GaMPEN

Release 1.0.0

Aritra Ghosh, Amrit Rau, Aayush Mishra

Jan 25, 2023

CONTENTS

GaMPEN is written in Python and relies on the PyTorch deep learning library to perform all of its tensor operations.

CHAPTER

ONE

INSTALLATION

Training and inference for GaMPEN requires Python 3.7+. Trained GaMPEN models can be run on a CPU to perform inference, but training a model requires access to a CUDA-enabled GPU for reasonable training times.

1. Create a new conda environment with Python 3.7+. Extensive instructions on how to create a conda enviornment can be found here. Of course, you could use any other method of creating a virtual environment, but we will assume you are using conda for the rest of this guide.

```
conda create -n gampen python=3.7
```

2. Activate the new environment

```
conda activate gampen
```

3. Navigate to the directory where you want to install GaMPEN and then clone this repository with

```
git clone https://github.com/aritraghsh09/GaMPEN.git
```

4. Navigate into the root directory of GaMPEN

```
cd GaMPEN
```

4. Install all the required dependencies with

```
make requirements
```

5. To confirm that the installation was successful, run

```
make check
```

It is okay if there are some warnings or some tests are skipped. The only thing you should look out for is errors produced by the `make check` command.

If you get an error about specific `libcudas` libraries being absent while running `make check`, this has probably to do with the fact that you don't have the appropriate CUDA and cuDNN versions installed for the PyTorch version being used by GaMPEN. See below for more details about GPU support.

CHAPTER

TWO

GPU SUPPORT

GaMPEN can make use of multiple GPUs while training if you pass in the appropriate arguments to the `train.py` script.

To check whether the GaMPEN is able to detect GPUs, type `python` into the command line from the root directory and run the following command:

```
from ggt.utils.device_utils import discover_devices
discover_devices()
```

The output should be `cuda`if GaMPEN can detect a GPU.

If the output is `cpu` then GaMPEN couldn't find a GPU. This could be because you don't have a GPU, or because you haven't installed the appropriate CUDA and cuDNN versions.

If you are using an NVIDIA GPU, then you can use this link for more details about specific CUDA and cuDNN versions that are compatible with different PyTorch versions. To check the version of PyTorch you are using, type `python` into the command line and then run the following code-block:

```
import torch
print(torch.__version__)
```

CHAPTER

THREE

QUICKSTART

The core parts of the GaMPEN ecosystem are :-

- Placing your data in a specific directory structure
- Using the `GaMPEN/ggt/data/make_splits.py` script to generate train/devel/test splits
- Using the `GaMPEN/ggt/train/train.py` script to train a GaMPEN model
- Using the MLFlow UI to monitor your model during and after training
- Using the `GaMPEN/ggt/modules/inference.py` script to perform predictions using the trained model.
- Using the `GaMPEN/ggt/modules/result_aggregator.py` script to aggregate the predictions into an easy-to-read pandas data-frame.

Attention: We highly recommend going through all our *Tutorials* to get an in-depth understanding of how to use GaMPEN, and an overview of all the steps above.

Here, we provide a quick-and-dirty demo to just get you started training your 1st GaMPEN model! This section is intentionally short without much explanation.

3.1 Data preparation

Let's download some simulated Hyper Suprime-Cam (HSC) images from the Yale servers. To do this run from the root directory of this repository:

```
make demodir=./../hsc hsc_demo
```

This should create a directory called `hsc` at the specified `demodir` path with the following components

```
- hsc
  - info.csv -- file names of the trianing images with labels
  - cutouts/ -- 67 images to be used for this demo
```

Now, let's split the data into train, devel, and test sets. To do this, run

```
python ./ggt/data/make_splits.py --data_dir=./../hsc/ --target_metric='bt'
```

This will create another folder called `splits` within `hsc` with the different data-splits for training, devel (validation), and testing.

3.2 Running the trainer

Run the trainer with

```
python ggt/train/train.py \
  --experiment_name='demo' \
  --data_dir='./../hsc/' \
  --split_slug='balanced-dev2' \
  --batch_size=16 \
  --epochs=2 \
  --lr=5e-7 \
  --momentum=0.99 \
  --crop \
  --cutout_size=239 \
  --target_metrics='custom_logit_bt,log_R_e,log_total_flux' \
  --repeat_dims \
  --no-nesterov \
  --label_scaling='std' \
  --dropout_rate=0.0004 \
  --loss='aleatoric_cov' \
  --weight_decay=0.0001 \
  --parallel
```

To list the all possible options along with explanations, head to the *Using GaMPEN* page or run

```
python ggt/train/train.py --help
```

3.3 Launching the MLFlow UI

Open a separate shell and activate the virtual environment that the model is training in. Then, run

```
mlflow ui
```

Now navigate to `http://localhost:5000/` to access the MLFlow UI which will show you the status of your model training.

3.3.1 MLFLow on a remote machine

First, on the server/HPC, navigate to the directory from where you initiated your GaMPEN run (you can do this on separate machine as well – only the filesystem needs to tbe same). Then execute the following command

```
mlflow ui --host 0.0.0.0
```

The `--host` option is important to make the MLFlow server accept connections from other machines.

Now from your local machine tunnel into the `5000` port of the server where you ran the above command. F or example, let's say you are in an HPC environment, where the machine where you ran the above command is named `server1` and the login node to your HPC is named `hpc.university.edu` and you have the username `astronomer`. Then to forward the port you should type the following command in your local machine

```
ssh -N -L 5000:server1:5000 astronomer@hpc.university.edu
```

If performing the above step without a login node (e.g., a server whhich has the IP `server1.university.edu`), you should be able to do

```
ssh -N -L 5000:localhost:5000 astronomer@server1.university.edu
```

After forwarding, if you navigate to `http://localhost:5000/` you should be able to access the MLFlow UI

3.4 Training on other datasets

1. First create the necessary directories with

```
mkdir -p (dataset-name)/cutouts
```

2. Place FITS files in `(dataset-name)/cutouts`.
3. Provide a file titled `info.csv` at `(dataset-name)`. This file should have (at least) a column titled `file_name` (corresponding to the names of the files in `(dataset-name)/cutouts`), a column titled `object_id` (with a unique ID of each file in `(dataset-name)/cutouts`) and one column for each of the variables that you are trying to predict. For example, if you are trying to predict the radius of a galaxy, you would have a column titled `radius`.
4. Generate train/devel/test splits with

```
python ggt/data/make_splits.py --data_dir=data/(dataset-name)/
```

The `make_splits.py` file splits the dataset into a variety of splits and you can choose to use any of these for your analysis. Details of the various splits are mentioned on the Using GaMPEN page.

After generating the splits, the `dataset-name` directory should look like this:

```
- dataset_name
    - info.csv
    - cutouts/
    - splits/
```

5. Follow the instructions under Running the trainer.

CHAPTER

FOUR

TUTORIALS

We have created the following tutorials to get you quickly started with using GaMPEN. To look into the details of each GaMPEN function used in these tutorials, please look at the API reference.

You can either download these notebooks from GitHub and run them on your own machine or use Google Colab to run these using Google GPUs.

4.1 Making Predictions

This tutorial demonstrates how to use trained GaMPEN models on galaxy images to predict morphological parameters.

4.2 Training GaMPEN

This tutorial demonstrates how to train GaMPEN from scratch using your own data.

4.3 Transfer Learning with GaMPEN

This tutorial demonstrates how to train GaMPEN when starting from a pre-trained model.

Coming Soon!

4.4 Using the Auto-Cropping Module Separately

This tutorial demonstrates how to use the auto-cropping module of a trained GaMPEN model to only crop galaxy images (to be used for any task – not necessarily morphological analysis).

CHAPTER

FIVE

USING GAMPEN

On this page, we go over the most important user-facing functions that GaMPEN has and what each of these functions do. Read this page carefully to understand the various arguments/options that can be set while using these functions.

5.1 Make Splits

5.1.1 ggt.data.make_splits

Functions

ggt.data.make_splits.**main**(*data_dir*, *target_metric*)
Generates train/devel/test splits from the dataset provided.

This GaMPEN/ggt/data/make_splits.py script slices and dices the data in info.csv in a bunch of different ways to create a lot of options for training, testing, and devel (validation) sets. All these splits of the info.csv file are stored in a splits/ folder within the parent data_dir directory.

First, this will create two types of splits:- balanced and unbalanced. In the unbalanced splits, objects are picked randomly from the dataset for each split without any constraint. This function also creates balanced splits, where it first splits the dataset into 4 partitions based on the target_metric variable; and then draws samples such that the samples used for trianing are balanced across these 4 partitions.

