API Documentation

API reference

Module sbrain.dataset

Class DataSetImageClassification

A DataSetImageClassification represents an image dataset that can be used for training classification models. Comprises of images and labels.

__init__ (name)

Instantiates a dataset object.

Args:

name : Name of the dataset

create (description,

source_archive_path,

classes,

collection_date,

image_iterator,

label_iterator)

Creates the dataset in the system.

Args:

description: description about the dataset

source_archive_path: path to the directory with the original images and labels, that the user wants to register with the SBrain.

classes: (optional) a dict with the different classes and unique numeric ids representing those classes.

e.g: {“cat”:0, ”dog”:1}

collection_date: String representing the date on which the data was collected. Should have the following format “mm-dd-yyyy” e.g. “06-12-2018”

image_iterator: A function that returns an iterator to a list with paths of images in the source_archive_path.

label_iterator: A function that returns an iterator to a list tuples. The tuple is of format (image_name, label) where image name is the name of image in the dataset and label is a string. It could be the class of the image, or multiple classes comma separated (in case of multi label problems).

Both image_iterator and label_iterator functions take “root_path” as argument. Root path is source_archive_path. This path will be provided by SBrain when calling these functions.

return a DataSetExtractionJob.

retry_create (source_archive_path,

classes,

collection_date,

image_iterator,

label_iterator)

Retry tries to retry creating the dataset in the system, typically when a dataset was creation failed due to image_iterator/ label_iterator

functions fail due to some issues.

Args:

source_archive_path: (optional) file path with the original images and labels, that the user wants to register with the SBrain. If this parameter is not given, the original source_archive_path, provided while dataset creation will be used.

classes: (optional) a dict with the different classes and unique numeric ids representing those classes. If this parameter is not given, the original 'classes', provided while dataset creation will be used.

e.g: {“cat”:0, ”dog”:1}

collection_date: (optional) String representing the date on which the data was collected. Should have the following format “mm-dd-yyyy” e.g. “06-12-2018”. If this parameter is not given, the original 'collection_date', provided while dataset creation will be used.

image_iterator: (optional) A function that returns an iterator to a list with paths of images in the source_archive_path. If this parameter is not given, the original 'image_iterator', provided while dataset creation will be used.

label_iterator: A function that returns an iterator to a list tuples. The tuple is of format (image_name, label) where image name is the name of image in the dataset and label is a string. It could be the class of the image, or multiple classes comma separated (in case of multi label problems).

Both image_iterator and label_iterator functions take “root_path” as argument. Root path is source_archive_path. This path will be provided by SBrain when calling these functions.

return a DataSetExtractionJob.

NOTE :

1. If the dataset.create() failed because of faulty image iterator, call the retry_create() with only the "image_iterator" parameter.

Other parameters are optional and the original values provided in create() will be used.

2. If the dataset.create() failed because of faulty label iterator, you can call the retry_create() with only the "label_iterator" parameter.

Other parameters are optional and the original values provided in create() will be used.

3. "collection_date" parameter can be used to override the date given in original create() api, only if this parameter is passed to retry_create()

along with "source_archive_path" and/or "image_iterator" parameter.

search(name, author, description)

searches image datasets with given search criteria.

Args:

name: (optional) Name of the dataset to be searched.

DataSets with partially matching names are also returned.

author: (optional) Author or name of the user who created the dataset.

DataSets with author names partially matching, are also returned.

description: (optional) description of the DataSet.

DataSets with words in the description matching the input are returned.

lookup(name)

Looks up a dataset by name. Name should match exactly, its case sensitive.

Args:

name: name of the dataset to look up.

version(version_name)

Returns a version of the dataset with given version_name.

Args:

version_name: exact name of the DataSetVersion belonging to this dataset to be returned.

search_versions(version_name=None, version_author=None, version_description=None)

searches image DataSetVersions with given search criteria for current dataset.

Args:

version_name: (optional) Name of the dataSetVersion to be searched.

DataSetVersions with partially matching names are also returned.

version_author: (optional) Author or name of the user who created the dataSetVersion.

