carpedm Documentation

Release 0.1.7

Neal Digre

May 29, 2018

Contents

CHAPTER 1

Description

CarpeDM is a general library for downloading, viewing, and manipulating image data. Originally developed as a ChARacter shaPE Data Manager, CarpeDM aims to make Japanese character shape () data and other image datasets more accessible to machine learning researchers.

Table 1: Datasets Currently Available for Download

ID	Dataset
pmjtc	Pre-Modern Japanese Text Character Shapes Dataset (), provided by the Center for Open Data in the Humanities (CODH).

Though still in the early stages of development, a high-level interface is also provided for

- Automatic model-ready data generation.
- Flexible training of models with a variety of deep learning frameworks.

Currently supported deep learning frameworks:

- TensorFlow

CHAPTER 2

Documentation

2.1 Installation

2.1.1 Recommended Environments

The following versions of Python can be used: 3.4, 3.5, 3.6.

We recommend setting up a virtual environment with Python 3.6 for using or developing CarpeDM.

We use virtualenv, but you could use Conda, etc.

```
$ virtualenv -p /path/to/python3 ~/.virtualenvs/carpedm
$ source ~/.virtualenvs/carpedm/bin/activate
```

Or, for Conda:

```
$ conda create --name carpedm python=3.6
$ conda activate carpedm
```

Note: CarpeDM is built and tested on MacOS. We cannot guarantee that it works on other environments, including Windows and Linux.

2.1.2 Dependencies

Before installing CarpeDM, we recommend to upgrade `setuptools` if you are using an old one:

```
$ pip install -U setuptools
```

The following Python packages are required to install CarpeDM. The latest version of each package will automatically be installed if missing.

- TensorFlow 1.5+

- Numpy 1.14+
- Pillow 5.1+

The following packages are optional dependencies.

- Plot and images support
  - matplotlib 2.1.2, 2.2.2

2.1.3 Install CarpeDM

Install CarpeDM via pip

We recommend installing the latest release of CarpeDM with pip:

```
$ pip install carpedm
```

Note: Any optional dependencies can be added after installing CarpeDM. Please refer to *Optional Dependencies*.

Install CarpeDM from Source

You can install a development version of CarpeDM from a cloned Git repository:

```
$ git clone https://github.com/SimulatedANeal/carpedm.git
$ cd carpedm
$ python setup.py develop
```

Optional Dependencies

Support Plotting and Viewing Images

Using the following (see *carpedm.data.meta*)

```
MetaLoader.view_images()
MetaLoader.data_stats(which_stats=('frequency'))
```

require `matplotlib`. We recommend installing it with pip:

```
$ pip install matplotlib
```

2.1.4 Uninstall CarpeDM

Use pip to uninstall CarpeDM:

```
$ pip uninstall carpedm
```

2.1.5 Upgrade CarpeDM

Just use `pip` with `-U` option:

```
$ pip install -U carpedm
```

2.1.6 FAQ

2.2 Guides

2.2.1 Basic Usage

Getting Started

There is some sample data provided, accessed as follows:

```
from carpedm.data import sample as PATH_TO_SAMPLE_DATA
```

This small dataset is useful for getting started and debugging purposes.

Full datasets can be downloaded with:

```
$ download_data -d <download/to/this/directory> -i <dataset-id>
```

It may take a while. For a list of available dataset IDs, use:

```
$ download_data -h
```

Exploring the Data

To quickly load and review data for a task, use the *carpedm.data.meta.MetaLoader* class directly. Here are some example datasets that vary each image's scope and the characters included.

```
import carpedm as dm


# Create objects for storing meta data
single_kana = dm.data.MetaLoader(data_dir=dm.data.sample, image_scope='char',␣
↪charset=dm.data.CharacterSet('kana'))
kanji_seq = dm.data.MetaLoader(data_dir=dm.data.sample, image_scope='seq', seq_len=3,␣
↪charset=dm.data.CharacterSet('kanji'))
full_page = dm.data.MetaLoader(data_dir=dm.data.sample, image_scope='page',␣
↪charset=dm.data.CharacterSet('all'))
```

Note that these objects only store the metadata for images in the dataset, so they are relatively time and space efficient. Assuming `matplotlib` is installed (see *Optional Dependencies*), you can use `view_images` to actually load and view images within the dataset. Or use `generate_dataset` to save training data for a machine learning algorithm. For example:

```
single_kana.view_images(subset='train', shape=(64,64))
kanji_seq.view_images(subset='dev', shape=(None, 64))
full_page.view_images(subset='test', shape=None)
```

(continues on next page)

(continued from previous page)

```
# Save the data as TFRecords (default format_store)
single_kana.generate_dataset(out_dir='/tmp/pmjtc_data', subset='train')
```

Note: Currently, `view_images` does not work in a Jupyter notebook instance.

Training a Model

The MetaLoader class on its own is useful for rapid data exploration, but the *Tasks* module provides a high-level interface for the entire training pipeline, from loading the raw data and automatically generating model-ready datasets, to actually training and evaluating a model.

Next, we will walk through a simple example that uses the provided single character recognition task and a simple baseline Convolutional Neural Network model.

First, let's set our TensorFlow verbosity so we can see the training progress.

```
import tensorflow as tf

tf.logging.set_verbosity(tf.logging.INFO)
```

Next, we'll initialize our single kana recognition task

```
import carpedm as dm

# Task definition
args = {'data_dir': dm.data.sample,
        'task_dir': '/tmp/carpedm_tasks',
        'shape_store': None,
        'shape_in': (64, 64)}
task = registry.task('ocr_single_kana')(**args)
```

Most of the Task functionality, such as the target `character_set`, `sequence_length` (if we're looking at character sequences `image_scope == 'seq'`), or `loss_fn` is encapsulated in the class definition. However, there are some **REQUIRED** run-time task arguments: `data_dir` and `task_dir` tell the task where to find the raw data, and where to store task-specific data/results, respectively. The other **optional** run-time arguments `shape_store` and `shape_in` determine the size of images when they are stored on disk and fed into our neural network, respectively. If `shape_store` or `shape_in` are not provided, the original image size is used.

Caution: Using the default for `shape_in` may break a model expecting fixed-size input.

For more information and a full list of optional arguments, please refer to the *Tasks* API.

A task can be accessed from the registry with the appropriate task ID. By default, the ID for a stored task is a "snake_cased" version of the task class name. Custom tasks can be added to the registry using the `@registry.register_model` decorator, importing the new class in `tasks.__init__`, and importing `carpedm`, more specifically, the `carpedm.tasks` package.

Now let's define our hyper-parameters for training and our model.

```
from carpedm.util import registry
```

(continues on next page)

(continued from previous page)

```
# Training Hyperparameters
num_epochs = 30
training_hparams = {'train_batch_size': 32,
                    'eval_batch_size': 1,
                    'data_format': 'channels_last',
                    'optimizer': 'sgd',
                    'learning_rate': 1e-3,
                    'momentum': 0.96,
                    'weight_decay': 2e-4,
                    'gradient_clipping': None,
                    'lr_decay_steps': None,
                    'init_dir': None,  # for pre-trained models
                    'sync': False}

# Model hyperparameters and definition
model_hparams = {}
model = registry.model('single_char_baseline')(num_classes=task.num_classes, **model_
↪hparams)
```

The `training_hparams` above represent the minimal set that *must* be defined for training to run. In practice, you may want to use a tool like argparse and define some defaults so you don't have to explicitly define each one manually every time. Accessing and registering models is similar to the process for tasks (see here for more details).

The `baseline_cnn` model is fully defined except for the number of classes to predict, so it doesn't take any hyperparameters.