Finally, for both balanced and unbalanced split, a number of different sub-splits are created with different fractions of data assigned to the training, devel, and test sets. The fractions assigned to each partition for the various types are mentioned below:-

- **split_types** -
  - xs - train=0.027, devel=0.003, test=0.970
  - sm - train=0.045, devel=0.005, test=0.950
  - md - train=0.090, devel=0.010, test=0.900
  - lg - train=0.200, devel=0.050, test=0.750
  - xl - train=0.450, devel=0.050, test=0.500
  - dev - train=0.70, devel=0.15, test=0.15
  - dev2 - train=0.700, devel=0.050, test=0.250

You can change these or definte your own splits. Simply alter split_types dictionary at the top of the GaMPEN/ggt/data/make_splits.py file.

5.1.2 Parameters:

- **data_dir** (*str*, required variable) - Path to the directory where the `info.csv` file is stored
- **target_metric** (*str*, default=`"bt_g"`) - Used for creating the `balanced` splits. This is the name of the column in the `info.csv` file that you want to use for creating the `balanced` splits.

5.2 Running the trainer

The backbone of GaMPEN's training feature is the `GaMPEN/ggt/train/train.py`. This script is meant to be invoked from the command line while passing in the appropriate arguments.

5.2.1 `ggt.train.train`

Functions

`ggt.train.train.`**`train`**`(`***kwargs*`)`
Trains GaMPEN models using passed arguments.

5.2.2 Parameters:

- **experiment_name** (*str*; default=`"demo"`) - MLFlow variable which controls the name of the experiment in MLFlow.
- **run_id** (*str*; default=`None`) - An MLFlow variable. This only needs to be used if you are resuming a previosuly run experiment, and want the information to be logged under the previous run.
- **run_name** (*str*; default=`None`) - The name assigned to the MLFlow run. A run is supposed to be a sub-class of an experiment in MLFLow. Typically you will have multiple runs (e.g., using multiple hyper-parameters) within an experiment.
- **model_type** (*str*; default=`"vgg16_w_stn_oc_drp"`) - The type of model you want to train. For most purposes, if you are trying to use GaMPEN as it was originally used, you should use `vgg16_w_stn_oc_drp`. We recommend referring to the source source code for information about the other options.
  - **ggt**
  - **vgg16**
  - **ggt_no_gconv**
  - **vgg16_w_stn**
  - **vgg16_w_stn_drp**
  - **vgg16_w_stn_drp_2**
  - **vgg16_w_stn_at_drp**
  - **vgg16_w_stn_oc_drp**
- **model_state** (*str*; default=`None`) - The path to a previosuly saved model file. This only needs to be set when you want to start training from a previously saved model.
- **data_dir** (*str*; required variable)- The path to the data directory containing the `info.csv` file and the `cutouts/` folder.

- **split_slug** (*str*; required variable) - This specifies which data-split is used from the `splits` folder in `data_dir`. For split, the options are `balanced`, `unbalanced` and for slug the options are `xs`, `sm`, `md`, `lg`, `xl`, `dev`, and `dev2`. Refer to the Make Splits section for more information.
- **target_metrics** (*str*; default=`"bt_g"`) - Enter the column names of `info.csv` that you want the model to learn/predict separated by a comma. For example, if you want the model to predict the `R_e` and `bt` columns, then you should enter `R_e,bt`.
- **loss** (*str*; default=`"aleatoric_cov"`) - This can be set to the following options:-
  - **mse**
  - **aleatoric**
  - **aleatoric_cov**

  For most purposes when you want full posterior distributions similar to the original GaMPEN results, you should use `aleatoric_cov`. The aleatoric covariance loss is the full loss function is given by

  $$-\log \mathcal{L} \propto \sum_n \frac{1}{2} \left[\boldsymbol{Y}_n - \hat{\boldsymbol{\mu}}_n\right]^\top \hat{\boldsymbol{\Sigma}_n}^{-1} \left[\boldsymbol{Y}_n - \hat{\boldsymbol{\mu}}_n\right] + \frac{1}{2} \log[\det(\hat{\boldsymbol{\Sigma}_n})]$$

  where Y_n is the target variable (values passed in `info.csv`); $\hat{\boldsymbol{\mu}}_n$ and $\hat{\boldsymbol{\Sigma}}_n$ are the mean and covariance matrix of the multivariate Gaussian distribution predicted by GaMPEN for an image. For an extended derivation of the loss function, please refer to Ghosh et. al. 2022.

  The `aleatoric_loss` option implements a similar loss function as above, but instead of using the full covariance matrix, it uses only the diagonal elements.

  The `mse` options implements a standard mean squared error loss function.
- **expand_data** (*int*; default=`1`) - This controls the factor by which the training data is augmented. For example, if you set this to 2, and you have 1000 images in the training set, then the training set will be expanded to 2000 images. This is useful when you want to train the model on a larger dataset.

  If you are using this option, then you should also set the `transform` option to `True`. This will ensure that the images are passed through a series of random transformations during training.
- **cutout_size** (*int*; default=`167`) - This variable is used to set the size of the input layer of the GaMPEN model. This should be set to the size of the cutouts that you are using; otherwise you will get size-mismatch errors.

  If you have cutouts that vary in size, you must ensure that all the cutouts are bigger than this `cutout_size` value, and you could use the `crop` option to crop the cutouts to this `cutot_size` value before being fed into the model.
- **channels** (*int*; default=`3`) - This variable is used to set the number of channels in the input layer of the GaMPEN model. Since GaMPEN's original based CNN is based on a VGG-16 pre-trained model, this variable is set to 3 by default.
- **n_workers** (*int*; default=`4`) - The number of workers to be used during the data loading process. You should set this to the number of CPU threads you have available.
- **batch_size** (*int*; default=`16`) - The batch size to be used during training the model. This variable specifies how many images will be processed in a single batch. This is a hyperparameter and must be tuned. The default value is a good starting point. While tuning this value, we recommend changing this by a factor of 2 each time.
- **epochs** (*int*; default=`40`) - The number of epochs you want to train GaMPEN for. Each epoch refers to all the training images being processed through the network once. You will need to optimize this value based on how much the loss function is decreasing with the number of epochs. The default value is a good starting point.
- **lr** (*float*; default=`5e-7`) - This is the learning rate to be used during the training process. This is a hyperparameter that should be tuned during the training process. The default value is a good starting point. While tuning this value, we recommend changing this by an order of magnitude each time.

- **momentum** (*float*; default=`0.9`) - The value of the momentum to be used in the gradient descent optimizer that is used to train the model. This must always be ≥ 0. This accelerates the gradient descent process. This is a hyperparameter that should be tuned during the training process. The default value is a good starting point. For tuning, we recommend trying out values between 0.8 and 0.99.
- **weight_decay** (*float*; default=`0`) - This represents the value you want to set for the L2 regularization term. This is a hyperparameter that should be tuned during the training process. The default value is a good starting point. For tuning we recommend starting from `1e-5` and increasing/decreasing by an order of magnitude each time.

  The `weight_decay` value is simply passed to the PyTorch SGD optimizer
- **parallel/ no-parallel** (*bool*; default=`True`) - The parallel argument controls whether or not to use multiple GPUs during training when they are available.

Note: The above notation (which is used for other arugments as well) implies that if you pass `--parallel` then the `parallel` argument is set to `True`. If you pass `--no-parallel` then the `parallel` argument is set to `False`.

- **normalize/no-normalize** (*bool*; default=`True`) - The normalize argument controls whether or not, the loaded images will be normalized using the `arsinh` function.
- **label_scaling** (*str*; default=`"std"`) - The label scaling option controls whether to standardize the training labels or not. Set this to `std` for sklearn's `StandardScaling()` and `minmax` for sklearn's `MinMaxScaler()`. This should usually always be set to `std` especially when using multiple target variables.
- **transform/no-transform** (*bool*; default=`True`) - If True, the training images are passed through a series of sequential random transformations:-
  - First the images are cropped to the `cutout_size` value.
  - Then the images are flipped horizontally with a 50% probability.
  - The the images are vertically flipped with a 50% probability.
  - Finally a random rotation is applied to the image with any value between 0 and 360 degrees.