DataSetVersions with author names partially matching, are also returned.

version_description: (optional) description of the DataSetVersion.

DataSetVersions with words in the description matching the input are returned.

Class DataSetVersion

Represents particular version of a root dataset. Version "v1" is created by default when a dataset is created.

Additional versions of the dataset can be created by applying different transformations.

__init__(source_dataset, version)

Instantiates a DataSetVersion object.

Args:

source_dataset: The source dataset from which this dataset version belongs to.

version: string name of the version.

transform (transformation**)**

Method to apply a transformation to the DataSetVersion.

Args:

transformation: object of type Transformation, which represents the transformation that is to be applied to the DataSetVersion.

split(split_name)

Looks up a DataSetSplit for this DataSetVersion, split_name should match exactly, its case sensitive.

Args:

split_name: name of the DataSetSplit to look up.

lookup(dataset_name, dataset_version_name)

Looks up a dataset version using dataset name and dataset version name, dataset_name and dataset_version_name should match exactly, its case sensitive.

Args:

dataset_name: name of the dataset to look up.

dataset_version_name: name of the dataset version to look up.

create_data_split(split_name, split_percentage, description, data_exclude_function, label_exclude_function)

Process to create DataSetSplit for this DataSetVersion

Args:

split_name: name of the DataSetSplit to be created

split_percentage: a list of numbers representing ratio of each portion of all desired splits, total must be equal to 100%.

description: description of the split process

data_exclude_function: a function that works like a filter to exclude data while processing the splits.

label_exclude_function: a function that works like a filter to exclude label while processing the splits.

search_splits(split_name, split_author, split_description)

searches DataSetsplits using the given search criteria for the current DataSetVersion.

Args:

split_name: (optional) Name of the dataSetSplit to be searched.

DataSetSplits with partially matching names are also returned.

split_author: (optional) Author or name of the user who created the dataSetSplit.

DataSetSplits with author names partially matching, are also returned.

split_description: (optional) description of the DataSetSplit.

DataSetSplits with words in the description matching the input are returned.

search(dataset_name, name, author, description)

searches DataSetVersions using the given search criteria.

Args:

dataset_name: (optional) Name of the dataSet to be searched.

DataSetVersions with partially matching names are also returned.

name: (optional) Name of the dataSetVersion to be searched.

DataSetVersions with partially matching names are also returned.

author: (optional) Author or name of the user who created the dataSetVersion.

DataSetVersions with author names partially matching, are also returned.

description: (optional) description of the DataSetVersion.

DataSetVersions with words in the description matching the input are returned.

Class DataSetSplit

A DataSetSplit represents the process to divide dataset to multiple splits using the given ratio, for example, split a dataset to train, validate and test.

__init__ (name, dataset_version, split_percentages)

Instantiates a DataSetSplit object.

Args:

name : Name of the DataSetSplit

dataset_version : instance of DataSetVersion

split_percentages: a list of numbers representing ratio of each portion of all desired splits, total must be equal to 100%

create (description, data_exclude_function, label_exclude_function)

Creates the DataSetSplit in the system.

Args:

description: description of the split process

data_exclude_function: a function that works like a filter to exclude data while processing the splits.

label_exclude_function: a function that works like a filter to exclude label while processing the splits.

search(dataset_name, dataset_version_name, split_name, author, description)

searches DataSetSplits with given search criteria.

Args:

dataset_name: (optional) Name of the dataset to be searched.

DatasetSplit with partially matching names are also returned.

dataset_version_name: (optional) Name of the DataSet Version to be searched.

DatasetSplit with partially matching names are also returned.

split_name: (optional) Name of the DataSetSplit to be searched.

DatasetSplit with partially matching names are also returned.

author: (optional) Author or name of the user who created the DataSetSplit.

DataSetSplit with author names partially matching, are also returned.

description: (optional) description of the DataSetSplit.

DataSetSplit with words in the description matching the input are returned.

lookup(dataset_name, dataset_version_name, split_name)

Looks up a DataSetSplit by dataset name, DataSetVersion name, split name. Name should match exactly, its case sensitive.