To distinguish this model from others, we should define a unique `job_id`, which can then be used in some boilerplate TensorFlow configuration.

```
# Unique job_id
experiment_id = 'example'
shape = re.sub(r'([,])', '_', re.sub(r'([() ])', '', str(args['shape_in'])))
job_id = os.path.join(experiment_id, shape, model.name)
task.job_id = job_id  # Used to check for first model initialization.
job_dir = os.path.join(task.task_log_dir, job_id)

# TensorFlow Configuration
sess_config = tf.ConfigProto(
    allow_soft_placement=True,
    log_device_placement=False,
    intra_op_parallelism_threads=0,
    gpu_options=tf.GPUOptions(force_gpu_compatible=True))
config = tf.estimator.RunConfig(session_config=sess_config,
                                model_dir=job_dir,
                                save_summary_steps=10)
hparams = tf.contrib.training.HParams(is_chief=config.is_chief,
                                      **training_hparams)
```

We include `shape_in` in the job ID to avoid conflicts with loading models meant for images of different sizes. Although we don't do so here for simplicity, it would also be a good idea to include training hyperparameter settings in the job ID, as those are not represented in `model.name`.

Now comes the important part: defining the input and model functions used by a TensorFlow Estimator.

```
# Input and model functions
train_input_fn = task.input_fn(hparams.train_batch_size,
                               subset='train',
```

(continues on next page)

(continued from previous page)

```
                                num_shards=1,
                                overwrite=False)
eval_input_fn = task.input_fn(hparams.eval_batch_size,
                              subset='dev',
                              num_shards=1,
                              overwrite=False)
model_fn = task.model_fn(model, num_gpus=0, variable_strategy='CPU',
                         num_workers=config.num_worker_replicas or 1)
```

As we can see, the Task interface makes this extremely easy! The appropriate data subset for the task is generated (and saved) *once* automatically when `task.input_fn` is called. You can overwrite previously saved data by setting the `overwrite` parameter to True. The `num_shards` parameter can be used for training in parallel, e.g. on multiple GPUs.

`model_fn` is a bit more complicated under the hood, but its components are simple:

- It uses `model.forward_pass` to generate predictions,
- `task.loss_fn` to train the model
- and `task.results` for compiling results.

I don't assume access to any GPUs, hence the values for `num_gpus` and `variable_strategy`. `variable_strategy` tells the training manager where to collect and update variables. You can ignore the `num_workers` parameter, unless you want to use special distributed training, e.g. on Google Cloud.

Note: The `input_fn` definitions must come before the `model_fn` definition because `model_fn` relies on a variable, `original_format`, defined in `input_fn`. This dependence will likely be removed in future versions.

We're almost ready to train. We just need to tell it how long to train,

```
# Number of training steps
train_examples = dm.data.num_examples_per_epoch(task.task_data_dir, 'train')
eval_examples = dm.data.num_examples_per_epoch(task.task_data_dir, 'dev')

if eval_examples % hparams.eval_batch_size != 0:
    raise ValueError(('validation set size (%d) must be multiple of '
                      'eval_batch_size (%d)') % (eval_examples,
                                                 hparams.eval_batch_size))

eval_steps = eval_examples // hparams.eval_batch_size
train_steps = num_epochs * ((train_examples // hparams.train_batch_size) or 1)
```

define our training manager,

```
estimator = tf.estimator.Estimator(model_fn=model_fn, config=config, params=hparams)
```

and hit the train button!

```
tf.estimator.train_and_evaluate(estimator, train_spec=train_spec, eval_spec=eval_spec)
```

Putting it all together, we have a very minimal *main.py* module for training models. Running it took **8 minutes** on a MacBook Pro, which includes data generation and training the model. At the end of 30 epochs, it achieved a development set accuracy of **65.27%**. Not great, but this example only uses the small sample dataset (1,447 training examples). And considering the **70** character classes and **4.19%** majority class for this task and specific dataset, we are already doing much better than chance!

Running this same code for the *full* currently available PMJTC dataset takes much longer but—as you would expect when adding more data—achieves a higher accuracy (see *Benchmarks*). Though certainly indicative of the benefit of more data, note that the accuracies presented in the benchmarks are not a fair comparison to the one above for two reasons:

1. There are more kana character classes in the full dataset: **131**
2. The development sets on which accuracies are reported are different.

Conclusion

I hope that this guide has introduced the basics of using CarpeDM and encourages you to define your own models and tasks, and conduct enriching research on Pre-modern Japanese Text Characters and beyond!

Seize the Data Manager!

2.3 Examples

2.3.1 Single Character Task

Below is an example Task definition for a single character recognition task and the corresponding import in `__init__.py` for accessing the task through the registry.

For more details on Task definition and default properties, please refer to the *Tasks* documentation.

ocr.py

```
#
# Copyright (C) 2018 Neal Digre.
#
# This software may be modified and distributed under the terms
# of the MIT license. See the LICENSE file for details.


"""Optical character recognition tasks.

TODO:
    * Modularize common loss functions, select by id
    * Modularize common regularization options, select by id
"""
import abc

import tensorflow as tf

from carpedm.data.lang import JapaneseUnicodes
from carpedm.tasks.generic import Task
from carpedm.util import registry
from carpedm.util.eval import confusion_matrix_metric


class OCRTask(Task):
    """Abstract class for OCR Tasks."""

    def __init__(self, **kwargs):
```

(continues on next page)

(continued from previous page)

```
        super(OCRTask, self).__init__(**kwargs)

    @property
    def target(self):
        return 'image/seq/char/id'

    @property
    def blocks(self):
        return False

    @property
    def character(self):
        return True

    @property
    def line(self):
        return False

    @property
    def label(self):
        return True

    @property
    def bbox(self):
        return False

    @property
    @abc.abstractmethod
    def sparse_labels(self):
        return False

    def regularization(self, hparams):
        raise NotImplementedError

    def results(self, loss, tower_features, tower_preds, tower_targets,
                is_training):
        raise NotImplementedError

    def loss_fn(self, features, model_output, targets, is_training):
        raise NotImplementedError


@registry.register_task
class OCRSingleKana(OCRTask):
    """Single character recognition tasks."""

    @property
    def image_scope(self):
        return 'char'

    @property
    def character_set(self):
        return JapaneseUnicodes('kana')

    def results(self, loss, tower_features, tower_preds, tower_targets,
                is_training):
        tensors_to_log = {'loss': loss}
```

(continues on next page)

(continued from previous page)

```
        tf.summary.image("sample_input", tower_features[0]['image/data'])

        all_logits = tf.concat([p for p in tower_preds], axis=0)
        predictions = {
            'classes': tf.argmax(all_logits, axis=1),
            'probabilities': tf.nn.softmax(all_logits)
        }

        stacked_labels = tf.squeeze(tf.concat(tower_targets, axis=0))

        accuracy = tf.metrics.accuracy(stacked_labels, predictions['classes'])
        metrics = {
            'accuracy': accuracy,
            'confusion': confusion_matrix_metric(
                stacked_labels, predictions['classes'], self.num_classes)
        }

        return tensors_to_log, predictions, metrics

    def loss_fn(self, features, model_output, targets, is_training):
        with tf.name_scope('batch_xentropy'):
            loss = tf.losses.sparse_softmax_cross_entropy(
                logits=model_output, labels=targets)
        return loss

    def regularization(self, hparams):
        model_params = tf.trainable_variables()
        weight_loss = tf.multiply(
            hparams.weight_decay,
            tf.add_n([tf.nn.l2_loss(v) for v in model_params]),
            name='weight_loss')
        return weight_loss

    @property
    def sparse_labels(self):
        return False


@registry.register_task
class OCRSeqKana3(OCRTask):

    def __init__(self, beam_width=100, **kwargs):
        self._beam_width = beam_width
        super(OCRSeqKana3, self).__init__(**kwargs)

    @property
    def character_set(self):
        return JapaneseUnicodes('kana')

    @property
    def image_scope(self):
        return 'seq'

    @property
    def sequence_length(self):
        return 3
```

(continues on next page)