  All the above transformations are performed using the kornia library.
- **crop/no-crop** (*bool*; default=`True`) - If True, all images are passed through a cropping operation before being fed into the network. Images are cropped to the `cutout_size` parameter.
- **nesterov/no-nesterov** (*bool*; default=`False`) - Whether to use Nesterov momentum or not. This variable is simply passsed to the PyTorch SGD optimizer. This is a hyperparameter that should be tuned during the training process.
- **repeat_dims/no-repeat_dims** (*bool*; default=`True`) - When you have a multichannel network and you are feeding in images with only one channel, you should set this parameter to `True`. This automatically repeats the image as many times as the number of channels in the network, while data-loading.
- **dropout_rate** (*float*; default=`None`) - The dropout rate to use for all the layers in the model. If this is set to None, then the default dropout rate in the specific model is used.

> **Attention:** The `dropout_rate` is an important hyperparameter that among other things, also controls the predicted epistemic uncertainity when using Monte Carlo Dropout. This hyperparameter should be tuned in order to achieve callibrated coverage probabilities.
>
> We recommend tuning the `dropout_rate` once you have tuned all other hyperparameters. Refer to Ghosh et. al. 2022 and Ghosh et. al. 2022b for more details on how we tuned this hyperparameter.

5.3 Inference

5.3.1 ggt.modules.inference

Functions

ggt.modules.inference.**main**(*model_path*, *output_path*, *data_dir*, *cutout_size*, *channels*, *parallel*, *slug*, *split*, *normalize*, *batch_size*, *n_workers*, *label_cols*, *model_type*, *repeat_dims*, *label_scaling*, *mc_dropout*, *dropout_rate*, *transform*, *errors*, *cov_errors*, *n_runs*, *ini_run_num*)

The `GaMPEN/ggt/modules/inference.py` script provides users the functionality to perform predictions on images using trained GaMPEN models.

5.3.2 Parameters

- **model_type** (*str*; default=`"vgg16_w_stn_oc_drp"`) - Same as the model types mentioned in Running the Trainer section previously. If using our pre-trained models, this should be set to `vgg16_w_stn_oc_drp`.
- **model_path** (*str*; required variable)- The full path to the trained `.pt` model file which you want to use for performing prediction.

  > **Attention:** The model path should be enclosed in single quotes `'/path/to/model/xxxxx.pt'` and NOT within double quotes `"/path/to/model/xxxxx.pt"`. If you encluse it within double quotes, then the script will throw up an error.

- **output_path** (*str*; required variable) - The full path to the output directory where the predictions of the model will be stored.
- **data_dir** (*str*; required variable) - The full path to the data directory that should contain a `cutouts` folder with all the images that you want to perform predictions on as well as an `info.csv` file that contains the filenames for all the images. For more information on how to create this directory structure during performing inference, plese refer to the Predictions Tutorial
- **cutout_size** (*int*; default=`167`) - Size of the input image that the model takes as input. For our pre-trained models, this should be set to `239`, `143`, `96` for the low, mid, and high redshift models respectively.
- **channels** (*int*; default=`3`) - Number of channels in the input image. For our pre-trained models, this should be set to `3`.
- **slug** (*str*; required variable) - This specifies which slug (balanced/unbalanced xs, sm, lg, dev, dev2) is used to perform predictions on. Each slug refers to a different way to split the data into train, devel, and test sets. For consistent results, you should set this to the same slug that was used to train the model. For more information on the fraction of data assigned to the train/deve/test sets for each slug, please refer to the `make_splits` function.

  If you are performing predictions on a dataset for which you don't have access to the ground truth labels (and thus you haven't run `make_splits`), this should be set to `None` as shown in the Predictions Tutorial.
- **split** (*str*; default=`"test"`) - The split of the data that you want to perform predictions on. This should be set to `test` if you are performing predictions on the test set. If you are performing predictions on the train or devel set, this should be set to `train` or `devel` respectively.

  If you are performing predictions on a dataset for which you don't have access to the ground truth labels (and thus you haven't run `make_splits`), this should be set to `None` as shown in the Predictions Tutorial.

- **normalize/no-normalize** (*bool*; default=`True`) - The normalize argument controls whether or not, the loaded images will be normalized using the `arsinh` function. This should be set to the same value as what was used during training the model.
- **label_scaling** (*str*; default=`"std"`) - The label scaling option controls whether to perform an inverse-transformation on the predicted values. Set this to `std` for sklearn's `StandardScaling()` and `minmax` for sklearn's `MinMaxScaler()`. This should usually always be set to `std` especially when using multiple target variables.

  Note that you should pass the same argument for `label_scaling` as was used during the training phase (of the model being used for inference). For all our pre-trained models, this should be set to `std`."
- **batch_size** (*int*; default=`256`) - The batch size to be used during inference. This specfies how many images will be processed in a single batch. During inference, the only consideration is to keep the batch size small enough so that the batch can be fit within the memory of the GPU.
- **n_workers** (*int*; default=`4`) - The number of workers to be used during the data loading process. You should set this to the number of threads you have access to.
- **parallel/ no-parallel** (*bool*; default=`True`) - The parallel argument controls whether or not to use multiple GPUs when they are available.

  Note that this variable needs to be set to whatever value was used during the training phase (of the model being used for inference). For all our pre-trained models, this should be set to `parallel`
- **label_cols** (*str*; default=`bt_g`) - Enter the label column(s) separated by commas. Note that you should pass the exactly same argument for label_cols as was used during the training phase (of the model being used for inference)
- **repeat_dims/no-repeat_dims** (*bool*; default=`True`) - In case of multi-channel data, whether to repeat a two dimensional image as many times as the number of channels. Note that you should pass the exactly same argument for `repeat_dims` as was used during the training phase (of the model being used for inference). For all our pre-trained models, this should be set to `repeat_dims`
- **mc_dropout/no-mc_dropout** (*bool*; default=`True`) - Turns on Monte Carlo dropout during inference. For most cases, this should be set to `mc_dropout`.
- **n_runs** (*int*; default=`1`) - The number of different models that will be generated using Monte Carlo dropout and used for infererence.
- **ini_run_num** (*int*; default=`1`) - Specifies the starting run-number for `n_runs`. For example, if `n_runs` is set to 5 and `ini_run_num` is set to 10, then the output csv files will be named as `inf_10.csv`, `inf_11.csv`, `inf_12.csv`, `inf_13.csv`, `inf_14.csv`.
- **dropout_rate** (*float*; default=`None`) - This should be set to the dropout rate that was used while training the model.
- **transform/no-transform** (*bool*; default=`False`) - If `True`, the images are passed through a cropping transformation to ensure proper cutout size. This should be left on for most cases.

  Attention: Note that if you set this to True and then use cutouts which have a smaller size than the `cutout_size`, this will lead to unpredictable behaviour.

- **errors/no-errors** (*bool*, default=`False`) - If True and if the model allows for it, aleatoric uncertainties are written to the output file. Only set this to True if you trained the model with `aleatoric` loss.
- **cov_errors/no-cov_errors** (*bool*, default=`False`) - If True and if the model allows for it, aleatoric uncertainties with full covariance conisdered are written to the output file. Only set this to True if you trained the model with `aleatoric_cov` loss. For our pre-trained models, this should be set to `cov_errors`.

- **labels/no-labels** (*bool*, default=`True`)- If True, this means you have labels available for the dataset. If False, this means that you have no labels available and want to perform predictions on a dataset for which you don't know the ground truth labels.

  This primarily used to control which files are used to perform scaling the prediction variables. If `--no-labels`, then you need to specify the data directory and slug that should be used to perform the scaling. If `--labels`, then the `scaling_data_dir` and `scaling_slug` are automatically set to values for `data_dir` and `slug` provided before.

- **scaling_data_dir** (*str*; default=`None`) - The data directory that should be used to perform unscaling of the prediction variables. You should only set this if using `--no-labels`.

  This scaling refers to the `label_scaling` variable that you passed before. Essentially to inverse transform the predictions, we need access to the original scaling parameters that were used to scale the data during trianing. In case you are using a pre-trained model directly on some data for which you have no labels, you need to point this to the `/splits` folder of the data-directory that was used to train the model. For all our pre-trained models, we make the relevant scaling files available. Refer to the Predictions Tutorial for a demonstration.

- **scaling_slug** (*str*; default=`None`) - This needs to be set only if you are using `--no-labels`. This specifies which slug (`balanced/unbalanced`, `xs`, `sm`, `lg`, `dev`,`dev2`) corresponding to the scaling_data_dir that should be used to perform the data scaling on.

  For example, if you want a `balanced-dev2-train.csv` file in the `scaling_data_dir` , then you should set this to `balanced-dev2`. Refer to the Predictions Tutorial for a demonstration.

5.4 Result Aggregator

The `GaMPEN/ggt/modules/result_aggregator.py` module is used to aggregate the prediction `.csv` files generated by the inference module.

Attention: The unsccaling properties of the `result_aggregator` module is mostly useful when you are predicting variables similar to the ones used in Ghosh et. al. 2022.

