junn

Release 1.0.0

Christian C. Sachs

Jun 30, 2021

CONTENTS

See *JUNN Readme* for information.

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

TWO

CONTENTS:

2.1 JUNN Readme

Important: The repository and accompanying manuscript are currently in the process of being finalized. During this period, not everything in the repository might be completely finished or in complete working order yet.

The Jülich U-Net Neural Network Toolkit. JUNN is a neural network training tool, aimed at allowing the easy training of configurable U-Nets or other pixel to pixel segmentation networks, with the resulting networks usable in a standalone manner.

2.2 Citation

The manuscript for JUNN is currently in preparation, if you use JUNN, please cite the upcoming publication.

2.3 Prerequisites

JUNN is split into two components: a training and a prediction component. The training component runs the computations locally, the usage of a CUDA-compatible GPU is highly advisable. The prediction component can either run the predictions locally, or offload the computations to remote machines, and in general prediction is, albeit slower, still somewhat feasible with CPU-only processing speed. Via the remote prediction functionality, a server with GPUs may be shared for multiple non-GPU prediction clients, see the documentation how such a scenario may be set up.

2.4 Installation/Usage

2.4.1 Docker

The simplest way to run JUNN is to use the Docker images provided (note that to use a GPU inside a container, the necessary runtime needs to be installed) :

```
# training
> docker run --gpus all --ipc=host --tty --interactive --rm -v `pwd`:/data modsim/junn␣
↪train --NeuralNetwork Unet --model model_name training_input.tif
# prediction
> docker run --gpus all --ipc=host --tty --interactive --rm -v `pwd`:/data modsim/junn␣
↪predict --model model_name data.tif
```

2.4.2 Anaconda

To install JUNN locally, we recommend using the Anaconda package manager.

Since Anaconda TensorFlow packages tend to lag a bit behind, it is necessary to install the official TensorFlow binaries from PyPI.

```
> conda create -n junn
> conda activate junn
> conda install -c modsim junn
> pip install tensorflow tensorflow-addons tensorflow-serving-api
# usage
> python -m junn train --NeuralNetwork Unet --model model_name training_input.tif
> python -m junn predict --model model_name data.tif
```

2.4.3 Github

Installing JUNN from Github is only recommended for users familiar with Python development. Either conda packages can be built locally and installed, or the pip packages can be built and installed, or the necessary dependencies can be installed and the source run in-place.

```
# let conda build and install the packages
> conda create -n junn
> conda activate junn
> conda build junn-predict/recipe recipe
> conda install -c local junn-predict junn

# ... or let pip build and install the packages
> pip install junn-predict
> pip install .

# always install TensorFlow via pip
> pip install tensorflow tensorflow-addons tensorflow-serving-api
```

2.5 Documentation

The documentation can be built using sphinx, or will be available at readthedocs.

2.6 Quick Start

Note: In the next two sections, junn's call will just be abbreviated junn. Depending on whether Docker or a local Python installation should be used, this would mean either `docker run --gpus all --ipc=host --tty --interactive --rm -v `pwd`:/data modsim/junn` or `python -m junn`.

2.6.1 Training

For training a new neural network, first the network structure needs to be chosen, therefore the `--NeuralNetwork` parameter is used. Most commonly the `Unet` neural network will be chosen here, with various parameters being available defining the network structure (e.g. `levels=4`, `filters=64`, `activation=relu` to name a few). The model save path is passed via the `--model` parameter, which will contain a TensorFlow formatted model after training (which is itself a directory structure). Finally, an input of ground truth data is needed, for example an ImageJ TIFF file with ROIs denoting the desired structures, such as cells. Various tunable parameters can be set via the `-t` switch; a list of available tunables can be output by using the `--tunables-show` argument.

```
> junn train --NeuralNetwork "Unet(optionA=1,optionB=2)" --model model_name -t␣
↪SomeTunable=value training_data.tif
```

Upon call, the training will start, outputting the configured metrics at each epoch. If configured, outputs for TensorBoard will be written. Once the training is finished, or was interrupted by the user, e.g. because the achieved quality is good enough, the model is ready for prediction:

2.6.2 Prediction

JUNN can take various input formats, such as Nikon ND2, Zeiss CZI, or OME-TIFF, and predict the image data, outputting either the raw probability masks from the neural networks, or detected objects as ImageJ ROIs.

```
> junn predict --model model_name file_to_predict.tif --output result.tif --output-type␣
↪roi
```

2.7 License

JUNN is licensed under the 2-clause BSD License, see *License*.

2.8 Concepts

This page's aim is to convey some of JUNN's concepts, which might go amiss if the merely the API documentation is observed.

2.8.1 Network and training pipeline assembly by OOP & Mix-ins

The core structure defining a network and training pipeline within JUNN is the NeuralNetwork class, it has various member methods which can be overridden in order to quickly create a new type of network or altered training methodology.