(continued from previous page)

```
    @property
    def sparse_labels(self):
        return True

    @property
    def target(self):
        return 'image/seq/char/id_sparse'

    def loss_fn(self, features, model_output, targets, is_training):
        return tf.nn.ctc_loss(labels=targets,
                              inputs=model_output['logits'],
                              sequence_length=model_output['seq_len'],
                              time_major=False)

    def results(self, loss, tower_features, tower_preds, tower_targets,
                is_training):

        tf.summary.image("sample_input", tower_features[0]['image/data'])

        all_logits = tf.concat([p['logits'] for p in tower_preds], axis=0)
        seq_lens = tf.concat([p['seq_len'] for p in tower_preds], axis=0)

        # TODO: fix when seqs are different lengths from multiple GPUs
        all_labels = tf.sparse_concat(0, [p for p in tower_targets])
        decoded, log_prob = tf.nn.ctc_beam_search_decoder(
            inputs=tf.transpose(all_logits, [1, 0, 2]),
            sequence_length=seq_lens,
            beam_width=self._beam_width)
        decoded = decoded[0]  # best path

        edit_distance = tf.edit_distance(decoded, tf.to_int64(all_labels),
                                         normalize=False)

        Z = tf.cast(tf.size(all_labels), tf.float32)
        ler = tf.reduce_sum(edit_distance) / Z
        S = tf.cast(tf.size(edit_distance), tf.float32)
        num_wrong_seqs = tf.cast(tf.count_nonzero(edit_distance), tf.float32)
        ser = num_wrong_seqs / S

        metrics = {
            'ler': tf.metrics.mean(ler),
            'ser': tf.metrics.mean(ser)
        }

        tensors_to_log = {'loss': loss, 'ler': ler, 'ser': ser}

        mapping_string = tf.constant(self._meta.vocab.types())
        table = tf.contrib.lookup.index_to_string_table_from_tensor(
            mapping_string, default_value='NULL')
        decoding = table.lookup(tf.to_int64(tf.sparse_tensor_to_dense(decoded)))
        gt = table.lookup(tf.to_int64(tf.sparse_tensor_to_dense(all_labels)))

        tf.summary.text('decoded', decoding)
        tf.summary.text('gt', gt)

        predictions = {
```

(continues on next page)

(continued from previous page)

```
            'classes': tf.argmax(input=all_logits, axis=1),
            'probabilities': tf.nn.softmax(all_logits),
            'decoded': decoding,
        }

        return tensors_to_log, predictions, metrics

    def regularization(self, hparams):
        model_params = tf.trainable_variables()
        weight_loss = tf.multiply(
            hparams.weight_decay,
            tf.add_n([tf.nn.l2_loss(v) for v in model_params]),
            name='weight_loss')
        return weight_loss
```

tasks.__init__.py

```
#
# Copyright (C) 2018 Neal Digre.
#
# This software may be modified and distributed under the terms
# of the MIT license. See the LICENSE file for details.

from carpedm.tasks import generic

# Defined tasks. Imports here force registration.
from carpedm.tasks.ocr import OCRSingleKana
```

2.3.2 Baseline Model

baseline.py

```
#
# Copyright (C) 2018 Neal Digre.
#
# This software may be modified and distributed under the terms
# of the MIT license. See the LICENSE file for details.

"""Baseline models."""

import tensorflow as tf

from carpedm.models.generic import TFModel
from carpedm import nn
from carpedm.util import registry


@registry.register_model
class SingleCharBaseline(TFModel):
    """A simple baseline CNN model."""

    def __init__(self, num_classes, *args, **kwargs):
```

(continues on next page)

(continued from previous page)

```
        """Initializer.

        Overrides TFModel.

        Args:
            num_classes: Number of possible character classes.
            *args: Unused arguments.
            **kwargs: Unused arguments.

        """
        self._num_classes = num_classes
        self._cnn = nn.conv.CNN()

    @property
    def name(self):
        return "Baseline_" + self._cnn.name

    def _forward_pass(self, features, data_format, axes_order,
                      is_training, reuse):
        x = features['image/data']
        x = self._cnn.forward_pass(
            x, data_format, axes_order, is_training, False, reuse)
        x = tf.layers.flatten(x)
        tf.logging.info('image after flatten: %s', x.get_shape())

        x = tf.layers.dense(
            inputs=x, units=200, activation=tf.nn.relu, name='dense1')
        nn.util.activation_summary(x)
        x = tf.layers.dense(
            inputs=x, units=200, activation=tf.nn.relu, name='dense2')
        nn.util.activation_summary(x)
        logits = tf.layers.dense(
            inputs=x, units=self._num_classes, name='logits')
        return logits


@registry.register_model
class SequenceBaseline(TFModel):
    """A simple baseline CNN-LSTM model."""

    def __init__(self, num_classes, lstm_layers=2, lstm_units=100,
                 feature_extractor=nn.conv.CNN(), *args, **kwargs):
        """Initializer.

        Overrides TFModel.

        Args:
            num_classes (int): Number of possible character classes.
            lstm_layers (int): Number of LSTM layers.
            lstm_unit (int): Number of units in LSTM cell
            feature_extractor:
            *args: Unused arguments.
            **kwargs: Unused arguments.
        """
        self._num_classes = num_classes + 1  # Add CTC null label.
        self._layers = lstm_layers
        self._units = lstm_units
```

(continues on next page)

(continued from previous page)

```
        self._feature_extractor = feature_extractor

    @property
    def name(self):
        return 'Baseline_seq_' + self._feature_extractor.name

    def _forward_pass(self, features, data_format, axes_order,
                      is_training, reuse):
        x = self._feature_extractor.forward_pass(
            features['image/data'], data_format, axes_order,
            is_training, False, reuse)
        if axes_order == [0, 3, 1, 2]:
            x = tf.transpose(x, [0, 2, 3, 1])
        x = tf.reshape(x, [-1, x.shape[1], x.shape[2] * x.shape[3]])
        x = nn.rnn.bi_lstm(x, n_layers=self._layers, n_units=self._units)
        seq_len = tf.tile(tf.expand_dims(tf.to_int32(tf.shape(x)[1]), 0),
                          [tf.to_int32(tf.shape(x)[0])])
        logits = tf.layers.dense(inputs=x, units=self._num_classes)

        return {'logits': logits, 'seq_len': seq_len}

    def initialize_pretrained(self, pretrained_dir):

        submodel = 'Baseline_' + self._feature_extractor.name

        variable_mapping = dict()

        for i in range(5):
            variable_mapping[submodel + '/conv{}/'.format(i)] \
                = self.name + '/conv{}/'.format(i)

        return variable_mapping
```

models.__init__.py

```
#
# Copyright (C) 2018 Neal Digre.
#
# This software may be modified and distributed under the terms
# of the MIT license. See the LICENSE file for details.

from carpedm.models import generic

# Defined models. Imports here force registration.
from carpedm.models.baseline import SingleCharBaseline
```

2.3.3 Using Tasks and Models

Below is a minimal `main.py` example for getting started training a model using the Task interface. For an in-depth description, please refer to the guide *Training a Model*.

```
#
# Copyright (C) 2018 Neal Digre.
```

(continues on next page)

(continued from previous page)

```
#
# This software may be modified and distributed under the terms
# of the MIT license. See the LICENSE file for details.

"""Minimal main module.

If this file is changed, please also change the ``:lines:`` option in
the following files where this code is referenced with the
``literalinclude`` directive.

    * ../guides/usage.rst

"""
import os
import re

import tensorflow as tf

import carpedm as dm
from carpedm.util import registry


tf.logging.set_verbosity(tf.logging.INFO)

# Task definition
args = {'data_dir': dm.data.sample,
        'task_dir': '/tmp/carpedm_tasks',
        'shape_store': None,
        'shape_in': (64, 64)}
task = registry.task('ocr_single_kana')(**args)

# Training Hyperparameters
num_epochs = 30
training_hparams = {'train_batch_size': 32,
                    'eval_batch_size': 1,
                    'data_format': 'channels_last',
                    'optimizer': 'sgd',
                    'learning_rate': 1e-3,
                    'momentum': 0.96,
                    'weight_decay': 2e-4,
                    'gradient_clipping': None,
                    'lr_decay_steps': None,
                    'init_dir': None,  # for pre-trained models
                    'sync': False}

# Model hyperparameters and definition
model_hparams = {}
model = registry.model('single_char_baseline')(num_classes=task.num_classes, **model_
↪hparams)

# Unique job_id
experiment_id = 'example'
shape = re.sub(r'([,])', '_', re.sub(r'([() ])', '', str(args['shape_in'])))
job_id = os.path.join(experiment_id, shape, model.name)
task.job_id = job_id  # Used to check for first model initialization.
job_dir = os.path.join(task.task_log_dir, job_id)
```