If you are using your own custom scaling of variables (or predicting other variables), then you will need to run the `result_aggregator` module with `--no-unscale` and perform the unscaling of variables yourself. Alternatively, you can also choose to alter the `unscale_preds` function in the `result_aggregator.py` module to suit your needs.

Attention: The result aggregator module also converts flux to magnitudes. However, this conversion is only valid for HSC. If you are using the module for some other survey, please alter the magnitude conversion line in the `unscale_preds` function of `result_aggregator.py` or ignore the mangitudes produced by the `result_aggregator` module.

5.4.1 ggt.modules.result_aggregator

Functions

ggt.modules.result_aggregator.**main**(*data_dir*, *num*, *out_summary_df_path*, *out_pdfs_path*, *unscale*, *scaling_df_path*, *drop_old*)

5.4.2 Parameters

- **data_dir** (*str*; required variable) - Full path to the directory that has the prediction csv files that need to be aggregated.
- **num** (*int*; default=`500`) - The number of prediction csv files that need to be aggregated.
- **out_summary_df_path** (*str*; required variable) - Full path to the output csv file that will contain the summary statistics.
- **out_pdfs_path** (*str*; required variable) - Full path to the output directory that will contain the posterior distribution functions (PDFs) of the predicted output variables for each galaxy.
- **unscale/no-unscale** (*bool*, default=`False`) - If `True`, the predictions are unscaled using the information `scaling_df_path`. This unscaling is for the inverse logit and logarithmic tansformations (e.g., converting $\log R_e$ to R_e)

  This is only useful if you are using our pre-trained models/you are predicted the same variables as in Ghosh et. al. 2022. For all other cases, if you want to set this to `True`, you will need to modify the `unscale_preds` function in `result_aggregator.py` according to the variables you are predicting and the transformations you made to them during training.

  > **Attention:** In order to make sure that you are not making a mistake, the module will throw an error if you are using the `--unscale` option and the inference `.csvs` do not have the column names exactly as is expected for our trained models (i.e., `custom_logit_bt`,`ln_R_e_asec`,`ln_total_flux_adus`).
  >
  > When you are using using some different scaling, you need to run this script with `--no-unscale` and transform the predictions yourself. Or you can also alter the `unscale_preds` function in `result_aggregator.py` according to your needs.

- **scaling_df_path** (*str*; default=`None`) - Full path to the `info.csv` file that contains the scaling information. This is only used if `unscale` is set to True.

  This is needed to perform the inverse logit transforamtion. As the logit transformation goes to infinty at the edges of the variable space and we need to perform an approximation. To perform this approximation, we need access to the `info.csv` file that was used during training. We make the `info.csv` files for all our pre-trained models available. Refer to the Predictions Tutorial for a demonstration.

- **drop_old/no-drop_old** (*bool*; default=`True`)- If `True`, the unscaled prediction columns will be dropped.

5.5 AutoCrop

5.5.1 ggt.modules.autocrop

Functions

ggt.modules.autocrop.**main**(*model_type*, *model_path*, *cutout_size*, *channels*, *n_pred*, *image_dir*, *out_dir*, *normalize*, *transform*, *repeat_dims*, *parallel*, *cov_errors*, *errors*)

The `GaMPEN/ggt/modules/autocrop.py` script provides users the functionality to perform cropping using a trained GaMPEN model and then save these cropped images as fits files for further analysis.

5.5.2 Parameters

- **model_type** (*str*; default=`"vgg16_w_stn_oc_drp"`) - Same as the model types mentioned in Running the Trainer section previously. If using our pre-trained models, this should be set to `vgg16_w_stn_oc_drp`.
- **model_path** (*str*; required variable)- The full path to the trained `.pt` model file which you want to use for performing prediction.

  > **Attention:** The model path should be enclosed in single quotes `'/path/to/model/xxxxx.pt'` and NOT within double quotes `"/path/to/model/xxxxx.pt"`. If you encluse it within double quotes, then the script will throw up an error.

- **cutout_size** (*int*; default=`167`) - Size of the input image that the model takes as input. For our pre-trained models, this should be set to `239`, `143`, `96` for the low, mid, and high redshift models respectively.
- **channels** (*int*; default=`3`) - Number of channels in the input image. For our pre-trained models, this should be set to `3`.
- **n_pred** (*int*; default=`1`) - Number of output variables that were used while training the model.
- **image_dir** (*str*; required variable) - Full path to the directory that contains the images that need to be cropped.
- **out_dir** (*str*; required variable) - Full path to the directory where the cropped images will be saved.
- **normalize/no-normalize** (*bool*; default=`True`) - The normalize argument controls whether or not, the loaded images will be normalized using the `arsinh` function. This should be set to the same value as what was used during training the model.
- **transform/no-transform** (*bool*; default=`True`) - The transform argument controls whether or not, the loaded images will be cropped to the mentioned `cutout_size` while being loaded. This should be set to `True` for most cases.
- **repeat_dims/no-repeat_dims** (*bool*; default=`True`) - In case of multi-channel data, whether to repeat a two dimensional image as many times as the number of channels. Note that you should pass the exactly same argument for `repeat_dims` as was used during the training phase (of the model being used for inference). For all our pre-trained models, this should be set to `repeat_dims`
- **parallel/ no-parallel** (*bool*; default=`True`) - The parallel argument controls whether or not to use multiple GPUs when they are available.

  Note that this variable needs to be set to whatever value was used during the training phase (of the model being used for inference). For all our pre-trained models, this should be set to `parallel`
- **errors/no-errors** (*bool*, default=`False`) - If True and if the model allows for it, aleatoric uncertainties are written to the output file. Only set this to True if you trained the model with `aleatoric` loss.

- **cov_errors/no-cov_errors** (*bool*, default=`False`) - If True and if the model allows for it, aleatoric uncertainties with full covariance conisdered are written to the output file. Only set this to True if you trained the model with `aleatoric_cov` loss. For our pre-trained models, this should be set to `cov_errors`.

CHAPTER

SIX

PUBLIC DATA RELEASE HANDBOOK

Note: We are still rolling out the full data release of HSC PDR2 morphological parameters. This page will be updated continuously through the Spring of 2023. If you can't find the portion of the data-release that you need, please drop us a line!

6.1 FTP Server

All components of the public data release are hosted on the Yale Astronomy FTP server `ftp.astro.yale.edu`. There are multiple ways you can access the FTP server, and we summarize some of the methods below.

6.1.1 Using Unix Command Line

```
ftp ftp.astro.yale.edu
cd pub/hsc_morph/<appropriate_subdirectory>
```

If prompted for a username, try `anonymous` and keep the password field blank. After connecting, you can download files using the `get` command.

6.1.2 Using a Browser

Navigate to `ftp://ftp.astro.yale.edu/pub/<appropriate_subdirectory>`

6.1.3 Using Finder on OSX

Open Finder, and then choose Go → Connect to Server (or command + K) and enter `ftp://ftp.astro.yale.edu/pub/hsc_morph/`. Choose to connect as `Guest` when prompted.

Thereafter, navigate to the appropriate subdirectory.

6.2 Hyper Suprime-Cam Wide PDR2 Morphology

6.2.1 Prediction Tables

The prediction tables are located at the following subdirectories on the FTP server:

- g-band HSC-Wide z < 0.25 galaxies → `/pub/hsc_morph/g_0_025/g_0_025_preds_summary.csv`
- r-band HSC-Wide 0.25 < z < 0.50 galaxies → `/pub/hsc_morph/r_025_050/r_025_050_preds_summary.csv`
- i-band HSC-Wide 0.50 < z < 0.75 galaxies → `/pub/hsc_morph/i_050_075/i_050_075_preds_summary.csv`

The various columns in the prediction tables are described below:

- `object_id`: The unique object ID for the galaxy. This is the same as the `object_id` in the HSC-Wide PDR2 catalog.
- `ra`: The right ascension of the galaxy in degrees.
- `dec`: The declination of the galaxy in degrees.
- `z_best`: The redshift of the galaxy. This is the same as the `z_best` in the HSC-Wide PDR2 catalog.
- `zmode`: The redshift mode of the galaxy. The two options are `specz` or `photz`.

There are multiple columns for each of the three morphological parmaeters: effective radius (`R_e`) (in arcsec), bulge-to-total_light_ratio (`bt`), total flux (`total_flux`) (in ADUs), and mangitude (`total_mag`). In all the columns below `xx` refers to the column names mentioned in brackets.

