir=''*, *shuffle=True*, *normalize=None*, *is_mask=False*, *normalize_fnc=<function identity>*, *resize=None*)

Bases: *tensorcv.dataflow.dataset.BSDS500.BSDS500*

tensorcv.dataflow.dataset.CIFAR module

class tensorcv.dataflow.dataset.CIFAR.**CIFAR**(*data_dir=''*, *shuffle=True*, *normalize=None*)

Bases: *tensorcv.dataflow.base.RNGDataFlow*

next_batch()

size()

tensorcv.dataflow.dataset.MNIST module

class tensorcv.dataflow.dataset.MNIST.**MNIST**(*name*, *data_dir=''*, *shuffle=True*, *normalize=None*)

Bases: *tensorcv.dataflow.base.RNGDataFlow*

next_batch()

size()

class tensorcv.dataflow.dataset.MNIST.**MNISTLabel**(*name*, *data_dir=''*, *shuffle=True*, *normalize=None*)

Bases: *tensorcv.dataflow.dataset.MNIST.MNIST*

next_batch()

Module contents

Submodules

tensorcv.dataflow.base module

class tensorcv.dataflow.base.**DataFlow**
Bases: object

base class for dataflow

after_reading()

before_read_setup(***kwargs*)

epochs_completed

next_batch()

next_batch_dict()

reset_epochs_completed(*val*)

reset_state()

set_batch_size(*batch_size*)

setup(*epoch_val*, *batch_size*, ***kwargs*)

size()

class tensorcv.dataflow.base.**RNGDataFlow**
Bases: *tensorcv.dataflow.base.DataFlow*

suffle_data()

tensorcv.dataflow.common module

tensorcv.dataflow.common.**dense_to_one_hot**(*labels_dense*, *num_classes*)
Convert class labels from scalars to one-hot vectors.

tensorcv.dataflow.common.**get_file_list**(*file_dir*, *file_ext*, *sub_name=None*)

tensorcv.dataflow.common.**get_folder_list**(*folder_dir*)

tensorcv.dataflow.common.**get_folder_names**(*folder_dir*)

tensorcv.dataflow.common.**input_val_range**(*in_mat*)

tensorcv.dataflow.common.**load_image**(*im_path*, *read_channel=None*, *pf=<function identity>*, *resize=None*, *resize_crop=None*)

tensorcv.dataflow.common.**print_warning**(*warning_str*)

tensorcv.dataflow.common.**reverse_label_dict**(*label_dict*)

tensorcv.dataflow.common.**tanh_normalization**(*data*, *half_in_val*)

tensorcv.dataflow.image module

class tensorcv.dataflow.image.**ImageData**(*ext_name*, *data_dir=''*, *shuffle=True*, *normalize=None*)

Bases: *tensorcv.dataflow.base.RNGDataFlow*

next_batch()

size()

class tensorcv.dataflow.image.**DataFromFile**(*ext_name*, *data_dir=''*, *num_channel=None*, *shuffle=True*, *normalize=None*, *batch_dict_name=None*, *normalize_fnc=<function identity>*)

Bases: *tensorcv.dataflow.base.RNGDataFlow*

Base class for image from files

get_sample_data()

next_batch()

next_batch_dict()

class tensorcv.dataflow.image.**ImageLabelFromFolder**(*ext_name*, *data_dir=''*, *num_channel=None*, *label_dict=None*, *num_class=None*, *one_hot=False*, *shuffle=True*, *normalize=None*, *resize=None*, *resize_crop=None*, *batch_dict_name=None*, *pf=<function identity>*)

Bases: *tensorcv.dataflow.image.ImageFromFile*

read image data with label in subfolder name

__init__(*ext_name*, *data_dir=''*, *num_channel=None*, *label_dict=None*, *num_class=None*, *one_hot=False*, *shuffle=True*, *normalize=None*, *resize=None*, *resize_crop=None*, *batch_dict_name=None*, *pf=<function identity>*)