(continues on next page)

(continued from previous page)

```
# TensorFlow Configuration
sess_config = tf.ConfigProto(
    allow_soft_placement=True,
    log_device_placement=False,
    intra_op_parallelism_threads=0,
    gpu_options=tf.GPUOptions(force_gpu_compatible=True))
config = tf.estimator.RunConfig(session_config=sess_config,
                                model_dir=job_dir,
                                save_summary_steps=10)
hparams = tf.contrib.training.HParams(is_chief=config.is_chief,
                                      **training_hparams)

# Input and model functions
train_input_fn = task.input_fn(hparams.train_batch_size,
                               subset='train',
                               num_shards=1,
                               overwrite=False)
eval_input_fn = task.input_fn(hparams.eval_batch_size,
                              subset='dev',
                              num_shards=1,
                              overwrite=False)
model_fn = task.model_fn(model, num_gpus=0, variable_strategy='CPU',
                         num_workers=config.num_worker_replicas or 1)

# Number of training steps
train_examples = dm.data.num_examples_per_epoch(task.task_data_dir, 'train')
eval_examples = dm.data.num_examples_per_epoch(task.task_data_dir, 'dev')

if eval_examples % hparams.eval_batch_size != 0:
    raise ValueError(('validation set size (%d) must be multiple of '
                      'eval_batch_size (%d)') % (eval_examples,
                                                 hparams.eval_batch_size))

eval_steps = eval_examples // hparams.eval_batch_size
train_steps = num_epochs * ((train_examples // hparams.train_batch_size) or 1)

train_spec = tf.estimator.TrainSpec(input_fn=train_input_fn, max_steps=train_steps)
eval_spec = tf.estimator.EvalSpec(input_fn=eval_input_fn, steps=eval_steps)

# Estimator definition and training
estimator = tf.estimator.Estimator(model_fn=model_fn, config=config, params=hparams)
tf.estimator.train_and_evaluate(estimator, train_spec=train_spec, eval_spec=eval_spec)
```

2.4 API

2.4.1 Data

carpedm.data.download

Download scripts.

This module provides the interface for downloading raw datasets from their source.

Table 1: Datasets Currently Available for Download

ID	Dataset
pmjtc	Pre-Modern Japanese Text Character Shapes Dataset (), provided by the Center for Open Data in the Humanities (CODH).

Example

Data may be downloaded externally using the provided script:

```
$ download_data --data-dir <download/to/this/directory> --data-id pmjtc
```

Note: If an expected data subdirectory already exists in the specified target `data-dir` that data will not be downloaded, even if the subdirectory is empty. This should be fixed in a future version.

Todo:

- Update `get_books_list` once list is included in downloadables.
- Check subdirectory contents.
- Generalize download structure for other datasets.

`carpedm.data.download.`**`get_books_list`**(*dataset='pmjtc'*)
Retrieve list of books/images in dataset.

Parameters **`dataset`** (*str*) – Identifier for dataset for which to retrieve information.

Returns Names of dataset subdirectories and/or files.

Return type `list` of `str`

`carpedm.data.download.`**`maybe_download`**(*directory*, *dataset='pmjtc'*)
Download character dataset if BOOKS not in directory.

Parameters

- **`directory`** (*str*) – Directory where dataset is located or should be saved.
- **`dataset`** (*str*) – Identifier for dataset to download.

carpedm.data.io

Input and output.

This module provides functionality for reading and writing data.

Todo:

- **Tests**
  - DataWriter

– CSVParser

class carpedm.data.io.**CSVParser**(*csv_file*, *data_dir*, *bib_id*)

Utility class for parsing coordinate CSV files.

character(*row*)

Convert CSV row to a Character object.

Returns The next character

Return type *Character*

characters()

Generates rest of characters in CSV.

Yields *carpedm.data.util.Character* – The next character.

parse_characters(*charset*)

Generate metadata for single character images.

Parameters **charset** (CharacterSet) – Character set.

A more efficient implementation of parse_sequences when image_scope='seq' and seq_len=1.

Only characters in the character set are included.

Returns Single character image meta data.

Return type list of *carpedm.data.util.ImageMeta*

parse_lines()

Generate metadata for vertical lines of characters.

Characters not in character set or vocabulary will be labeled as unknown when converted to integer IDs.

Returns Line image meta data.

Return type list of *carpedm.data.util.ImageMeta*

parse_pages()

Genereate metadata for full page images.

Includes every character on page. Characters not in character set or vocabulary will be labeled as unknown when converted to integer IDs.

Returns Page image meta data.

Return type list of *carpedm.data.util.ImageMeta*

parse_sequences(*charset*, *len_min*, *len_max*)

Generate metadata for images of character sequences.

Only includes sequences of chars in the desired character set. If len_min == len_max, sequence length is deterministic, else each sequence is of random length from [len_min, len_max].

Parameters

- **charset** (CharacterSet) – The character set.
- **len_min** (*int*) – Minimum sequence length.
- **len_max** (*int*) – Maximum sequence length.

Returns Sequence image meta data.

Return type list of *carpedm.data.util.ImageMeta*

class carpedm.data.io.**DataWriter**(*format_out*, *images*, *image_shape*, *vocab*, *chunk*, *character*, *line*, *label*, *bbox*, *subdirs*)

Utility for writing data to disk in various formats.

available_formats
list – The available formats.

References

Heavy modification of `_process_dataset` in the input pipeline for the TensorFlow *im2txt* models.

write(*fname_prefix*, *num_threads*, *num_shards*)
Write data to disk.

Parameters

- **fname_prefix** (*str*) – Path base for data files.
- **num_threads** (*int*) – Number of threads to run in parallel.
- **num_shards** (*int*) – Total number of shards to write, if any.

Returns Total number of examples written.

Return type int

carpedm.data.lang

Language-specific and unicode utilities.

Todo:

- Variable UNK token in Vocabulary

---

class carpedm.data.lang.**CharacterSet**(*charset*, *name=None*)
Character set abstract class.

in_charset(*unicode*)
Check if a character is in the defined character set.

Parameters **unicode** (*str*) – String representation of unicode value.

presets
Pre-defined character sets.

Returns Character set IDs.

Return type `list` of `str`

class carpedm.data.lang.**JapaneseUnicodes**(*charset*)
Utility for accessing and manipulating Japanese character unicodes.

Inherits from *CharacterSet*.

Unicode ranges taken from [1] with edits for exceptions.

References

[1] http://www.unicode.org/charts/

presets()
Pre-defined character sets.

Returns Character set IDs.

Return type list of str

class carpedm.data.lang.**Vocabulary**(*reserved*, *vocab*)
Simple vocabulary wrapper.

References

Lightly modified TensorFlow "im2txt" Vocabulary.

char_to_id(*char*)
Returns the integer id of a character string.

get_num_classes()
Returns number of classes, includes <UNK>.

get_num_reserved()
Returns number of reserved IDs.

id_to_char(*char_id*)
Returns the character string of a integer id.

carpedm.data.lang.**char2code**(*unicode*)
Returns the ASCII code for a unicode character.

Parameters **unicode** (*str*) –

Raises TypeError – string is length two.

carpedm.data.lang.**code2char**(*code*)
Returns the unicode string for the character.

carpedm.data.lang.**code2hex**(*code*)
Returns hex integer for a unicode string.