- `preds_xx_mode`: The mode of the posterior distribution for the morphological parameter. (**Recommended**)
- `preds_xx_mean`: The mean of the posterior distribution of the morphological parameter.
- `preds_xx_median`: The median of the posterior distribution of the morphological parameter.
- `preds_xx_std`: The standard deviation of the posterior distribution of the morphological parameter.
- `preds_xx_skew`: The skewness of the posterior distribution of the morphological parameter.
- `preds_xx_kurtosis`: The kurtosis of the posterior distribution of the morphological parameter.
- `preds_xx_sig_ci`: The 1-sigma confidence interval of the posterior distribution of the morphological parameter.
- `preds_xx_twosig_ci`: The 2-sigma confidence interval of the posterior distribution of the morphological parameter.
- `preds_xx_threesig_ci`: The 3-sigma confidence interval of the posterior distribution of the morphological parameter.

6.2.2 Posterior Distribution Files for Individual Galaxies

The predicted posterior distributions for individual galaxies are available as `.npy` files. The files are named as `zz.npy` where zz is the `object_id` mentioned in the prediction tables. The files located at the following subdirectories on the FTP server:

- g-band HSC-Wide z < 0.25 galaxies → `/pub/hsc_morph/g_0_025/posterior_arrays/`
- r-band HSC-Wide 0.25 < z < 0.50 galaxies → `/pub/hsc_morph/r_025_050/posterior_arrays/`
- i-band HSC-Wide 0.50 < z < 0.75 galaxies → `/pub/hsc_morph/i_050_075/posterior_arrays/`

You can load the `.npy` files using the `np.load` function in `Numpy`. The array dimensions are as follows:

- 0 → x of radius (in arcsec)
- 4 → y of radius (in arcsec)
- 1 → x of flux (in ADUs)
- 5 → y of flux (in ADUs)
- 2 → x of bulge-to-total_light_ratio
- 6 → y of bulge-to-total_light_ratio
- 3 → x of magnitude
- 7 → y of magnitude

6.2.3 Trained GaMPEN Models

The trained GaMPEN models are available as `.pt` PyTorch files. The models are at the following locations:-

Real Data Models

- g-band HSC-Wide z < 0.25 galaxies → `/pub/hsc_morph/g_0_025/trained_model/g_0_025_model.pt`
- r-band HSC-Wide 0.25 < z < 0.50 galaxies → `/pub/hsc_morph/r_025_050/trained_model/r_025_050_model.pt`
- i-band HSC-Wide 0.50 < z < 0.75 galaxies → `/pub/hsc_morph/i_050_075/trained_model/i_050_075_model.pt`

Simulated Data Models

- Simulated g-band HSC-Wide z < 0.25 galaxies → `/pub/hsc_morph/sim_g_0_025/trained_model/sim_g_0_025.pt`
- Simulated r-band HSC-Wide 0.25 < z < 0.50 galaxies → `/pub/hsc_morph/sim_r_025_050/trained_model/sim_r_025_050.pt`
- Simulated i-band HSC-Wide 0.50 < z < 0.75 galaxies → `/pub/hsc_morph/sim_i_050_075/trained_model/sim_i_050_075.pt`

Trained Model Parameters

We mention some of the finally tuned hyper-parameters that we used for the above models. Note that while performing inference using the above models, you will need to use some of these parameters.

Real Data Models

Parameter Name	Low-z Real Data	Mid-z Real Data	High-z Real Data
model_type	vgg16_w_stn_oc_drp	vgg16_w_stn_oc_drp	vgg16_w_stn_oc_drp
cutout_size	239	143	96
droput_rate	0.0004	0.0002	0.0002
label_scaling	std	std	std
loss	aleatoric_cov	aleatoric_cov	aleatoric_cov
lr	5e-8	5e-8	5e-6
momentum	0.99	0.99	0.99
nesterov	False	False	False
weight_decay	0.0001	0.0001	0.0001
parallel	True	True	True
target_metrics	custom_logit_bt, ln_R_e_asec, ln_total_flux_adus	custom_logit_bt, ln_R_e_asec, ln_total_flux_adus	custom_logit_bt, ln_R_e_asec, ln_total_flux_adus
split_slug	balanced-dev2	balanced-dev2	balanced-dev2

Simulated Data Models

Parameter Name	Low-z Sims.	Mid-z Sims.	High-z Sims.
model_type	vgg16_w_stn_oc_drp	vgg16_w_stn_oc_drp	vgg16_w_stn_oc_drp
cutout_size	239	143	96
droput_rate	0.0007	0.0007	0.0004
label_scaling	std	std	std
loss	aleatoric_cov	aleatoric_cov	aleatoric_cov
lr	5e-7	5e-7	5e-7
momentum	0.99	0.99	0.99
nesterov	False	False	False
weight_decay	0.0001	0.0001	0.0001
parallel	True	True	True
target_metrics	custom_logit_bt, ln_R_e_asec, ln_total_flux_adus	custom_logit_bt, ln_R_e_asec, ln_total_flux_adus	custom_logit_bt, ln_R_e_asec, ln_total_flux_adus
split_slug	balanced-dev	balanced-dev	balanced-dev

Scaling Data

Note that as mentioned in the Predictions Tutorial, in order to unscale the predictions made using the above models, you need access to the training files.

You can access these files at the following locations using `wget`:

`ftp://ftp.astro.yale.edu/pub/hsc_morph/xxxx/scaling_data_dir/info.csv`

and

`ftp://ftp.astro.yale.edu/pub/hsc_morph/xxxx/scaling_data_dir/splits/`

where `xxxx` is `g_0_025`, `r_025_050`, or `i_050_075` for low-, mid-, and high-z real data models respectively; and `sim_g_0_025`, `sim_r_025_050`, or `sim_i_050_075` for low-, mid-, and high-z simulated data models respectively.

Custom Scaling Function

As mentioned in the *Tutorials*, all the trained GaMPEN models first make predictions in the `logit(bulge-to-total light ratio)` space. The predictions are then scaled to the `bulge-to-total light ratio` space using the custom inverse-scaling function defined in `/GaMPEN/ggt/modules/result_aggregator.py`.

Here, for completeness, we provide the custom scaling function that we used for the **forward** logit transformation while creating our `info.csv` files. The only way this is different from the standard logit transformation is that we prevent the function from blowing up for values of `bulge-to-total_light_ratio` that are very close to 0 or 1.

```
from scipy.special import logit

def logit_custom(x_input):

    '''Handling for 0s and 1s while doing a
       logit transformation

       x_input should be the entire column/array
       in info.csv over which you are applying
       the transformation'''

    x = np.array(x_input)

    if np.min(x) < 0 or np.max(x) > 1:
        raise ValueError("x must be between 0 and 1")

    if np.min(x) == 0:
        min_x = np.min(x[x != 0])
        add_epsilon = min_x/2.0
        x[np.where(x==0)[0]] = add_epsilon

    if np.max(x) == 1:
        max_x = np.max(x[x != 1])
        sub_epsilon = (1-max_x)/2.0
        x[np.where(x==1)[0]] = 1.0 - sub_epsilon

    return logit(x)
```

CHAPTER

SEVEN

API REFERENCE

This page contains auto-generated API reference documentation[1].

7.1 ggt

7.1.1 Subpackages

ggt.data

Submodules

ggt.data.dataset

Module Contents

Classes

FITSDataset	Dataset from FITS files. Pre-caches FITS files as PyTorch tensors to

class ggt.data.dataset.**FITSDataset**(*data_dir*, *slug=None*, *split=None*, *channels=1*, *cutout_size=167*, *label_col='bt_g'*, *normalize=True*, *transform=None*, *expand_factor=1*, *repeat_dims=False*, *label_scaling=None*, *scaling_data_dir=None*, *scaling_slug=None*, *load_labels=True*)

Bases: `torch.utils.data.Dataset`

Dataset from FITS files. Pre-caches FITS files as PyTorch tensors to improve data load speed.

__getitem__(*index*)
Magic method to index into the dataset.

__len__()
Return the effective length of the dataset.

static load_fits_as_tensor(*filename*)
Open a FITS file and convert it to a Torch tensor.

[1] Created with sphinx-autoapi

ggt.data.make_splits

Module Contents

Functions

interleave(L)	
make_splits(x, weights[, split_col])	
main(data_dir, target_metric)	Generate train/devel/test splits from the dataset provided.