Args:

dataset_name: Name of the dataset to look up.

dataset_version_name: Name of the DataSetVersion to look up.

split_name: Name of the DataSetSplit to look up.

retrieve(id)

Looks up a DataSetSplit by id. id is the primary key in the database. It is int type and should be matched exactly.

Args:

id: id of the DatasetSplit to look up.

Class TransformationSet

A TransformationSet represents the process to apply a sequence of transformations to a given source DataSetVersion.

__init__ (source_dataset_version)

Instantiates a TransformationSet object.

Args:

source_dataset_version : a source DataSetVersion object

transform (transformation**)**

add a transformation to list that will be applied to the source DataSetVersion.

Args:

transformation: object of type Transformation, which represents the transformation to be applied to the source DataSetVersion.

run (number_workers, target_version, cores, memory, partitions, data_exclude_function, label_exclude_function)

kick off the action to apply all transformations added in transformation set to the source DataSetVersion and produce a target DataSetVersion, a transformation job will be created for the process.

Args:

number_workers: number of spark cluster workers to be provisioned for the transformation job.

target_version: the target DataSetVersion name to be created for result of the transformation job.

cores: number of cores of spark cluster worker machine that will be provisioned for the transformation job, default 2.

memory: memory size of spark cluster worker machine that will be provisioned for the transformation job, default 2G.

partitions: repartition the source dataset to specified partition number. default value 0.

-1 : no repartition,

0 : repartition to the size of cores x number _workers.

i : positive integer i, repartititon to i, otherwise error.

data_exclude_function: a function that works like a filter to exclude data while processing the transformation.

label_exclude_function: a function that works like a filter to exclude label while processing the transformation.

apply_to_file(src_path, des_path, transformations_set)

Its a static method, provided for testing and debugging purposes, that can be used to test given set of transformations on a single image file.

Args:

src_path: path to source the image file.

des_path: path where to write the output image.

transformations_set: list of transformation objects to be applied on the source image.

Class Transformation

A Transformation represents the process logic that transforms a given input from one form to another form.

__init__ (name)

Instantiates a Transformation object.

Args:

name : Name of the transformation

create (author, description)

Creates the transformation in the system.

Args:

author: author of the transformation

description: description about the transformation

search(name, author, description)

searches image transformation with given search criteria.

Args:

name: (optional) Name of the transformation to be searched.

Transformations with partially matching names are also returned.

author: (optional) Author or name of the user who created the transformation.

Transformation with author names partially matching, are also returned.

description: (optional) description of the Transformation.

Transformations with words in the description matching the input are returned.

lookup(name)

Looks up a transformation by name. Name should match exactly, its case sensitive.

Args:

name: name of the transformation to look up.

retrieve(transform_id)

Looks up a transformation by transform id. transform_id is the primary key in the database. It is int type and should be matched exactly

Args:

transform_id: id of the transformation to look up.

override(override_parm)**

Overrides an inherited transformation object's attributes.

Args:

override_parm: a number of key value pairs in the form of "key=value"

process(arr_in)**

apply the transformations to the input.

Args:

arr_in: the input to be applied transformations.

Class DataSetExtractionJob

A DataSetExtractionJob represents the process that creates a new data set.

__init__ (job_name, job_id, dataset_name, dataset_version_name, job_status, job_created_date, details)

Instantiates a DataSetExtractionJob object.

Args:

job_name : Name of the job

job_id : id of the job

dataset_name : Name of the dataset

dataset_version_name : Name of dataset version

job_status : status of the job

job_created_date : create date of the job

details: details about this job

get_status()

get current status of job.

get_dataset()

get the dataset created from this job.

cancel()

cancel this job.

search_jobs(dataset_name, job_name, age, created_date, updated_date, author, status**)**

Args:

dataset_name : (optional) dataset name to be searched, partial match allowed

job_name : (optional), job name, partial match allowed.

age : (optional), age of job, only job age is less than this age will be returned

created_date : (optional) created date of job

updated_date : (optional) updated date of job

author : (optional) author, partial matched allowed

status: (optional) status of job, partial matched allowed

Class DataSeSplitJob

A DataSetSplitJob represents the process that creates a data set split.