Parameters **label_dict** (*dict*) – empty or full

get_data_list()

get_label_list()

set_data_list(*new_data_list*)

size()

class tensorcv.dataflow.image.**ImageLabelFromFile**(*ext_name*, *data_dir=''*, *label_file_name=''*, *num_channel=None*, *one_hot=False*, *label_dict={}*, *num_class=None*, *shuffle=True*, *normalize=None*, *resize=None*, *resize_crop=None*, *batch_dict_name=None*, *pf=<function identity>*)

Bases: *tensorcv.dataflow.image.ImageLabelFromFolder*

read image data with label in a separate file txt

class tensorcv.dataflow.image.**ImageFromFile**(*ext_name*, *data_dir=''*, *num_channel=None*, *shuffle=True*, *normalize=None*, *normalize_fnc=<function identity>*, *resize=None*, *resize_crop=None*, *batch_dict_name=None*, *pf=<function identity>*)

Bases: *tensorcv.dataflow.image.DataFromFile*

get_data_list()

set_data_list(*new_data_list*)

set_pf(*pf*)

size()

suffle_data()

class tensorcv.dataflow.image.**ImageDenseLabel**(*ext_name*, *im_pre*, *label_pre*, *mask_pre=None*, *data_dir=''*, *num_channel=None*, *shuffle=True*, *normalize=None*, *normalize_fnc=<function identity>*, *resize=None*, *resize_crop=None*, *batch_dict_name=None*, *is_binary=False*)

Bases: *tensorcv.dataflow.image.ImageFromFile*

get_data_list()

get_label_list()

set_data_list(*new_data_list*)

tensorcv.dataflow.matlab module

class tensorcv.dataflow.matlab.**MatlabData**(*data_dir=''*, *mat_name_list=None*, *mat_type_list=None*, *shuffle=True*, *normalize=None*)

Bases: *tensorcv.dataflow.base.RNGDataFlow*

dataflow from .mat file with mask

next_batch()

size()

tensorcv.dataflow.randoms module

class tensorcv.dataflow.randoms.**RandomVec**(*len_vec=100*)

Bases: *tensorcv.dataflow.base.DataFlow*

random vector input

next_batch()

reset_state()

size()

Module contents

tensorcv.models package

Submodules

tensorcv.models.base module

class tensorcv.models.base.**ModelDes**
Bases: object

base model for ModelDes

create_graph()

create_model(*inputs=None*)

ex_init_model(*dataflow*, *trainer*)

get_batch_size()

get_global_step

get_graph_feed()

get_prediction_placeholder()

get_train_placeholder()

model_input

set_batch_size(*val*)

set_dropout(*dropout_placeholder*, *keep_prob=0.5*)

set_is_training(*is_training=True*)

set_model_input(*inputs=None*)

set_prediction_placeholder(*plhs=None*)

set_train_placeholder(*plhs=None*)

setup_summary()

class tensorcv.models.base.**BaseModel**
Bases: *tensorcv.models.base.ModelDes*

Model with single loss and single optimizer

default_collection

get_grads()

get_loss()

get_optimizer()

class tensorcv.models.base.**GANBaseModel**(*input_vec_length*, *learning_rate*)
Bases: *tensorcv.models.base.ModelDes*

Base model for GANs

d_collection

def_loss(*dis_loss_fnc*, *gen_loss_fnc*)
updata definintion of loss functions

g_collection

get_discriminator_grads()

get_discriminator_loss()

get_discriminator_optimizer()

get_gen_data()

get_generator_grads()

get_generator_loss()

get_generator_optimizer()

get_graph_feed()

get_random_vec_placeholder()

get_sample_gen_data()

tensorcv.models.layers module

tensorcv.models.layers.**batch_flatten**(*x*)
Flatten the tensor except the first dimension.

tensorcv.models.layers.**batch_norm**(*x*, *train=True*, *name='bn'*)
batch normal