To this extend, various steps, such as defining the network, defining the input augmentations, etc. are done by individual methods.

Furthermore, concepts, such as 'tile-based training' or 'augmentation' are done in individual mix-ins, which can just be inherited into a desired training class.

E.g. to implement a novel network following a tile-based prediction approach with the standard JUNN augmentation functionality, one can just subclass the NeuralNetwork class as follows:

TODO: Example

2.8.2 Strictly building a compute graph using TensorFlow primitives

Other tools often follow a mixed approach, where data is pre-processed within Python to specifically match a TensorFlow compute graph generated, which has the downside that (possibly slower) Python computation is used, and the resulting model remains highly dependent on the Python-based pre/postprocessing.

To avoid those potential downsides, JUNN tries to model all steps necessary for processing the data in pure TensorFlow: Once data is transferred into a .tfrecord file, processing is done solely by (performant C++/GPU-based) TensorFlow primitives: The data augmentation pipeline, as well as the training. As an additional gimmick, the prediction routines (such as tiling the image, performing normalization, etc.) are encoded via TensorFlow operations as well, yielding to a completely standalone usable model, which can readily be used in completely distinct tools, such as TensorFlow Serving or DeepImageJ.

To this extend, JUNN expects the programmer to define @tf.function decorated functions for various purposes, which can be completely transformed into a TensorFlow compute graph.

TODO: Example

Furthermore, the prediction part of JUNN makes use of these properties: Instead of completely reinstantiating the model, only the graph is loaded in lightweight manner and execution is left to TensorFlow. By this unified approach, the various prediction backends (direct, via GRPC/HTTP and TensorFlow Serving) were easily implementable.

2.9 junn package

2.9.1 Subpackages

2.9.1.1 junn.common package

Subpackages

junn.common.functions package

Submodules

junn.common.functions.affine module

Additional affine transformation functions written as TensorFlow-tf.functions.

junn.common.functions.affine.**radians**(*degrees*)
Convert degrees to radians.

Parameters **degrees** – Degrees to convert

Returns Resulting radians

junn.common.functions.affine.**shape_to_h_w**(*shape*)
Extract height and width from a shape tuple with between 1 and 4 dimensions.

Parameters **shape** – The input shape

Returns A tuple of height and width.

junn.common.functions.affine.**tfm_identity**(*shape=(0, 0)*)
Generate an identity matrix for use with the affine transformation matrices.

Parameters **shape** – The parameter is ignored and only present for call-compatibility with the other functions.

Returns The affine transformation matrix

junn.common.functions.affine.**tfm_reflect**(*x=0.0*, *y=0.0*, *shape=(0, 0)*)
Generate a reflection affine transformation matrix.

Parameters

- **x** – Whether to reflect in horizontal direction, a value of 1 leads to a reflection, any other value not.
- **y** – Whether to reflect in vertical direction, a value of 1 leads to a reflection, any other value not.
- **shape** – The shape of the image to be transformed

Returns The affine transformation matrix

junn.common.functions.affine.**tfm_rotate**(*angle=0.0*, *shape=(0, 0)*)
Generate a rotation affine transformation matrix.

Parameters

- **angle** – The angle to r
- **shape** – The shape of the image to be transformed

Returns The affine transformation matrix

junn.common.functions.affine.**tfm_scale**(*xs=1.0*, *ys=None*, *shape=(0, 0)*)
Generate a scaling affine transformation matrix.

Parameters

- **xs** – The scale factor for the horizontal direction
- **ys** – The scale factor for the vertical direction, if None, then the image will be uniformly scaled
- **shape** – The shape of the image to be transformed

Returns The affine transformation matrix

junn.common.functions.affine.**tfm_shear**(*x_angle=0.0*, *y_angle=0.0*, *shape=(0, 0)*)
Generate a shear transformation matrix.

Parameters

- **x_angle** – The angle to shear the image in horizontal direction
- **y_angle** – The angle to shear the image in vertical direction
- **shape** – The shape of the image to be transformed

Returns The affine transformation matrix

junn.common.functions.affine.**tfm_shift**(*x=0.0*, *y=0.0*, *shape=None*)
Generate a shift affine transformation matrix.

Parameters

- **x** – Shift value in horizontal direction
- **y** – Shift value in vertical direction

- **shape** – The parameter is ignored and only present for call-compatibility with the other functions.

Returns The affine transformation matrix

junn.common.functions.affine.**tfm_to_tf_transform**(*mat*)
Truncate an affine transformation matrix to the parameters used by tf transform.

Parameters **mat** – Input matrix

Returns Truncated output matrix

junn.common.launcher package

junn.common.layers package

Submodules

junn.common.layers.run_model_layer module

Layer to run a model in a tiled manner.

class junn.common.layers.run_model_layer.**RunModelTiled**(**args: Any*, ***kwargs: Any*)
Bases: tensorflow.python.keras.layers.