__init__ (job_name, job_id, dataset_name, dataset_version_name, dataset_split_name, job_status, job_created_date, split_percentage, details)

Instantiates a DataSetExtractionJob object.

Args:

job_name : Name of the job

job_id : id of the job

dataset_name : Name of the dataset

dataset_version_name : Name of dataset version

dataset_split_name : Name of dataset split

job_status : status of the job

job_created_date : create date of the job

split_percentage: a list of numbers representing ratio of each portion of all desired splits, total must be equal to 100%.

details: details about this job

get_status()

get current status of job.

get_dataset_split()

get the DataSetSplit created from this job.

cancel()

cancel this job.

Class TransformationJob

A TransformationJob represents the process that run transformation over dataset.

__init__ (name, status, details)

Instantiates a TransformationJob object.

Args:

job_name : Name of the job

status : status of the job

details: details of the job

get_status()

get current status of job.

list_jobs(name, description, dataset_name, dataset_version_from_name, dataset_version_to_name, created_date, updated_date, author, status, age**)**

list jobs by the filter.

Args:

name : (optional), job name, partial match allowed.

dataset_name : (optional) dataset name to be searched, partial match allowed

dataset_version_from_name : (optional) source dataset version to be searched, partial match allowed

dataset_version_to_name : (optional) target dataset version to be searched, partial match allowed

created_date : (optional) created date of job

updated_date : (optional) updated date of job

author : (optional) author, partial matched allowed

status: (optional) status of job, partial matched allowed

age : (optional), age of job, only job age is less than this age will be returned

get_job_metrics_details(job_name, partition_index**)**

list job metrics, all fields has default None, all fields are exact match, if no filter field is specified, get_job_metrics_details will return all job entries.

cancel()

cancel this job.

Module sbrain.learning.experiment

Class Estimator

Estimator is a high-level API that simplifies machine learning programming and encapsulate various information related to model training like matrix multiplications, saving checkpoints and so on.

create(estimator_name, description, estimator_obj)

Stores the estimator object in the repository.

Args:

estimator_name: unique name for the estimator.

description: text description of the estimator.

estimator_obj: estimator object created with the NewClassificationEstimator method.

Returns: Instance of created estimator.

lookup(name):

This function allows you to lookup Estimator object by providing name.

Args:

name : Name of the Estimator.

Returns: Instance of estimator which exactly matches the given name.

retrieve(id):

This function allows you to lookup Estimator object by providing estimator id.

Args:

id : Unique Id of the Estimator.

Returns: Instance of estimator for the given id.

list_all():

This function allows to search all Estimators.

Returns: Returns the list of all estimator instance. Also prints them on notebook.

search(estimator_name, description):

This function allows you to search Estimator object by providing estimator name and description.

Args:

estimator_name: Name of the Estimator.

description: Description of the Estimator

Returns: Returns the list of all estimator instance matching the search criteria. Also prints them on notebook.

NewClassificationEstimator(model_fn):

Instantiates an estimator model given a model function

Args:

model_fn : Python function that creates the computational graph with the TensorFlow API. The function must return an object of the class tf.EstimatorSpec.

Returns: Static constructor to create a new instance of classification estimator.

Class HParams

This object allows you to initialize hyper parameters before training the neural network model.

__init__(iterations, batch_size)

Args:

iterations : Number of passes, each pass using [batch size] number of examples.

batch_size : Number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.

Class HParamValues

This object allows you to initialize hyper parameters values.

__init__(name, paramlist)

Args:

name : name of this HParamValues list.

paramlist : parameter list.

Class HyperParamsSpace

This object allows you to initialize hyper parameters space. You can specify the space as either discrete list or as a constant value.

__init__(list_of_hparam_vals)

Args:

list_of_hparam_vals : list of hyper parameter values.

grid_search()

Returns grid search settings for this HyperParamsSpace.

Returns: Returns the HParamsSearch object.