The argument code could either be an ascii representation, (e.g. U+3055, <UNK>) or a unicode character.

Parameters **code** (*str*) – Code to convert.

Returns

Return type int

carpedm.data.meta

Image metadata management.

This module loads and manages metadata stored as CSV files in the raw data directory.

carpedm.data.meta.**DEFAULT_SEED**
int – The default random seed.

Examples

```
import carpedm as dm
```

Load, view, and generate a dataset of single kana characters.

```
single_kana = dm.data.MetaLoader(data_dir=dm.data.sample, image_scope='char',␣
↪charset=dm.data.CharacterSet('kana'))
single_kana.view_images(subset='train', shape=(64,64))
single_kana.generate_dataset(out_dir='/tmp/pmjtc_data', subset='train')
```

Load and view a dataset of sequences of 3 kanji.

```
kanji_seq = dm.data.MetaLoader(data_dir=dm.data.sample, image_scope='seq', seq_len=3,␣
↪charset=dm.data.CharacterSet('kanji'))
kanji_seq.view_images(subset='dev', shape=(None, 64))
```

Load and view a dataset of full pages.

```
full_page = dm.data.MetaLoader(data_dir=dm.data.sample, image_scope='page',␣
↪charset=dm.data.CharacterSet('all'))
full_page.view_images(subset='test', shape=None)
```

Note: Unless stated otherwise, image shape arguments in this module should be a tuple (height, width). Tuple values may be one of the following:

1. `int` specifies the absolute size (in pixels) for that axis
2. `float` specifies a rescale factor relative to the original image size
3. `None` the corresponding axis size will be computed such that the aspect ratio is maintained. If both height and width are *None*, no resize is performed.

Caution: If the new shape is smaller than the original, information will be lost due to interpolation.

Todo:

- **Tests**
  - generate_dataset
- Sort characters by reading order, i.e. character ID.
- Rewrite data as CSV following original format
- Data generator option instead of writing data.
- **Output formats and/or generator return types for `generate_dataset`**
  - numpy
  - hdf5
  - pandas DataFrame
- Chunked `generate_dataset` option to include partial characters.

- **Low-priority:**
  - Fix bounding box display error in `view_images`
  - **specify number of character type in sequence**
    - e.g. 2 Kanji, 1 kana
  - Instead of padding, fill specified shape with surrounding

---

class carpedm.data.meta.**MetaLoader**(*data_dir*, *test_split='hnsd00000'*, *dev_split=0.1*, *dev_factor=1*, *vocab_size=None*, *min_freq=0*, *reserved=('<PAD>', '<GO>', '<END>', '<UNK>')*, *charset=<carpedm.data.lang.JapaneseUnicodes object>*, *image_scope='char'*, *seq_len=None*, *seq_maxlen=None*, *verbose=False*, *seed=None*)

Class for loading image metadata.

data_stats(*which_sets=('train', 'dev', 'test')*, *which_stats=('majority', 'frequency', 'unknowns')*, *save_dir=None*, *include=(None, None)*)

Print or show data statistics.

Parameters

- **which_sets** (*tuple*) – Data subsets to see statistics for.
- **which_stats** (*tuple*) – Statistics to view. Default gives all options.
- **save_dir** (*str*) – If not None, save figures/files to this directory.
- **include** (*tuple*) – Include class IDs from this range.

generate_dataset(*out_dir*, *subset*, *format_store='tfrecords'*, *shape_store=None*, *shape_in=None*, *num_shards=8*, *num_threads=4*, *target_id='image/seq/char/id'*, *sparse_labels=False*, *chunk=False*, *character=True*, *line=False*, *label=True*, *bbox=False*, *overwrite=False*)

Generate data usable by machine learning algorithm.

Parameters

- **out_dir** (*str*) – Directory to write the data to if 'generator' not in `format_store`.
- **subset** (*str*) – The subset of data to generate.
- **format_store** (*str*) – Format to save the data as.
- **shape_store** (*tuple or None*) – Size to which images are resized for storage (on disk). The default is to not perform any resize. Please see this *note on image shape* for more information.
- **shape_in** (*tuple or None*) – Size to which images are resized by interpolation or padding before being input to a model. Please see this *note on image shape* for more information.
- **num_shards** (*int*) – Number of sharded output files.
- **num_threads** (*int*) – Number of threads to run in parallel.
- **target_id** (*str*) – Determines the target feature (one of keys in dict returned by ImageMeta.generate_features).
- **sparse_labels** (*bool*) – Provide sparse_labels, only used for TFRecords.

- **chunk** (*bool*) – Instead of using the original image, extract non-overlapping chunks and corresponding features from the original image on a regular grid. Pad the original image to divide by `shape` evenly.

  Note: Currently only characters that fit entirely in the block will be propagated to appropriate features.

- **character** (*bool*) – Include character info, e.g. label, bbox.
- **line** (*bool*) – Include line info (bbox) in features.
- **label** (*bool*) – Include label IDs in features.
- **bbox** (*str or None*) – If not None, include bbox in features as unit (e.g. 'pixel', 'ratio' [of image]))
- **overwrite** (*bool*) – Overwrite any existing data.

Returns Object for accessing batches of data.

Return type *carpedm.data.providers.DataProvider*

max_image_size(*subset*, *static_shape=(None*, *None)*)
Retrieve the maximum image size (in pixels).

Parameters

- **subset** (*str or None*) – Data subset from which to get image sizes. If None, return max sizes of all images.
- **static_shape** (`tuple` of `int`) – Define static dimensions. Axes that are None will be of variable size.

Returns Maximum size (height, width)

Return type tuple

view_images(*subset*, *shape=None*)
View and explore images in a data subset.

Parameters

- **subset** (*str*) – The subset to iterate through. One of {'train', 'dev', 'test'}.
- **shape** (*tuple or None*) – Shape to which images are resized. Please see this *note on image shape* for more information.

carpedm.data.meta.**num_examples_per_epoch**(*data_dir*, *subset*)
Retrieve number of examples per epoch.

Parameters

- **data_dir** (*str*) – Directory where processed dataset is stored.
- **subset** (*str*) – Data subset.

Returns Number of examples.

Return type int

carpedm.data.ops

Data operations.

This module contains several non-module-specific data operations.

Todo:

- **Tests**
  - `to_sequence_example`, `parse_sequence_example`
  - `sparsify_label`
  - `shard_batch`
  - `same_line`
  - `ixs_in_region`
  - `seq_norm_bbox_values`

`carpedm.data.ops.`**`in_line`**(*xmin_line*, *xmax_line*, *ymin_line*, *xmin_new*, *xmax_new*, *ymax_new*)

Heuristic for determining whether a character is in a line.

Note: Currently dependent on the order in which characters are added. For example, a character may vertically overlap with a line, but adding it to the line would be out of reading order. This should be fixed in a future version.

Parameters

- **xmin_line** (`list` of `int`) – Minimum x-coordinate of characters in the line the new character is tested against.
- **xmax_line** (`list` of `int`) – Maximum x-coordinate of characters in the line the new character is tested against.
- **ymin_line** (*int*) – Minimum y-coordinate of line the new character is tested against.
- **xmin_new** (*int*) – Minimum x-coordinate of new character.
- **xmax_new** (*int*) – Maximum x-coordinate of new character.
- **ymax_new** (*int*) – Maximum y-coordinate of new character.

Returns The new character vertically overlaps with the "average" character in the line.

Return type bool

`carpedm.data.ops.`**`in_region`**(*obj*, *region*, *entire=True*)

Test if an object is in a region.

Parameters

- **obj** (*tuple or* `BBox`) – Object bounding box (xmin, xmax, ymin, ymax) or point (x, y).
- **region** (*tuple or* `BBox`) – Region (xmin, xmax, ymin, ymax).
- **entire** (*bool*) – Object is entirely contained in region.

Returns Result

Return type bool

carpedm.data.ops.**ixs_in_region**(*bboxes*, *y1*, *y2*, *x1*, *x2*)
Heuristic for determining objects in a region.