Attributes

split_types	
log_fmt	

ggt.data.make_splits.**split_types**

ggt.data.make_splits.**interleave**(*L*)

ggt.data.make_splits.**make_splits**(*x*, *weights*, *split_col=None*)

ggt.data.make_splits.**main**(*data_dir*, *target_metric*)
Generate train/devel/test splits from the dataset provided.

ggt.data.make_splits.**log_fmt = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'**

Package Contents

Classes

FITSDataset	Dataset from FITS files. Pre-caches FITS files as PyTorch tensors to

Functions

get_data_loader(dataset, batch_size, n_workers[, shuffle])	

class ggt.data.**FITSDataset**(*data_dir*, *slug=None*, *split=None*, *channels=1*, *cutout_size=167*, *label_col='bt_g'*, *normalize=True*, *transform=None*, *expand_factor=1*, *repeat_dims=False*, *label_scaling=None*, *scaling_data_dir=None*, *scaling_slug=None*, *load_labels=True*)

Bases: `torch.utils.data.Dataset`

Dataset from FITS files. Pre-caches FITS files as PyTorch tensors to improve data load speed.

__getitem__(*index*)
Magic method to index into the dataset.

__len__()
Return the effective length of the dataset.

static load_fits_as_tensor(*filename*)
Open a FITS file and convert it to a Torch tensor.

ggt.data.**get_data_loader**(*dataset*, *batch_size*, *n_workers*, *shuffle=True*)

ggt.losses

Submodules

ggt.losses.aleatoric_cov_loss

Module Contents

Functions

aleatoric_cov_loss(outputs, targets[, num_var, average])	Computes the Aleatoric Loss while including the full

ggt.losses.aleatoric_cov_loss.**aleatoric_cov_loss**(*outputs*, *targets*, *num_var=3*, *average=True*)

Computes the Aleatoric Loss while including the full covariance matrix of the outputs.

If you are predicting for n output variables, then the number of output neuros required for this loss is (3n + n^2)/2.

Args: outputs: (tensor) - predicted outputs from the model targets: (tensor) - ground truth labels size_average: (bool) - if True, the losses are

averaged over all elements of the batch

Returns: aleatoric_cov_loss: (tensor) - aleatoric loss

Formula:

loss = 0.5 * [Y - Y_hat].T * cov_mat_inv

- [Y - Y_hat] + 0.5 * log(det(cov_mat))

ggt.losses.aleatoric_loss

Module Contents

Functions

aleatoric_loss(outputs, targets[, average])	Computes the aleatoric loss.

ggt.losses.aleatoric_loss.**aleatoric_loss**(*outputs*, *targets*, *average=True*)

Computes the aleatoric loss. Args:

outputs: (tensor) - predicted outputs from the model targets: (tensor) - ground truth labels size_average: (bool) - if True, the losses are

averaged over all elements of the batch

Returns: aleatoric_loss: (tensor) - aleatoric loss

ggt.losses.losses

Module Contents

Classes

AleatoricLoss	Base class for all neural network modules.
AleatoricCovLoss	Base class for all neural network modules.

class ggt.losses.losses.**AleatoricLoss**(*average=True*)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

forward(*outputs*, *targets*)

class ggt.losses.losses.**AleatoricCovLoss**(*num_var=3*, *average=True*)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

forward(*outputs*, *targets*)

Package Contents

Classes

AleatoricLoss	Base class for all neural network modules.
AleatoricCovLoss	Base class for all neural network modules.

class ggt.losses.**AleatoricLoss**(*average=True*)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)
```

(continues on next page)

(continued from previous page)

```
    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

forward(*outputs*, *targets*)

class ggt.losses.**AleatoricCovLoss**(*num_var=3*, *average=True*)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

forward(*outputs*, *targets*)

ggt.metrics

Submodules

ggt.metrics.elementwise_mae

Module Contents

Classes

ElementwiseMae	Calculates the element-wise mean absolute error.

class ggt.metrics.elementwise_mae.**ElementwiseMae**(*output_transform: Callable = lambda x: ...*, *device: Optional[Union[str, torch.device]] = None*)

Bases: `ignite.metrics.Metric`

Calculates the element-wise mean absolute error.

reset() → None
Resets the metric to it's initial state.

By default, this is called at the start of each epoch.

update(*output: Sequence[torch.Tensor]*) → None
Updates the metric's state using the passed batch output.

By default, this is called once for each batch.

Args: output: the is the output from the engine's process function.

compute() → Union[float, torch.Tensor]
Computes the metric based on it's accumulated state.

By default, this is called at the end of each epoch.

Returns:

Any: | the actual quantity of interest. However, if a Mapping is returned, it will be (shallow) flattened into *engine.state.metrics* when completed() is called.

Raises: NotComputableError: raised when the metric cannot be computed.

Package Contents

Classes

ElementwiseMae	Calculates the element-wise mean absolute error.

class ggt.metrics.**ElementwiseMae**(*output_transform: Callable = lambda x: ...*, *device: Optional[Union[str, torch.device]] = None*)

Bases: ignite.metrics.Metric

Calculates the element-wise mean absolute error.

reset() → None
Resets the metric to it's initial state.

By default, this is called at the start of each epoch.

update(*output: Sequence[torch.Tensor]*) → None
Updates the metric's state using the passed batch output.

By default, this is called once for each batch.

Args: output: the is the output from the engine's process function.

compute() → Union[float, torch.Tensor]
Computes the metric based on it's accumulated state.

By default, this is called at the end of each epoch.

Returns:

Any: | the actual quantity of interest. However, if a Mapping is returned, it will be (shallow) flattened into *engine.state.metrics* when completed() is called.

Raises: NotComputableError: raised when the metric cannot be computed.

ggt.models

Submodules

ggt.models.ggt

Module Contents

Classes

GGT	Galaxy Group-Equivariant Transformer model.
GGTNoGConv	Galaxy Group-Equivariant Transformer model with no group

class ggt.models.ggt.**GGT**(*cutout_size*, *channels*, *n_out=1*, *dropout=0.5*)
Bases: torch.nn.Module

Galaxy Group-Equivariant Transformer model.

setup_stn(*input_shape*)

setup_featurizer()

setup_regression()

setup_pooling(*input_shape=(6, 6)*)

setup_dropout(*dropout*)

spatial_transform(*x*)

forward(*x*)

class ggt.models.ggt.**GGTNoGConv**(*cutout_size*, *channels*, *n_out=1*, *dropout=0.5*)
Bases: *GGT*

Galaxy Group-Equivariant Transformer model with no group convolutional layers.

setup_featurizer()

forward(*x*)

ggt.models.ggt_no_gcov

Module Contents

Classes

GGT_no_gconv	Galaxy Group-Equivariant Transformer model.

class ggt.models.ggt_no_gcov.**GGT_no_gconv**(*cutout_size*, *channels*, *n_out=1*)
Bases: torch.nn.Module

Galaxy Group-Equivariant Transformer model.

spatial_transform(*x*)

forward(*x*)

ggt.models.vgg

Module Contents

Functions

vgg16(cutout_size, channels[, n_out, pretrained])	

ggt.models.vgg.**vgg16**(*cutout_size*, *channels*, *n_out=1*, *pretrained=True*)

ggt.models.vgg16_w_stn_at_drp

Module Contents

Classes

vgg16_w_stn_at_drp	Base class for all neural network modules.

class ggt.models.vgg16_w_stn_at_drp.**vgg16_w_stn_at_drp**(*cutout_size*, *channels*, *n_out=1*, *pretrained=True*, *dropout=False*, *dropout_rate=0.5*)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

spatial_transform(*x*)

forward(*x*)

ggt.models.vgg16_w_stn_drp

Module Contents

Classes

vgg16_w_stn_drp	Base class for all neural network modules.

class ggt.models.vgg16_w_stn_drp.**vgg16_w_stn_drp**(*cutout_size*, *channels*, *n_out=1*, *pretrained=True*, *dropout=False*, *dropout_rate=0.5*)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

spatial_transform(*x*)

forward(*x*)

ggt.models.vgg16_w_stn_drp_2

Module Contents

Classes

vgg16_w_stn_drp_2	Base class for all neural network modules.

class ggt.models.vgg16_w_stn_drp_2.**vgg16_w_stn_drp_2**(*cutout_size*, *channels*, *n_out=1*, *pretrained=True*, *dropout=False*, *dropout_rate=0.5*)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

spatial_transform(*x*)

forward(*x*)

ggt.models.vgg16_w_stn_oc_drp

Module Contents

Classes

vgg16_w_stn_oc_drp	Base class for all neural network modules.

class ggt.models.vgg16_w_stn_oc_drp.**vgg16_w_stn_oc_drp**(*cutout_size*, *channels*, *n_out=1*, *pretrained=True*, *dropout=False*, *dropout_rate=0.5*)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F
```

(continues on next page)

(continued from previous page)