Run a model in a tiled manner.

With a fixed input size in a tiled manner over the (larger) input tensor (image).

call(*raw_input_tensor*, ***kwargs*)
Inner function called by Keras.

Parameters

- **raw_input_tensor** –
- **kwargs** –

Returns

Submodules

junn.common.callbacks module

junn.common.distributed module

Helper functions for distributed training.

junn.common.distributed.**barrier**(*name*)
Create a barrier which will synchronize all processes.

Can be used if the worker processes need to wait for the main process.

Parameters **name** –

Returns

junn.common.distributed.**get_callbacks**()
Get Keras callbacks.

If Horovod is present, this will be the BroadCastGlobalVariables callback.

Returns

junn.common.distributed.**init**(*device_pinning=True*)
Initialize Horovod if present.

Parameters **device_pinning** – Whether GPUs should be pinned.

Returns

junn.common.distributed.**is_rank_zero**()
Return whether the current process is rank zero, i.e. the main process.

Returns

junn.common.distributed.**is_running_distributed**()
Return True if JUNN is running distributed and Horovod is initialized.

Returns

junn.common.distributed.**local_rank**()
Return the local rank (on the physical machine).

Returns

junn.common.distributed.**pin_devices**()
Pin each GPU to a worker.

Returns

junn.common.distributed.**rank**()
Return the global rank.

This is the rank within all running instances on possibly multiple machines.

Returns

junn.common.distributed.**size**()
Return the count of running instances.

Returns

junn.common.distributed.**wrap_optimizer**(*optimizer*)
Wrap the Keras optimizer.

If Horovod is present, with DistributedOptimizer.

Parameters **optimizer** –

Returns

junn.common.losses module

Additional losses suitable for segmentation tasks.

junn.common.losses.**accuracy**(*y_true*, *y_pred*)
Calculate the accuracy.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction

Returns accuracy

junn.common.losses.**dice_index**(*y_true*, *y_pred*)
Calculate the Dice index.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction

Returns the Dice index

junn.common.losses.**dice_index_direct**(*y_true*, *y_pred*)
Directly calculate the Dice index.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction

Returns Dice index

junn.common.losses.**dice_index_weighted**(*y_true*, *y_pred*, *y_weight*)
Calculate the Dice index with weighting applied.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction
- **y_weight** – weights

Returns weighted Dice index

junn.common.losses.**dice_loss**(*y_true*, *y_pred*)
Calculate the Dice loss, i.e. the negative Dice index.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction

Returns Dice loss

junn.common.losses.**dice_loss_unclipped**(*y_true*, *y_pred*)
Calculate the Dice loss, i.e. the negative Dice index without clipping.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction

Returns Dice loss

junn.common.losses.**dice_loss_weighted**(*y_true*, *y_pred*, *y_weight*)
Calculate the weighted Dice loss, i.e. negative Dice index.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction
- **y_weight** – weights

Returns weighted Dice loss

junn.common.losses.**epsilon**()
Epsilon of the current compute hardware.

Returns epsilon as a floating point type

junn.common.losses.**f_score**(*y_true*, *y_pred*)
Calculate the F score.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction

Returns F-score

junn.common.losses.**false_negative**(*y_true*, *y_pred*)

junn.common.losses.**false_positive**(*y_true*, *y_pred*)

junn.common.losses.**flatten_and_clip**(*values*)
Flatten and clip a tensor.

Parameters **values** – values

Returns flattened and clipped to 0-1 values

junn.common.losses.**generate_tversky_loss**(*alpha=0.5*, *beta=0.5*)
Generate a Tversky loss function.

Parameters

- **alpha** – alpha
- **beta** – beta

Returns a Tversky loss function with fixed parameters

junn.common.losses.**jaccard_index**(*y_true*, *y_pred*)
Calculate the Jaccard index.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction

Returns the Jaccard index

junn.common.losses.**mixin_flatten_and_clip**(*what*)
Mixin to assure a function gets a flattened and clipped tensor.

Parameters **what** – function to be modified

Returns modified function

junn.common.losses.**nan_loss**(*y_true*, *y_pred*)
Return always NaN for debugging and testing purposes.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction

Returns all NaN

junn.common.losses.**precision**(*y_true*, *y_pred*)
Calculate the precision.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction

Returns precision

junn.common.losses.**recall**(*y_true*, *y_pred*)
Calculate the recall.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction

Returns recall

junn.common.losses.**tp_tn_fp_fn_precision_recall**(*y_true*, *y_pred*)
Calculate a set of simple metrics in one go.