Parameters

- **x** (*tf.tensor*) – a tensor
- **name** (*str*) – name scope
- **train** (*bool*) – whether training or not

Returns tf.tensor with name 'name'

tensorcv.models.layers.**conv**(*x*, *filter_size*, *out_dim*, *name='conv'*, *stride=1*, *padding='SAME'*, *nl=<function identity>*, *data_dict=None*, *init_w=None*, *init_b=None*, *use_bias=True*, *wd=None*, *trainable=True*)
2D convolution

Parameters

- **x** (*tf.tensor*) – a 4D tensor Input number of channels has to be known
- **filter_size** (*int or list with length 2*) – size of filter
- **out_dim** (*int*) – number of output channels
- **name** (*str*) – name scope of the layer
- **stride** (*int or list*) – stride of filter
- **padding** (*str*) – 'VALID' or 'SAME'
- **init_b** (*init_w,*) – initializer for weight and bias variables. Default to 'random_normal_initializer'
- **nl** – a function

Returns tf.tensor with name 'output'

tensorcv.models.layers.**dconv**(*x*, *filter_size*, *out_dim=None*, *out_shape=None*, *out_shape_by_tensor=None*, *name='dconv'*, *stride=2*, *padding='SAME'*, *nl=<function identity>*, *data_dict=None*, *init_w=None*, *init_b=None*, *wd=None*, *trainable=True*)

2D deconvolution

Parameters

- **x** (*tf.tensor*) – a 4D tensor Input number of channels has to be known
- **filter_size** (*int or list with length 2*) – size of filter
- **out_dim** (*int*) – number of output channels
- **out_shape** (*list(int)*) – shape of output without None
- **out_shape_by_tensor** (*tf.tensor*) – a tensor has the same shape of output except the out_dim
- **name** (*str*) – name scope of the layer
- **stride** (*int or list*) – stride of filter
- **padding** (*str*) – 'VALID' or 'SAME'
- **init** – initializer for variables. Default to 'random_normal_initializer'
- **nl** – a function

Returns tf.tensor with name 'output'

tensorcv.models.layers.**dropout**(*x*, *keep_prob*, *is_training*, *name='dropout'*)

Dropout

Parameters

- **x** (*tf.tensor*) – a tensor
- **keep_prob** (*float*) – keep prbability of dropout
- **is_training** (*bool*) – whether training or not
- **name** (*str*) – name scope

Returns tf.tensor with name 'name'

tensorcv.models.layers.**fc**(*x*, *out_dim*, *name='fc'*, *nl=<function identity>*, *init_w=None*, *init_b=None*, *data_dict=None*, *wd=None*, *trainable=True*, *re_dict=False*)

Fully connected layer

Parameters

- **x** (*tf.tensor*) – a tensor to be flattened The first dimension is the batch dimension
- **num_out** (*int*) – dimension of output
- **name** (*str*) – name scope of the layer
- **init** – initializer for variables. Default to 'random_normal_initializer'
- **nl** – a function

Returns tf.tensor with name 'output'

tensorcv.models.layers.**get_shape2D**(*in_val*)

Return a 2D shape

Parameters **in_val** (*int or list with length 2*) –

Returns list with length 2

tensorcv.models.layers.**get_shape4D**(*in_val*)
Return a 4D shape

Parameters **in_val** (*int or list with length 2*) –

Returns list with length 4

tensorcv.models.layers.**global_avg_pool**(*x*, *name='global_avg_pool'*, *data_format='NHWC'*)

tensorcv.models.layers.**leaky_relu**(*x*, *leak=0.2*, *name='LeakyRelu'*)
Allow a small non-zero gradient when the unit is not active

Parameters

- **x** (*tf.tensor*) – a tensor
- **leak** (*float*) – Default to 0.2

Returns tf.tensor with name 'name'

tensorcv.models.layers.**max_pool**(*x*, *name='max_pool'*, *filter_size=2*, *stride=None*, *padding='VALID'*)
Max pooling layer

Parameters

- **x** (*tf.tensor*) – a tensor
- **name** (*str*) – name scope of the layer
- **filter_size** (*int or list with length 2*) – size of filter
- **stride** (*int or list with length 2*) – Default to be the same as shape
- **padding** (*str*) – 'VALID' or 'SAME'. Use 'SAME' for FCN.