```
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

spatial_transform(*x*)

forward(*x*)

Package Contents

Classes

GGT	Galaxy Group-Equivariant Transformer model.
GGT_no_gconv	Galaxy Group-Equivariant Transformer model.
vgg16_w_stn_drp	
vgg16_w_stn_drp_2	Base class for all neural network modules.
vgg16_w_stn_at_drp	
vgg16_w_stn_oc_drp	

Functions

vgg16(cutout_size, channels[, n_out, pretrained])	
model_stats(model)	
model_factory(modeltype)	
save_trained_model(model, slug)	

class ggt.models.**GGT**(*cutout_size*, *channels*, *n_out=1*, *dropout=0.5*)

Bases: `torch.nn.Module`

Galaxy Group-Equivariant Transformer model.

setup_stn(*input_shape*)

setup_featurizer()

setup_regression()

setup_pooling(*input_shape=(6, 6)*)

setup_dropout(*dropout*)

spatial_transform(*x*)

forward(*x*)

class ggt.models.**GGT_no_gconv**(*cutout_size*, *channels*, *n_out=1*)

Bases: torch.nn.Module

Galaxy Group-Equivariant Transformer model.

spatial_transform(*x*)

forward(*x*)

ggt.models.**vgg16**(*cutout_size*, *channels*, *n_out=1*, *pretrained=True*)

class ggt.models.**vgg16_w_stn_drp**(*cutout_size*, *channels*, *n_out=1*, *pretrained=True*, *dropout=False*, *dropout_rate=0.5*)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call to(), etc.

spatial_transform(*x*)

forward(*x*)

class ggt.models.**vgg16_w_stn_drp_2**(*cutout_size*, *channels*, *n_out=1*, *pretrained=True*, *dropout=False*, *dropout_rate=0.5*)

Bases: torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

spatial_transform(*x*)

forward(*x*)

class ggt.models.**vgg16_w_stn_at_drp**(*cutout_size*, *channels*, *n_out=1*, *pretrained=True*, *dropout=False*, *dropout_rate=0.5*)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

spatial_transform(*x*)

forward(*x*)

class ggt.models.**vgg16_w_stn_oc_drp**(*cutout_size*, *channels*, *n_out=1*, *pretrained=True*, *dropout=False*, *dropout_rate=0.5*)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

spatial_transform(*x*)

forward(*x*)

ggt.models.**model_stats**(*model*)

ggt.models.**model_factory**(*modeltype*)

ggt.models.**save_trained_model**(*model*, *slug*)

ggt.modules

Submodules

ggt.modules.autocrop

Module Contents

Functions

main(model_type, model_path, cutout_size, channels, ...)	Using the spatial transformer layer of the model defined in *model_path*,

Attributes

log_fmt

ggt.modules.autocrop.**main**(*model_type*, *model_path*, *cutout_size*, *channels*, *n_pred*, *image_dir*, *out_dir*, *normalize*, *transform*, *repeat_dims*, *parallel*, *cov_errors*, *errors*)

Using the spatial transformer layer of the model defined in *model_path*, write cropped versions of each image in *image_dir* back to disk.

ggt.modules.autocrop.**log_fmt = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'**

ggt.modules.inference

Module Contents

Functions

predict(model_path, dataset, cutout_size, channels[, ...])	Using the model defined in model path, return the output values for
main(model_path, output_path, data_dir, cutout_size, ...)	

Attributes

log_fmt

ggt.modules.inference.**predict**(*model_path*, *dataset*, *cutout_size*, *channels*, *parallel=False*, *batch_size=256*, *n_workers=1*, *model_type='ggt'*, *n_out=1*, *mc_dropout=False*, *dropout_rate=None*)

Using the model defined in model path, return the output values for the given set of images

ggt.modules.inference.**main**(*model_path*, *output_path*, *data_dir*, *cutout_size*, *channels*, *parallel*, *slug*, *split*, *normalize*, *batch_size*, *n_workers*, *label_cols*, *model_type*, *repeat_dims*, *label_scaling*, *mc_dropout*, *dropout_rate*, *transform*, *errors*, *cov_errors*, *n_runs*, *ini_run_num*, *labels*, *scaling_data_dir*, *scaling_slug*)

ggt.modules.inference.**log_fmt = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'**

ggt.modules.result_aggregator

Module Contents

Functions

calculate_cdf(data[, n, warnings])	
find_idx_nearest_val(array, value)	
calculate_ci(data[, n, warnings])	
expit_custom(x_input[, scaling_array])	Performs a custom implementation of Scipy's expit
unscale_preds(df_input[, scaling_df_path, drop_old])	
file_loader(args)	

continues on next page

Table 25 – continued from previous page

bayesian_inference_file_gobbler(data_dir[, num, ...])	
create_summary_df(data)	This function takes the xarray produced by
save_pdfs(args)	
main(data_dir, num, out_summary_df_path, ...)	A function which performs all the above steps

ggt.modules.result_aggregator.**calculate_cdf**(*data*, *n=1000*, *warnings=True*)

ggt.modules.result_aggregator.**find_idx_nearest_val**(*array*, *value*)

ggt.modules.result_aggregator.**calculate_ci**(*data*, *n=1000*, *warnings=False*)

ggt.modules.result_aggregator.**expit_custom**(*x_input*, *scaling_array=None*)
Performs a custom implementation of Scipy's expit function. If the scaling array is supplied, it does the unscaling keeping in mind how 0s and 1s were mapped to different values while the scaling was done. expit is the inverse of the logit function.

ggt.modules.result_aggregator.**unscale_preds**(*df_input*, *scaling_df_path=None*, *drop_old=True*)

ggt.modules.result_aggregator.**file_loader**(*args*)

ggt.modules.result_aggregator.**bayesian_inference_file_gobbler**(*data_dir*, *num=300*, *unscale=False*, *scaling_df_path=None*, *drop_old=True*)

ggt.modules.result_aggregator.**create_summary_df**(*data*)
This function takes the xarray produced by bayesian_inference_file_gobbler and then produces a dataframe with summary statistics.

ggt.modules.result_aggregator.**save_pdfs**(*args*)

ggt.modules.result_aggregator.**main**(*data_dir*, *num*, *out_summary_df_path*, *out_pdfs_path*, *unscale*, *scaling_df_path*, *drop_old*)
A function which performs all the above steps necessary to prepeare the data for analysis

Package Contents

Functions

predict(model_path, dataset, cutout_size, channels[, ...])	Using the model defined in model path, return the output values for

ggt.modules.**predict**(*model_path*, *dataset*, *cutout_size*, *channels*, *parallel=False*, *batch_size=256*, *n_workers=1*, *model_type='ggt'*, *n_out=1*, *mc_dropout=False*, *dropout_rate=None*)
Using the model defined in model path, return the output values for the given set of images

ggt.train

Submodules

ggt.train.create_trainer

Module Contents

Functions

create_trainer(model, optimizer, criterion, loaders, ...)	Set up Ignite trainer and evaluator.

ggt.train.create_trainer.**create_trainer**(*model*, *optimizer*, *criterion*, *loaders*, *device*)
Set up Ignite trainer and evaluator.

ggt.train.train

Module Contents

Functions

train(**kwargs)	Runs the training procedure using MLFlow.

Attributes

log_fmt

ggt.train.train.**train**(***kwargs*)
Runs the training procedure using MLFlow.

ggt.train.train.**log_fmt = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'**

Package Contents

Functions

create_trainer(model, optimizer, criterion, loaders, ...)	Set up Ignite trainer and evaluator.

ggt.train.**create_trainer**(*model*, *optimizer*, *criterion*, *loaders*, *device*)
Set up Ignite trainer and evaluator.

ggt.utils

Submodules

ggt.utils.data_utils

Module Contents

Functions

load_cat(data_dir, slug, split)	Loads and returns pandas dataframe

ggt.utils.data_utils.**load_cat**(*data_dir*, *slug*, *split*)
Loads and returns pandas dataframe

ggt.utils.device_utils

Module Contents

Functions

discover_devices()	Check for available devices.

ggt.utils.device_utils.**discover_devices**()
Check for available devices.