True positives, true negatives, false positives, false negatives, precision as well as recall.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction

Returns tuple of metrics (tp, tn, fp, fn, precision, recall)

junn.common.losses.**true_negative**(*y_true*, *y_pred*)

junn.common.losses.**true_positive**(*y_true*, *y_pred*)

junn.common.losses.**tversky_index**(*y_true*, *y_pred*, *alpha=0.5*, *beta=0.5*)
Calculate the Tversky index.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction
- **alpha** – alpha
- **beta** – beta

Returns the Tversky index

junn.common.losses.**weighted_loss**(*y_true*, *y_pred*, *y_weight*)
Weighted loss function.

Parameters

- **y_true** – ground truth
- **y_pred** – prediction
- **y_weight** – weights

Returns weighted loss

2.9.1.2 junn.datasets package

Submodules

junn.datasets.tfrecord module

2.9.1.3 junn.io package

Submodules

junn.io.file_lists module

File list helper functions.

junn.io.file_lists.**generate_glob_and_replacer**(*search*, *replace*)
Prepare a wildcard pattern for globbing and replacing.

Parameters

- **search** –
- **replace** –

Returns

junn.io.file_lists.**generate_replacer**(*search*, *replace*)
Prepare a wildcard pattern to be used a replacement regex.

Parameters

- **search** –
- **replace** –

Returns

junn.io.file_lists.**prepare_for_regex**(*input_*, *task='search'*)
Prepare a wildcard pattern for specific use.

Parameters

- **input** –
- **task** –

Returns

junn.io.tiffmasks module

junn.io.training module

2.9.1.4 junn.networks package

Subpackages

junn.networks.functional package

Submodules

junn.networks.functional.link_net module

junn.networks.functional.unet_layer module

junn.networks.functional.msd module

junn.networks.mixins package

Submodules

junn.networks.mixins.tile_based_network module

junn.networks.mixins.weighted_loss module

junn.networks.mixins.preprocessing module

junn.networks.mixins.tile_based_prediction module

junn.networks.mixins.tile_based_training module

junn.networks.mixins.augmentation module

junn.networks.mixins.multichannel module

junn.networks.mixins.prediction module

junn.networks.mixins.losses module

junn.networks.mixins.deepimagej_helper module

Submodules

junn.networks.LinkNet module

junn.networks.Unet module

junn.networks.util module

junn.networks.MSD module

junn.networks.all module

2.9.1.5 junn.predict package

2.9.1.6 junn.train package

Submodules

junn.train.cli module

2.10 junn_predict package

2.10.1 Subpackages

2.10.1.1 junn_predict.common package

Submodules

junn_predict.common.cli module

junn_predict.common.logging module

junn_predict.common.configure_tensorflow module

Helper functionality to configure TensorFlow.

junn_predict.common.configure_tensorflow.**configure_tensorflow**(*seed=None*, *windows_maximum_gpu_memory=0.75*)

Configure TensorFlow.

It is important that this function is called BEFORE first TF import.

- Reduces TF's log-level (before) loading.
- Sets all devices to dynamic memory allocation (growing instead of complete)
- (On Windows) Sets overall maximum TensorFlow memory to windows_maximum_gpu_memory
- Removes the tensorflow log adapter from the global logger
- Sets Keras to use TensorFlow (and sets USE_TENSORFLOW_KERAS to 1, custom environment variable)

Parameters

- **seed** – Optional. If set, will be passed to set_seed()
- **windows_maximum_gpu_memory** – Set the maximum GPU memory fraction to use (Windows only).

Returns None

junn_predict.common.configure_tensorflow.**get_gpu_memory_usages_megabytes**()
Get the current GPU memory usages.

Returns List of memory usages.

junn_predict.common.configure_tensorflow.**set_seed**(*seed*)
Set various RNG seeds, so their behavior becomes reproducible.

Seeds NumPy, Python random, and TensorFlow.

Parameters **seed** – Seed value to use.

Returns

junn_predict.common.tensorflow_addons module

Helper functions to dynamically load TensorFlow Addons.

junn_predict.common.tensorflow_addons.**try_load_tensorflow_addons**() → None
Load the TensorFlow Addons module, if present.

Will ignore any compatibility warnings, and loads all tfa activations into the normal namespace.

junn_predict.common.autoconfigure_tensorflow module

junn_predict.common.timed module

Helper module to perform on the fly benchmarking/time keeping.

class junn_predict.common.timed.**Timed**(*name=None*)
Bases: object

Context manager to keep track of elapsed time.

precision = 3

2.10.1.2 junn_predict.predict package

Subpackages

junn_predict.predict.connectors package

Submodules

junn_predict.predict.connectors.http_connector module

junn_predict.predict.connectors.grpc_connector module

junn_predict.predict.connectors.model_connector module

junn_predict.predict.connectors.local_model module

Submodules

junn_predict.predict.cli module

junn_predict.predict.detectors module

2.11 License

PYTHON MODULE INDEX

j

INDEX

N

P

R

S

T

W