Returns tf.tensor with name 'name'

tensorcv.models.layers.**new_biases**(*name*, *idx*, *shape*, *initializer=None*, *data_dict=None*, *trainable=True*)

tensorcv.models.layers.**new_normal_variable**(*name*, *shape=None*, *trainable=True*, *stddev=0.002*)

tensorcv.models.layers.**new_variable**(*name*, *idx*, *shape*, *initializer=None*)

tensorcv.models.layers.**new_weights**(*name*, *idx*, *shape*, *initializer=None*, *wd=None*, *data_dict=None*, *trainable=True*)

tensorcv.models.losses module

tensorcv.models.losses.**GAN_discriminator_loss**(*d_real*, *d_fake*, *name='d_loss'*)

tensorcv.models.losses.**GAN_generator_loss**(*d_fake*, *name='g_loss'*)

tensorcv.models.losses.**comp_loss_fake**(*discrim_output*)

tensorcv.models.losses.**comp_loss_real**(*discrim_output*)

Module contents

tensorcv.predicts package

Submodules

tensorcv.predicts.base module

class tensorcv.predicts.base.**Predictor**(*config*)
Bases: object

Base class for a predictor. Used to run all predictions.

config
PridectConfig – the config used for this predictor

model
ModelDes

input
DataFlow

sess
tf.Session

hooked_sess
tf.train.MonitoredSession

__init__(*config*)
Inits Predictor with config (PridectConfig).

Will create session as well as monitored sessions for each predictions, and load pre-trained parameters.

Parameters **config** (PridectConfig) – the config used for this predictor

after_prediction()

run_predict()
Run predictions and the process after finishing predictions.

tensorcv.predicts.config module

class tensorcv.predicts.config.**PridectConfig**(*dataflow=None*, *model=None*, *model_dir=None*, *model_name=''*, *restore_vars=None*, *session_creator=None*, *predictions=None*, *batch_size=1*, *default_dirs=None*)
Bases: object

__init__(*dataflow=None*, *model=None*, *model_dir=None*, *model_name=''*, *restore_vars=None*, *session_creator=None*, *predictions=None*, *batch_size=1*, *default_dirs=None*)
Args:

callbacks

tensorcv.predicts.predictions module

class tensorcv.predicts.predictions.**PredictionImage**(*prediction_image_tensors*, *save_prefix*, *merge_im=False*, *tanh=False*, *color=False*)

Bases: tensorcv.predicts.predictions.PredictionBase

Predict image output and save as files.

Images are saved every batch. Each batch result can be save in one image or individule images.

__init__(*prediction_image_tensors*, *save_prefix*, *merge_im=False*, *tanh=False*, *color=False*)

Parameters

- **prediction_image_tensors** (*list*) – a list of tensor names
- **save_prefix** (*list*) – a list of file prefix for saving each tensor in prediction_image_tensors
- **merge_im** (*bool*) – merge output of one batch or not

class tensorcv.predicts.predictions.**PredictionScalar**(*prediction_scalar_tensors*, *print_prefix*)

Bases: tensorcv.predicts.predictions.PredictionBase

__init__(*prediction_scalar_tensors*, *print_prefix*)

Parameters

- **prediction_scalar_tensors** (*list*) – a list of tensor names
- **print_prefix** (*list*) – a list of name prefix for printing each tensor in prediction_scalar_tensors

class tensorcv.predicts.predictions.**PredictionMat**(*prediction_tensors*, *save_prefix*)

Bases: tensorcv.predicts.predictions.PredictionBase

class tensorcv.predicts.predictions.**PredictionMeanScalar**(*prediction_scalar_tensors*, *print_prefix*)