Class HParamsSearch

This object allows you to search hyper parameters.

__init__(list_of_hparam_vals)

Args:

list_of_hparam_vals : list of hyper parameter values.

Class RunConfig

This object allows you to set parameters for running experiments in distributed environment.

__init__(no_of_ps, no_of_workers, summary_save_frequency, run_eval, use_gpun, transfer_learning_config)

Args:

no_of_ps : Number of parameter servers required.

no_of_workers: Number of tensor-flow workers required.

summary_save_frequency: Summary save frequency.

run_eval: Boolean value for run eval.

use_gpu: Boolean value if CPU or GPU is required.

transfer_learning_config: Instance of TransferLearningConfig.

Class TransferLearningConfig

This object allows you to set parameters for transfer learning.

__init__(model_checkpoint, vars_to_load)

Args:

model_checkpoint : model checkpoint instance.

vars_to_load: variables to load.

Class Experiment

Experiment is a high-level API for distributed training. It contains all information needed to train and build models on a local host or on a distributed multi-server environment on CPUs and GPUs.

no_of_jobs()

returns total number of jobs for this experiment.

Returns: Returns the number of jobs as int.

list_jobs()

list all jobs for this experiment with the current state.

Returns: Returns list of job instances.

get_jobs()

gets all jobs for this experiment, but may not be current. Could be stale depending on when this experiment instance was looked up.

Returns: Returns list of job instances.

list_models()

list all models with its current state.

Returns: Returns list of model instances.

get_models()

gets all models, but may not be current. Could be stale depending on when this experiment instance was looked up.

Returns: Returns list of model instances.

list_all()

list all experiments.

Returns: Returns list of experiment instances.

get_single_job()

gets the single job under the experiment if it is of single job type. Else throws error.

Returns: Returns the single job instance.

cancel()

Cancels this experiment and waits for it to exit.

Returns: Void.

request_cancellation()

Requests cancellation of this experiment but does not wait for it exit.

Returns: void.

get_best_model_until_now(key_function)

This function sorts the models under this experiment based on the key function provided and gets the first model under it.

Args:

key_function: The function which takes in the model metric dictionary and returns the key to search on.

Returns: Returns the best model based on the provided key_function.

wait_until_finish(time_out_in_seconds, check_every_n_seconds)

This function allows to wait for experiment until it has finished.

Args:

time_out_in_seconds : time out seconds

check_every_n_seconds: periodic time to check experiment status.

Returns: void.

has_finished()

check if experiment has completed.

Returns: Boolean.

report_status()

Prints out experiment status onto the notebook.

Returns: void.

search(name_filter, description_filter)

searches experiments with given search criteria.

Args:

name_filter: (optional) Name of the experiment to be searched.

Experiments with partially matching names are also returned.

description_filter: (optional) description of the experiments.

Experiments with words in the description matching the input are returned.

Returns: Returns list of experiment instances.

lookup(name)

Looks up a experiment by name. Name should match exactly, its case sensitive.

Args:

name: name of the experiement to look up.

Returns: Returns experiment instance which matches the name exactly.

retrieve(experiment_id)

Looks up an experiment by experiment id. Experiment id is the primary key in the database. It is int type and should be matched exactly

Args:

experiment_id: id of the experiment to look up.

Returns: Returns experiment instance for the given id.

list_all()

list all experiments.

Returns: Returns the list of all experiment instances.

run(experiment_name, description, estimator, run_config, hyper_parameters, hparams_search_settings, dataset_version_split, input_function, transfer_learning_config)

This function allows you to run experiment by providing estimator, hyper parameters and run config.

Args:

experiment_name : Name of the experiment.

description : Description of the experiement.

estimator : Estimator object.

run_config : RunConfig **o**bject.

hyper_parameters : HParams object.

hparams_search_settings: HParamsSearch object.

dataset_version_split : DataSetSplit object.

input_function : Input function.

transfer_learning_config : TransferLearningConfig object.

Returns: Returns the new experiment instance created as part of this run request.