Parameters

- **bboxes** (list of *carpedm.data.util.BBox*) – Bounding boxes for object boundaries.
- **y1** (*int*) – Top (lowest row index) of region.
- **y2** (*int*) – Bottom (highest row index) of region.
- **x1** (*int*) – left side (lowest column index) of region.
- **x2** (*int*) – right side (highest column index) of region.

Returns Indices of objects inside region.

Return type list of int

carpedm.data.ops.**parse_sequence_example**(*serialized*)
Parse a sequence example.

Parameters **serialized** (tf.Tensor) – Serialized 0-D tensor of type string.

Returns Dictionary of features.

Return type dict

carpedm.data.ops.**seq_norm_bbox_values**(*bboxes*, *height*, *width*)
Sequence and normalize bounding box values.

Parameters

- **bboxes** (list of *carpedm.data.util.BBox*) – Bounding boxes to process.
- **width** (*int*) – Width (in pixels) of image bboxes are in.
- **height** (*int*) – Height (in pixels) of image bboxes are in.

Returns

tuple containing:

list of float: Normalized minimum x-values

list of float: Normalized minimum y-values

list of float: Normalized maximum x-values

list of float: Normalized maximum y-values

Return type tuple

carpedm.data.ops.**shard_batch**(*features*, *labels*, *batch_size*, *num_shards*)
Shard a batch of examples.

Parameters

- **features** (*dict*) – Dictionary of features.
- **labels** (tf.Tensor) – labels
- **batch_size** (*int*) – The batch size.
- **num_shards** (*int*) – Number of shards into which batch is split.

Returns Features as a list of dictionaries.

Return type list of dict

carpedm.data.ops.**sparsify_label**(*label*, *length*)
Convert a regular Tensor into a SparseTensor.

Parameters

- **label** (tf.Tensor) – The label to convert.
- **length** (tf.Tensor) – Length of the label

Returns tf.SparseTensor

carpedm.data.ops.**to_sequence_example**(*feature_dict*)
Convert features to TensorFlow SequenceExample.

Parameters **feature_dict** (*dict*) – Dictionary of features.

Returns tf.train.SequenceExample

carpedm.data.preproc

Preprocessing methods.

This module provides methods for preprocessing images.

Todo:

- **Tests**
  - convert_to_grayscale
  - normalize
  - pad_borders
- Fix and generalize distort_image

---

carpedm.data.preproc.**convert_to_grayscale**(*image*)
Convert RGB image to grayscale.

carpedm.data.preproc.**normalize**(*image*)
Rescale pixels values (to [-1, 1]).

carpedm.data.preproc.**pad_borders_or_shrink**(*image*, *char_bbox*, *line_bbox*, *shape*, *maintain_aspect=True*)
Pad or resize the image.

If the desired shape is larger than the original, then that axis is padded equally on both sides with the mean pixel value in the image. Otherwise, the image is resized with BILINEAR interpolation such that the aspect ratio is maintained.

Parameters

- **image** (tf.Tensor) – Image tensor [height, width, channels].
- **char_bbox** (tf.Tensor) – Character bounding box [4].
- **line_bbox** (tf.Tensor) – Line bounding box [4].
- **shape** (tuple of int) – Output shape.
- **maintain_aspect** (*bool*) – Maintain the aspect ratio.

Returns Resized image. `tf.Tensor`: Adjusted character bounding boxes. `tf.Tensor`: Adjusted line bounding boxes.

Return type `tf.Tensor`

carpedm.data.providers

Data providers for Task input function.

This module provides a generic interface for providing data useable by machine learning algorithms.

A provider may either (1) receive data from the method that initialized it, or (2) receive a directory path where the data to load is stored.

Todo:

- **Generator**
  - numpy
  - pandas DataFrame

---

class carpedm.data.providers.**DataProvider**(*target_id*)

Data provider abstract class.

make_batch(*batch_size*)

Generator method that returns a new batch with each call.

Parameters **batch_size** (*int*) – Number of examples per batch.

Returns Batch features. array_like: Batch targets.

Return type dict

class carpedm.data.providers.**TFDataSet**(*target_id*, *data_dir*, *subset*, *num_examples*, *pad_shape*, *sparse_labels*)

TensorFlow DataSet provider from TFRecords stored on disk.

make_batch(*batch_size*, *single_char=False*)

Generator method that returns a new batch with each call.

Parameters **batch_size** (*int*) – Number of examples per batch.

Returns Batch features. array_like: Batch targets.

Return type dict

carpedm.data.util

Data utilities.

This module provides utility methods/classes used by other data modules.

Todo:

- **Tests**
  - `generate_features`
- Refactor `generate_features`

- Fix `class_mask` for overlapping characters.

---

class carpedm.data.util.**BBox**(*xmin*, *xmax*, *ymin*, *ymax*)
Bounding box helper class.

class carpedm.data.util.**Character**(*label*, *image_id*, *x*, *y*, *block_id*, *char_id*, *w*, *h*)
Helper class for storing a single character.

class carpedm.data.util.**ImageMeta**(*filepath*, *full_image=False*, *first_char=None*)
Class for storing and manipulating image metadata.

add_char(*char*)
Add a character to the image.

Parameters **char** (Character) – The character to add.

char_bboxes
Bounding boxes for characters.

Returned bounding boxes are relative to (*xmin()*, *ymin()*).

Returns The return values.

Return type list of *carpedm.data.util.BBox*

char_labels
Character labels

Returns The return value.

Return type list of str

char_mask
Generate pseudo-pixel-level character mask.

Pixels within character bounding boxes are assigned to positive class (1), others assigned negative class (0).

Returns Character mask of shape (height, width, 1)

Return type numpy.ndarray

class_mask(*vocab*)
Generate a character class image mask.

Note: Where characters overlap, the last character added is arbitrarily the one that will be represented in the mask. This should be fixed in a future version.

Parameters **vocab** (Vocabulary) – The vocabulary for converting to ID.

Returns Class mask of shape (height, width, 1)

Return type numpy.ndarray

combine_with(*images*)

Parameters **images** (*list of ImageMeta*) –

full_h
Height (in pixels) of full raw parent image.

Returns The return value.

Return type int

full_w
Width (in pixels) of full raw parent image.

Returns The return value.

Return type int

generate_features(*image_shape*, *vocab*, *chunk*, *character*, *line*, *label*, *bbox*)

Parameters

- **image_shape** (*tuple or None*) – Shape (height, width) to which images are resized, or the size of each chunk if chunks == True.
- **vocab** (Vocabulary *or None*) – Vocabulary for converting characters to IDs. Required `if character and label`.
- **chunk** (*bool*) – Instead of using the original image, return a list of image chunks and corresponding features extracted from the original image on a regular grid. The original image is padded to divide evenly by chunk shape.
- **character** (*bool*) – Include character info (ID, bbox).
- **line** (*bool*) – Include line info (bbox) in features.
- **label** (*bool*) – Include label IDs in features.
- **bbox** (*str or None*) – If not None, include bbox in features as unit (e.g. 'pixel', 'ratio' [of image]))

Returns Feature dictionaries.

Return type `list` of `dict`

height
Height (in pixels) in full parent image original scale.

Returns The return value.

Return type int

line_bboxes
Bounding boxes for lines in the image,

Note: Currently only meaningful when using full page image.

Returns The return values.

Return type `list` of *BBox*

line_mask
Generate pseudo-pixel-level line mask.

Pixels within line bounding boxes are assigned to positive class (1), others assigned negative class (0).

Returns Line mask of shape (height, width, 1)

Return type `numpy.ndarray`

load_image(*shape*)
Load image and resize to shape.

If `shape` is None or (None, None), original size is maintained.

Parameters **shape** (*tuple or None*) – Output dimensions (height, width).

Returns Resized image.

Return type numpy.ndarray

new_shape(*shape*, *ratio=False*)
Resolves (and computes) input shape to a consistent type.