Bases: *tensorcv.predicts.predictions.PredictionScalar*

class tensorcv.predicts.predictions.**PredictionOverlay**(*prediction_image_tensors*, *save_prefix*, *merge_im=False*, *tanh=False*, *color=False*)

Bases: *tensorcv.predicts.predictions.PredictionImage*

tensorcv.predicts.simple module

class tensorcv.predicts.simple.**SimpleFeedPredictor**(*config*)

Bases: *tensorcv.predicts.base.Predictor*

predictor with feed input

Module contents

tensorcv.train package

Submodules

tensorcv.train.base module

class tensorcv.train.base.**Trainer**(*config*)
Bases: object

base class for trainer

epochs_completed

get_global_step

main_loop()

register_callback(*cb*)

register_monitor(*monitor*)

setup()

setup_graph()

train()

tensorcv.train.config module

class tensorcv.train.config.**TrainConfig**(*dataflow=None*, *model=None*, *callbacks=[]*, *session_creator=None*, *monitors=None*, *batch_size=1*, *max_epoch=100*, *summary_periodic=None*, *is_load=False*, *model_name=None*, *default_dirs=None*)
Bases: object

callbacks

class tensorcv.train.config.**GANTrainConfig**(*dataflow=None*, *model=None*, *discriminator_callbacks=[]*, *generator_callbacks=[]*, *session_creator=None*, *monitors=None*, *batch_size=1*, *max_epoch=100*, *summary_d_periodic=None*, *summary_g_periodic=None*, *default_dirs=None*)
Bases: *tensorcv.train.config.TrainConfig*

dis_callbacks

gen_callbacks

tensorcv.train.simple module

class tensorcv.train.simple.**SimpleFeedTrainer**(*config*)
Bases: *tensorcv.train.base.Trainer*

single optimizer

Module contents

tensorcv.utils package

Submodules

tensorcv.utils.common module

tensorcv.utils.common.**apply_mask**(*input_matrix*, *mask*)
Get partition of input_matrix using index 1 in mask.

Parameters

- **input_matrix** (*Tensor*) – A Tensor
- **mask** (*int*) – A Tensor of type int32 with indices in {0, 1}. Shape has to be the same as input_matrix.

Returns A Tensor with elements from data with entries in mask equal to 1.

tensorcv.utils.common.**apply_mask_inverse**(*input_matrix*, *mask*)
Get partition of input_matrix using index 0 in mask.

Parameters

- **input_matrix** (*Tensor*) – A Tensor
- **mask** (*int*) – A Tensor of type int32 with indices in {0, 1}. Shape has to be the same as input_matrix.

Returns A Tensor with elements from data with entries in mask equal to 0.

tensorcv.utils.common.**get_tensors_by_names**(*names*)
Get a list of tensors by the input name list.

Parameters **names** (*str*) – A str or a list of str

Returns A list of tensors with name in input names.

> **Warning:** If more than one tensor have the same name in the graph. This function will only return the tensor with name NAME:0.

tensorcv.utils.common.**deconv_size**(*input_height*, *input_width*, *stride=2*)
Compute the feature size (height and width) after filtering with a specific stride. Mostly used for setting the shape for deconvolution.

Parameters

- **input_height** (*int*) – height of input feature
- **input_width** (*int*) – width of input feature
- **stride** (*int*) – stride of the filter

Returns *(int, int)* – Height and width of feature after filtering.

tensorcv.utils.common.**match_tensor_save_name**(*tensor_names*, *save_names*)
Match tensor_names and corresponding save_names for saving the results of the tenors. If the number of tensors is less or equal to the length of save names, tensors will be saved using the corresponding names in save_names. Otherwise, tensors will be saved using their own names. Used for prediction or inference.