Class LearningJob

LearningJob object allows you to get job status and acquire model information.

get_tensorboard_url()

get the url to access tensor board. If tensorboard is shutdown, a new one will be provided.

Returns: Returns the tensorboard url for this learning job as a string.

is_success()

check job if it succeeds.

Returns: returns boolean indicating whether job is success. Throws error if job is not finished.

is_failure()

check job if it fails.

Returns: returns boolean indicating whether job is failure. Throws error if job is not finished.

is_cancelled()

check job if it is cancelled.

Returns: returns boolean indicating whether job is cancelled. Throws error if job is not finished.

cancel()

Cancels this experiment and waits for it to exit.

Returns: void.

request_cancellation()

Requests cancellation of this experiment but does not wait for it exit.

Returns: void.

search(job_name, description)

searches jobs with given search criteria.

Args:

job_name: (optional) Name of the job to be searched.

Jobs with partially matching names are also returned.

description: (optional) description of the job.

Jobs with words in the description matching the input are returned.

Returns: list of job instances matched by the given search criteria.

lookup(name)

Looks up a job by name. Name should match exactly, its case sensitive.

Args:

name: name of the job to look up.

Returns: Instance of job which exactly matches the given name.

retrieve(id)

Looks up a job by job id. Job id is the primary key in the database. It is int type and should be matched exactly

Args:

id: id of the job to look up.

Returns: Instance of job for the given id.

has_finished()

This function allows to check status of a particular job whether it has finished or not.

Returns: boolean indicating whether job is finished (success, failure or cancelled).

wait_until_finished(time_out_in_seconds, check_every_n_seconds)

This function allows to wait for a particular job until it has finished.

Args:

time_out_in_seconds : time out seconds

check_every_n_seconds: periodic time to check job status.

Returns: void.

get_model()

This function allows to get the current model object.

Returns: returns the model instance for this job.

Class Model

This class represent Model information.

get_result_metrics()

This function allows you to get model metrics.

Returns: returns result metrics as a dictionary.

search(model_name, description)

searches models with given search criteria.

Args:

model_name: (optional) Name of the model to be searched.

models with partially matching names are also returned.

description: (optional) description of the models.

Models with words in the description matching the input are returned.

Returns: returns the list of model instances matching the search criteria.

lookup(name)

Looks up a model by name. Name should match exactly, its case sensitive.

Args:

name: name of the model to look up.

Returns: returns the model instance which matches the name exactly.

retrieve(id)

Looks up a model by id. id is the primary key in the database. It is int type and should be matched exactly

Args:

id: id of the model to look up.

Returns: returns the model instance for the given id.

list_all()

list all models.

Returns: returns the list of all model instances.

submit_inference_job(job_name, description, input_function, output_function, best_model=False, gpu_required=False)

Submits an inference job on this model.

Args:

job_name: Name of the job.

description: Description for the job.

input_function: The function which feeds the input to the tensorflow estimator. This functions gets fed directly to the estimator without any modification.

output_function: Function to handle the output.

best_model: Boolean indicating whether this should run on a best model reference.

gpu_required: Should use gpu for inference job.

Returns: ModelInferenceJob instance.

Class ModelCheckPoint

This class represent model checkpoint information.

get_all_trainable_vars()

This function allows you to get all trainable variables.

Returns: Returns the list of all trainable vars in the checkpoint as a list of strings.

get_all_vars()

This function allows you to get all variables.

Returns: Returns the list of all vars in the checkpoint as a list of strings.

get_passing_regex_vars(regexpr, trainable)

get vars with given search criteria.

Args:

regexpr: regular expression.

vars filtered by regular expression.

trainable: boolean value indicating if they are trainable vars.

Returns: Returns the list of all vars in the checkpoint, which matches the given regular expresssion, as a list of strings.

lookup(name)

Looks up a checkpoint by name. Name should match exactly, its case sensitive.

Args:

name: name of the checkpoint to look up.

Returns: Returns the model checkpoint instance for the given name.

retrieve(checkpoint_id)

Looks up a checkpoint by id. checkpoint id is the primary key in the database. It is int type and should be matched exactly

Args:

checkpoint_id: id of the checkpoint to look up.