Parameters

- **shape** (*tuple or None*) – New shape of image (height, width), with potentially inconsistent types.
- **ratio** (*bool*) – Return new size as ratio of original size.

Returns Absolute or relative height int or float: Absolute or relative width

Return type int or float

num_chars
Number of characters in the image.

Returns The return value.

Return type int

valid_char(*char*, *same_line=False*)
Check if char is a valid character to include in image.

Parameters

- **char** (Character) – The character to validate.
- **same_line** (*bool*) – Consider whether char is in the same line as those already in the image example.

Returns True for valid, False otherwise.

Return type bool

width
Width (in pixels) in full parent image original scale.

Returns The return value.

Return type int

xmax
Image's maximum x-coordinate (column) in raw parent image.

Returns The return value.

Return type int

xmin
Image's minimum x-coordinate (column) in raw parent image.

Returns The return value.

Return type int

ymax
Image's maximum y-coordinate (row) in raw parent image.

Returns The return value.

Return type int

ymin
Image's minimum y-coordinate (row) in raw parent image.

Returns The return value.

Return type int

class carpedm.data.util.**ImageTFOps**
Helper class for decoding and resizing images.

carpedm.data.util.**image_path**(*data_dir*, *bib_id*, *image_id*)
Generate path to a specified image.

Parameters

- **data_dir** (*str*) – Path to top-level data directory.
- **bib_id** (*str*) – Bibliography ID.
- **image_id** (*str*) – Image ID.

Returns: String

2.4.2 Neural Networks

carpedm.nn.conv

Convolutional layers and components.

class carpedm.nn.conv.**CNN**(*kernel_size=((3, 3), (3, 3), (3, 3), (3, 3))*, *num_filters=(64, 96, 128, 160)*, *padding='same'*, *pool_size=((2, 2), (2, 2), (2, 2), (2, 2))*, *pool_stride=(2, 2, 2, 2)*, *pool_every_n=1*, *pooling_fn=<MagicMock name='mock.max_pooling2d' id='140624924645360'>*, *activation_fn=<MagicMock name='mock.relu' id='140624925111128'>*, **args*, ***kwargs*)
Modular convolutional neural network layer class.

name
Unique identifier for the model.

The model name will serve as directory name for model-specific results and as the top-level `tf.variable_scope`.

Returns The model name.

Return type str

carpedm.nn.op

Operations for transforming network layer or input.

carpedm.nn.rnn

Recurrent layers and components.

carpedm.nn.util

Utilities for managing and visualizing neural network layers.

carpedm.nn.util.**activation_summary**(*x*)
Helper to create summaries for activations. Creates a summary that provides a histogram of activations. Creates a summary that measures the sparsity of activations. :param x: Tensor

Returns nothing

carpedm.nn.util.**name_nice**(*raw*)
Convert tensor name to a nice format.

Remove 'tower_[0-9]/' from the name in case this is a multi-GPU training session. This helps the clarity of presentation on tensorboard.

2.4.3 Models

carpedm.models.generic

This module defines base model classes.

class carpedm.models.generic.**Model**
Abstract class for models.

forward_pass(*features*, *data_format*, *axes_order*, *is_training*)
Main model functionality.

Must be implemented by subclass.

Parameters

- **features** (*array_like or dict*) – Input features.
- **data_format** (*str*) – Image format expected for computation, 'channels_last' (NHWC) or 'channels_first' (NCHW).
- **axes_order** (*list or None*) – If not None, is a list defining the axes order to which image input should be transposed in order to match data_format.
- **is_training** (*bool*) – Training if true, else evaluating.

Returns The return value, e.g. class logits.

Return type array_like or dict

initialize_pretrained(*pretrained_dir*)
Initialize a pre-trained model or sub-model.

Parameters **pretrained_dir** (*str*) – Path to directory where pretrained model is stored. May be used to extract model/sub-model name. For example:

```
name = pretrained_dir.split('/')[-1].split('_')[0]
```

Returns Map from pre-trained variable to model variable.

Return type dict

name
Unique identifier for the model.

Used to identify results generated with the model.

Must be implemented by subclass.

Returns The model name.

Return type str

class carpedm.models.generic.**TFModel**
Abstract class for TensorFlow models.

_forward_pass(*features*, *data_format*, *axes_order*, *is_training*, *reuse*)
Main model functionality.

Must be implemented by subclass.

forward_pass(*features*, *data_format*, *axes_order*, *is_training*, *new_var_scope=False*, *reuse=False*)
Wrapper for making nested variable scopes.

Extends Model.

Parameters

- **new_var_scope** (*bool*) – Use a new variable scope.
- **reuse** (*bool*) – Reuse variables with same scope.

name
Unique identifier for the model.

The model name will serve as directory name for model-specific results and as the top-level `tf.variable_scope`.

Returns The model name.

Return type str

2.4.4 Tasks

carpedm.tasks.generic

Base task class.

Todo:

- Get rid of `model_fn` dependency on `input_fn`.
- LONG TERM: Training methods other than TensorFlow Estimator.

---

class carpedm.tasks.generic.**Task**(*data_dir*, *task_dir*, *test_split='hnsd00000'*, *dev_split=0.1*, *dev_factor=1*, *dataset_format='tfrecords'*, *num_shards=8*, *num_threads=8*, *shape_store=None*, *shape_in=None*, *vocab_size=None*, *min_frequency=0*, *seed=None*, ***kwargs*)
Abstract class for Tasks.

__init__(*data_dir*, *task_dir*, *test_split='hnsd00000'*, *dev_split=0.1*, *dev_factor=1*, *dataset_format='tfrecords'*, *num_shards=8*, *num_threads=8*, *shape_store=None*, *shape_in=None*, *vocab_size=None*, *min_frequency=0*, *seed=None*, ***kwargs*)
Initializer.

Parameters

- **data_dir** (*str*) – Directory where raw data is stored.

- **task_dir** (*str*) – Top-level directory for storing tasks data and results.
- **test_split** (*float or str*) – Either the ratio of all data to use for testing or specific bibliography ID(s). Use comma-separated IDs for multiple books.
- **dev_split** (*float or str*) – Either the ratio of training data to use for dev/val or specific bibliography ID(s). Use comma-separated IDs for multiple books.
- **dev_factor** – (int): Size of development set should be divisible by this value. Useful for training on multiple GPUs.
- **dataset_format** (*str*) – Base storage unit for the dataset.
- **vocab_size** (*int*) – Maximum vocab size.
- **min_frequency** (*int*) – Minimum frequency of type to be included in vocab.
- **shape_store** (*tuple or None*) – Size to which images are resized for storage, if needed, e.g. for TFRecords. The default is to not perform any resize. Please see this *note on image shape* for more information.
- **shape_in** (*tuple or None*) – Size to which images are resized by interpolation or padding before being input to a model. Please see this *note on image shape* for more information.
- **num_shards** (*int*) – Number of sharded output files.
- **num_threads** (*int*) – Number of threads to run in parallel.
- **seed** (*int or None*) – Number for seeding rng.
- ****kwargs** – Unused arguments.

__metaclass__
alias of abc.ABCMeta

__weakref__
list of weak references to the object (if defined)

bbox
When creating a dataset, generate appropriate bounding boxes for the tasks (determined by e.g. self.character, self.line).

Returns Use bounding boxes.

Return type bool

character
When creating a dataset, tell the meta_loader to generate character features, e.g. label, bbox.

Returns Use character features.

Return type bool

character_set
The Japanese characters (e.g. kana, kanji) of interest.

Preset character sets may include the following component sets:

- hiragana
- katakana
- kana
- kanji

- punct (punctuation)
- misc

Returns The character set.

Return type *CharacterSet*

chunk
When creating a dataset, instead of using the original image, extract non-overlapping chunks of size *image_shape* and the corresponding features from the original image on a regular grid. The original image is padded to divide evenly by *image_shape*.

Note: currently only objects that are entirely contained in the block will have its features propagated.

Returns

Return type bool

image_scope
Portion of original image for each example.