Parameters

- **tensor_names** (*str*) – List of tensor names
- **save_names** (*str*) – List of names for saving tensors

Returns *(list, list)* – List of tensor names and list of names to save the tensors.

tensorcv.utils.default module

tensorcv.utils.default.**get_default_session_config**(*memory_fraction=1*)
Default config of a TensorFlow session

Parameters **memory_fraction** (*float*) – Memory fraction of GPU for this session

Returns *tf.ConfigProto()* – Config of session.

tensorcv.utils.sesscreate module

class tensorcv.utils.sesscreate.**NewSessionCreator**(*target=''*, *graph=None*, *config=None*)
Bases: tensorflow.python.training.monitored_session.SessionCreator

tf.train.SessionCreator for a new session

__init__(*target=''*, *graph=None*, *config=None*)
Inits NewSessionCreator with targe, graph and config.

Parameters

- **target** – same as tf.Session.__init__().
- **graph** – same as tf.Session.__init__().
- **config** – same as tf.Session.__init__(). Default to utils.default.get_default_session_config().

create_session()
Create session as well as initialize global and local variables

Returns A tf.Session object containing nodes for all of the operations in the underlying TensorFlow graph.

class tensorcv.utils.sesscreate.**ReuseSessionCreator**(*sess*)
Bases: tensorflow.python.training.monitored_session.SessionCreator

tf.train.SessionCreator for reuse an existed session

__init__(*sess*)
Inits ReuseSessionCreator with an existed session.

Parameters **sess** (*tf.Session*) – an existed tf.Session object

create_session()
Create session by reusing an existing session

Returns A reused tf.Session object containing nodes for all of the operations in the underlying TensorFlow graph.

tensorcv.utils.utils module

tensorcv.utils.utils.**get_rng**(*obj=None*)

This function is copied from tensorpack. Get a good RNG seeded with time, pid and the object. :param obj: some object to use to generate random seed.

Returns *np.random.RandomState* – the RNG.

tensorcv.utils.viz module

tensorcv.utils.viz.**image_overlay**(*im_1*, *im_2*, *color=True*, *normalize=True*)

Overlay two images with the same size.

Parameters

- **im_1** (*np.ndarray*) – image arrary
- **im_2** (*np.ndarray*) – image arrary
- **color** (*bool*) – Whether convert intensity image to color image.
- **normalize** (*bool*) – If both color and normalize are True, will normalize the intensity so that it has minimum 0 and maximum 1.

Returns *np.ndarray* – an overlay image of im_1*0.5 + im_2*0.5

tensorcv.utils.viz.**intensity_to_rgb**(*intensity*, *cmap='jet'*, *normalize=False*)

This function is copied from tensorpack. Convert a 1-channel matrix of intensities to an RGB image employing a colormap. This function requires matplotlib. See matplotlib colormaps for a list of available colormap.

Parameters

- **intensity** (*np.ndarray*) – array of intensities such as saliency.
- **cmap** (*str*) – name of the colormap to use.
- **normalize** (*bool*) – if True, will normalize the intensity so that it has minimum 0 and maximum 1.

Returns *np.ndarray* – an RGB float32 image in range [0, 255], a colored heatmap.

tensorcv.utils.viz.**save_merge_images**(*images*, *merge_grid*, *save_path*, *color=False*, *tanh=False*)

Save multiple images with same size into one larger image.

The best size number is int(max(sqrt(image.shape[0]),sqrt(image.shape[1]))) + 1

Parameters

- **images** (*np.ndarray*) – A batch of image array to be merged with size [BATCH_SIZE, HEIGHT, WIDTH, CHANNEL].
- **merge_grid** (*list*) – List of length 2. The grid size for merge images.
- **save_path** (*str*) – Path for saving the merged image.
- **color** (*bool*) – Whether convert intensity image to color image.
- **tanh** (*bool*) – If True, will normalize the image in range [-1, 1] to [0, 1] (for GAN models).

Example

The batch_size is 64, then the size is recommended [8, 8]. The batch_size is 32, then the size is recommended [6, 6].

Module contents

1.1.2 Module contents

Python Module Index

Index

Symbols

A

B

C

D

E

F

G

R

S

T