Returns: Returns the model checkpoint instance for the given id.

search(name)

searches checkpoints with given search criteria.

Args:

name: (optional) Name of the checkpoints to be searched.

checkpoints with partially matching names are also returned.

description: (optional) description of the checkpoint.

Checkpoints with words in the description matching the input are returned.

Returns: Returns the list of model checkpoint instances for the given criteria.

list_all()

list all checkpoints.

Returns: Returns the list of all model checkpoint instances.

export()

export model checkpoint under the "shared-dir"

Args:

model_id: (optional) id of the model for which you want to export the check point.

model_name: (optional) name of the model for which you want to export the check point.

checkepoint_id: id of the checpoint which you want to export.

checkpoint_name: name of the checkpoint which you want to export.

NOTE: need to provide either the model_id or model_name or checkpoint_id or checkpoint_name.

export_saved_model: Default False. If set to true the model will be exported as Tensorflow saved model format.

serving_input_receiver_func: if the export_saved_model=True, then this argument takes the serving input receiver function to be used.

params: if export_saved_model = True, this argument is a dict that takes any extra parameters to be passed on to the serving input

receiver function.

Returns:

Returns **ModelExportInformation** object with the following fields:

export_dir_name: name of the directory under "shared-dir" where the model checkpoint was exported

checkpoint_id: id of the checkpoint

checkpoint_name: name of the checkpoint

model_id: id of the model if applies.

model_name: name of the model if applies.

saved_model_dir: name of the directory under the export_dir where the model is exported as saved model. This path will be returned only in case export_saved_model=True in the input arguments.

Class ModelEndPoint

This class deploys a model to a REST end point. The REST end point can be called to get predictions.

create(model, endpoint_name, description, number_of_service_replicas, gpu_required)

creates and brings up a REST end point for a model.

Args:

model: model object that was created previously.

endpoint_name: name of end point (must be unique).

description : description of the end point.

number_of_service_replicas: **c**ount of pods to be replicated for load balancing traffic**.**

gpu_required: deploy on gpu capable pods or not.

Returns: Returns the model endpoint instance that got created.

shutdown()

Brings down a specific REST end point.

Returns: void.

lookup(endpoint_name)

Looks up a checkpoint by name. Name should match exactly, its case sensitive.

Args:

endpoint_name: name of the REST end point to look up.

Returns: Returns the model endpoint instance for the given name.

search(endpoint_name, author, description, status, model_name)

searches end points with given search criteria.

Args:

endpoint_name: (optional) Name of the end point to be searched. A partial mask also can be supplied.

author: (optional) author of the end point. A partial mask also can be supplied.

description: (optional) description of the end point. A partial mask also can be supplied.

status: (optional) status of the end point. A partial mask also can be supplied.

model_name: (optional) model name of the end point. A partial mask also can be supplied.

Returns: Returns the list of model endpoints that match the search criteria.

predict(feature_dict)

calls predict on the model endpoint

Args:

feature_dict: dictionary with structure {"features":[<array of datapoints>]}, where the each datapoint could be a base64encoded binary image, or image as numpy array.

Returns:

returns the output prediction outputs defined in the model function for each data point

raw_predict(input_function, output_function)

Calls the predict on the underlying tensorflow estimator without invoking any transformation in SBrain.

Args:

input_function: Function which feeds the input to the estimator.

output_function: Function which serializes the output as a string, which is passed as the response of the raw_predict() call.

Returns:

Returns the same string that output_function produces.

Class ModelInferenceJob

This class represents an inference batch job

Attributes:

model_inference_job_id: Id of the job.

model_inference_job_name: Name of the job.

description: User provided description.

model_id: Model id reference for this job.

status: Status of the job.

created_by_user: user that created the job.

gpu_required: Whether gpu is required.

created_date: Created date.

updated_date: Updated date.

retrieve(job_id)

Retrieves a job for the given job id.

Returns: ModelInferenceJob instance.

Troubleshooting