Available scopes are 'char', 'seq', 'line', 'page'.

Returns Task image scope

Return type str

input_fn (*batch_size*, *subset*, *num_shards*, *overwrite=False*)
Returns (sharded) batches of data.

Parameters

- **batch_size** (*int*) – The batch_size
- **subset** (*str*) – The subset to use. One of {train, dev, test}.
- **num_shards** (*int*) – Number of data_shards to produce.
- **overwrite** (*bool*) – Overwrite existing data.

Returns Features of length num_shards. (list): Labels of length num_shards.

Return type (list)

label
When creating a dataset, generate character labels.

Returns Use character labels

Return type bool

line
When creating a dataset, tell the meta_loader to generate line features, e.g. bbox.

Returns Use line features.

Return type bool

loss_fn (*features*, *model_output*, *targets*, *is_training*)
Computes an appropriate loss for the tasks.

Must be implemented in subclass.

Parameters

- **features** (*dict*) – Additional features for computing loss.

- **model_output** (*tf.Tensor or dict of tf.Tensor*) – Model output used for computing the batch loss, e.g. class logits.
- **targets** (*tf.Tensor*) – Ground truth targets.
- **is_training** (*bool*) – The model is training.

Returns Losses of type 'int32' and shape [batch_size, 1]

Return type tf.Tensor

max_sequence_length
Maximum sequence length.

Only used if `image_scope == 'seq'`.

Returns

Return type int or None

model_fn (*model*, *variable_strategy*, *num_gpus*, *num_workers*, *devices=None*)
Model function used by TensorFlow Estimator class.

Parameters

- **model** (*pmjtc.models.generic.Model*) – The models to run.
- **variable_strategy** (*str*) – Where to locate variable operations, either 'CPU' or 'GPU'.
- **num_gpus** (*int*) – Number of GPUs to use, if available.
- **devices** (*tuple*) – Specific devices to use. If provided, overrides num_gpus.
- **num_workers** (*int*) – Parameter for distributed training.

Returns:

num_classes
Total number of output nodes, includes reserved tokens.

regularization (*hparams*)

Parameters **hparams** – Hyperparameters, e.g. weight_decay

Returns:

reserved
Reserved tokens for the tasks.

The index of each token in the returned tuple will be used as its integer ID.

Returns The reserved characters

Return type tuple

results (*loss*, *tower_features*, *tower_preds*, *tower_targets*, *is_training*)
Accumulates predictions, computes metrics, and determines the tensors to log and/or visualize.

Parameters

- **loss** (*tf.float*) – Global loss.
- **tower_features** (*list of dict*) – Tower feature dicts.
- **tower_preds** (*list*) – Tower predictions.
- **tower_targets** (*list of tf.Tensor*) – Tower targets.

- **is_training** (*bool*) – The model is training.

Returns The tensors to log dict: All predictions dict: Evaluation metrics

Return type dict

sequence_length
If max_sequence_length is None, this gives the deterministic length of a sequence, else the minimum sequence length.

Only used if `image_scope == 'seq'`.

Returns

Return type int or None

sparse_labels
Generate labels as a SparseTensor, e.g. for CTC loss.

Returns Use sparse labels.

Return type (bool)

target
Determines the value against which predictions are compared.

For a list of possible targets, refer to carpedm.data.util.ImageMeta.generate_features()

Returns feature key for the target

Return type str

task_data_dir
Directory where tasks data is stored.

Returns str

2.4.5 Utilities

carpedm.util.eval

Evaluation helpers.

carpedm.util.eval.**confusion_matrix_metric**(*labels*, *predictions*, *num_classes*)
A confusion matrix metric.

Parameters

- **labels** (*tf.Tensor*) – Ground truth labels.
- **predictions** (*tf.Tensor*) – Predictions.
- **num_classes** (*int*) – Number of classs.

Returns tf.update_op:

Return type tf.Tensor

carpedm.util.eval.**plot_confusion_matrix**(*cm*, *classes*, *normalize=False*, *save_as=None*, *title='Confusion matrix'*)
This function prints and plots the confusion matrix. Normalization can be applied by setting *normalize=True*.

Slight modification of methods here

carpedm.util.registry

Registry for models and tasks.

Define a new models by subclassing models.Model and register it:

```
@registry.register_model
class MyModel(models.Model):
    ...
```

Access by snake-cased name: `registry.model("my_model")`.

See all the models registered: `registry.list_models()`.

References

1. Lightly modified Tensor2Tensor registry.

carpedm.util.registry.**default_name**(*obj_class*)
Convert class name to the registry's default name for the class.

Parameters **obj_class** – the name of a class

Returns The registry's default name for the class.

carpedm.util.registry.**default_object_name**(*obj*)
Convert object to the registry's default name for the object class.

Parameters **obj** – an object instance

Returns The registry's default name for the class of the object.

carpedm.util.registry.**display_list_by_prefix**(*names_list*, *starting_spaces=0*)
Creates a help string for `names_list` grouped by prefix.

carpedm.util.registry.**help_string**()
Generate help string with contents of registry.

carpedm.util.registry.**model**(*name*)
Retrieve a model by name.

carpedm.util.registry.**register_model**(*name=None*)
Register a models. `name` defaults to class name snake-cased.

carpedm.util.registry.**register_task**(*name=None*)
Register a Task. `name` defaults to cls name snake-cased.

carpedm.util.registry.**task**(*name*)
Retrieve a task by name.

carpedm.util.train

Training utilities.

This modules provides utilities for training machine learning models. It uses or makes slight modifications to code from the TensorFlow CIFAR-10 estimator tutorial.

carpedm.util.train.**config_optimizer**(*params*)
Configure the optimizer used for training.

Sets the learning rate schedule and optimization algorithm.

Parameters **params** (*tf.contrib.training.HParams*) – Hyperparameters.

Returns tf.train.Optimizer

2.5 Benchmarks

2.5.1 Single Kana OCR

Running the example *main.py* for the full PMJTC dataset (171,944 training examples, 131 character classes, as of 2 May 2018)

- **On a 2017 MacBook Pro:**
  - Generating the (train & dev) data: 1 hour, 20 minutes
  - Training the model for 5 epochs: 2 hours, 27 minutes
  - Dev Accuracy: 94.67%
- **On a Linux Machine using 1 Titan X (Pascal) GPU:**
  - Generating the (train & dev) data: 31 minutes
  - Training the model for 5 epochs: 21 minutes
  - Dev Accuracy: 95.23%

2.6 Contributing

When contributing to CarpeDM, please first discuss the change you wish to make via github issue, email, or any other method with the owner of this repository before making a change.

Please note we have a *Code of Conduct*, please follow it in all your interactions with the project.

2.6.1 Making Changes

1. Fork the repository.
2. Clone the fork to your local machine:

```
$ git clone https://github.com/YOUR-USERNAME/carpedm
```

3. Add an upstream remote for syncing with the master repo:

```
$ cd carpedm
$ git remote add upstream https://github.com/SimulatedANeal/carpedm
```

4. Make sure your repository is up to date with `master`:

```
$ git pull upstream master
```

5. (Create a topical branch):

```
$ git checkout -b branch-name
```

6. Make your changes.

7. Again, make sure your repo is up to date.
8. Push to your forked repo:

```
$ git push origin branch-name
```

9. Make Pull Request.

2.6.2 Pull Requests

1. Make changes as directed above.
2. Update `CHANGES.md` with details of changes to the interface.
3. Increase the `__version__` in `carpedm.__init__.py` to the new version number that this Pull Request would represent. The versioning scheme we use is SemVer.
4. You may merge the Pull Request in once you have the sign-off of the lead developer, or if you do not have permission to do that, you may request the reviewer to merge it for you.

2.7 Code of Conduct

2.7.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

2.7.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

2.7.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

2.7.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

2.7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at . All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

2.7.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at http://contributor-covenant.org/version/1/4

2.8 License

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

A

B

C

D

F

V

W

X

Y