ggt.utils.model_utils

Module Contents

Functions

get_output_shape(model, image_dim)	Get output shape of a PyTorch model or layer
enable_dropout(model)	Enable random dropout during inference. From StackOverflow #63397197
specify_dropout_rate(model, rate)	Specify the dropout rate of all layers

ggt.utils.model_utils.**get_output_shape**(*model*, *image_dim*)
Get output shape of a PyTorch model or layer

ggt.utils.model_utils.**enable_dropout**(*model*)
Enable random dropout during inference. From StackOverflow #63397197

ggt.utils.model_utils.**specify_dropout_rate**(*model*, *rate*)
Specify the dropout rate of all layers

ggt.utils.tensor_utils

Module Contents

Functions

tensor_to_numpy(x)	Convert a torch tensor to NumPy for plotting.
arsinh_normalize(X)	Normalize a Torch tensor with arsinh.
load_tensor(filename, tensors_path[, as_numpy])	Load a Torch tensor from disk.
standardize_labels(input, data_dir, split, slug, ...)	Standardizes data. During training, input should
metric_output_transform_al_loss(output)	Transforms the output of the model, when using
metric_output_transform_al_cov_loss(output)	Transforms the output of the model, when using

ggt.utils.tensor_utils.**tensor_to_numpy**(*x*)
Convert a torch tensor to NumPy for plotting.

ggt.utils.tensor_utils.**arsinh_normalize**(*X*)
Normalize a Torch tensor with arsinh.

ggt.utils.tensor_utils.**load_tensor**(*filename*, *tensors_path*, *as_numpy=True*)
Load a Torch tensor from disk.

ggt.utils.tensor_utils.**standardize_labels**(*input*, *data_dir*, *split*, *slug*, *label_col*, *scaling*, *invert=False*)
Standardizes data. During training, input should be the labels, and during inference, input should be the predictions.

ggt.utils.tensor_utils.**metric_output_transform_al_loss**(*output*)
Transforms the output of the model, when using aleatoric loss, to a form which can be used by the ignote metric calculators

ggt.utils.tensor_utils.**metric_output_transform_al_cov_loss**(*output*)
Transforms the output of the model, when using aleatoric covariance loss, to a form which can be used by the ignote metric calculators

Package Contents

Functions

discover_devices()	Check for available devices.
tensor_to_numpy(x)	Convert a torch tensor to NumPy for plotting.
arsinh_normalize(X)	Normalize a Torch tensor with arsinh.
load_tensor(filename, tensors_path[, as_numpy])	Load a Torch tensor from disk.
standardize_labels(input, data_dir, split, slug, ...)	Standardizes data. During training, input should
metric_output_transform_al_loss(output)	Transforms the output of the model, when using
metric_output_transform_al_cov_loss(output)	Transforms the output of the model, when using
load_cat(data_dir, slug, split)	Loads and returns pandas dataframe
get_output_shape(model, image_dim)	Get output shape of a PyTorch model or layer
enable_dropout(model)	Enable random dropout during inference. From StackOverflow #63397197
specify_dropout_rate(model, rate)	Specify the dropout rate of all layers

ggt.utils.**discover_devices**()
Check for available devices.

ggt.utils.**tensor_to_numpy**(*x*)
Convert a torch tensor to NumPy for plotting.

ggt.utils.**arsinh_normalize**(*X*)
Normalize a Torch tensor with arsinh.

ggt.utils.**load_tensor**(*filename*, *tensors_path*, *as_numpy=True*)
Load a Torch tensor from disk.

ggt.utils.**standardize_labels**(*input*, *data_dir*, *split*, *slug*, *label_col*, *scaling*, *invert=False*)
Standardizes data. During training, input should be the labels, and during inference, input should be the predictions.

ggt.utils.**metric_output_transform_al_loss**(*output*)
Transforms the output of the model, when using aleatoric loss, to a form which can be used by the ignote metric calculators

ggt.utils.**metric_output_transform_al_cov_loss**(*output*)
Transforms the output of the model, when using aleatoric covariance loss, to a form which can be used by the ignote metric calculators

ggt.utils.**load_cat**(*data_dir*, *slug*, *split*)
Loads and returns pandas dataframe

ggt.utils.**get_output_shape**(*model*, *image_dim*)
Get output shape of a PyTorch model or layer

ggt.utils.**enable_dropout**(*model*)
Enable random dropout during inference. From StackOverflow #63397197

ggt.utils.**specify_dropout_rate**(*model*, *rate*)
Specify the dropout rate of all layers

ggt.visualization

Submodules

ggt.visualization.spatial_transform

Module Contents

Functions

visualize_spatial_transform(model, loader, output_dir)	
main(model_type, model_path, cutout_size, channels, ...)	Visualize the transformation performed by the spatial transformer

Attributes

log_fmt

ggt.visualization.spatial_transform.**visualize_spatial_transform**(*model*, *loader*, *output_dir*, *device='cpu'*, *nrow=6*, *return_grids=False*, *matplotlib_backend='agg'*)

ggt.visualization.spatial_transform.**main**(*model_type*, *model_path*, *cutout_size*, *channels*, *n_out*, *data_dir*, *split_slug*, *split*, *batch_size*, *nrow*, *n_workers*, *normalize*)

Visualize the transformation performed by the spatial transformer module.

ggt.visualization.spatial_transform.**log_fmt = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'**

7.2 test_metrics

7.2.1 Module Contents

Functions

test_elementwise_mae()

test_metrics.**test_elementwise_mae**()

7.3 test_utils

7.3.1 Module Contents

Functions

test_specify_dropout_rate()

test_utils.**test_specify_dropout_rate**()

7.4 test_models

7.4.1 Module Contents

Functions

test_ggt()

test_ggt_no_gconv()

test_vgg16()

test_vgg16_pretrained()

test_models.**test_ggt**()

test_models.**test_ggt_no_gconv**()

test_models.**test_vgg16**()

test_models.**test_vgg16_pretrained**()

7.5 test_install

7.5.1 Module Contents

Functions

test_install()

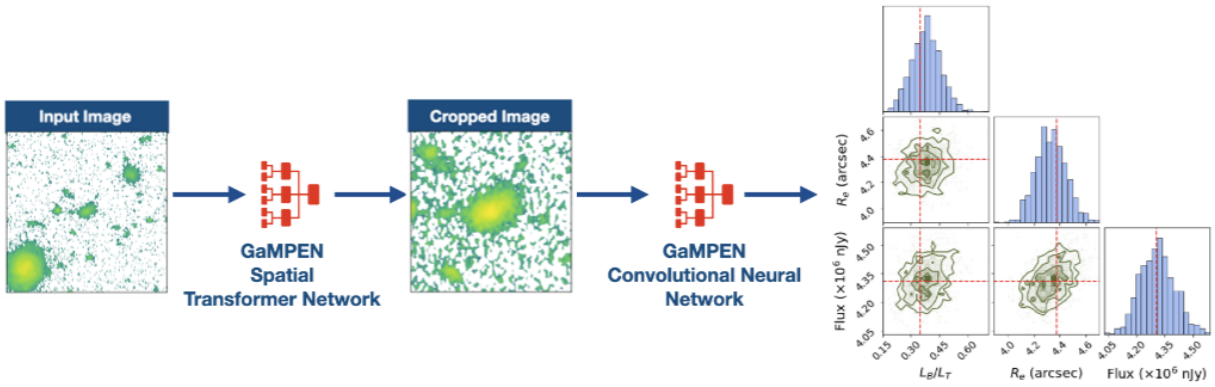
test_install.**test_install**()

Attention: Note that although GaMPEN's current documentation is fairly substantative, we are still working on some parts of the documentation and some Tutorials. If you run into issues while trying to use GaMPEN, please contact us! We will be more than happy to help you!

CHAPTER

EIGHT

ABOUT GAMPEN



GALAXY MORPHOLOGY POSTERIOR ESTIMATION NETWORK

The first ML framework for automatic cropping & estimating posteriors of different galaxy morphological parameters

The Galaxy Morphology Posterior Estimation Network (GaMPEN) is a novel machine learning framework that estimates Bayesian posteriors (i.e., values + uncertainties) of morphological parameters for arbitrarily large numbers of galaxies.

As the above image shows, GaMPEN uses a Spatial Transformer Network (STN) to first automatically crop input images to an optimal size and then uses a Convolutional Neural Network (CNN) to predict joint posterior distributions of user-specified structural/morphological parameters.

8.1 First Steps with GaMPEN

1. Follow the installation instructions and quick-start guide in *Getting Started*.
2. Go through the *Tutorials* to learn how to use GaMPEN for a variety of different tasks.
3. Review the *Using GaMPEN* page to dive into the details about the various user-facing functions that GaMPEN provides.

For a quick blog-esque introduction to the most important features of GaMPEN, please check out this page. For a deep-dive, please refer to Ghosh et. al. 2022

Note that if you want to access the publicly released trained models + morphological parameters for specific surveys (e.g., Hyper Suprime-Cam), please refer to the *Public Data Release Handbook* page.

8.2 What Parameters and Surveys can GaMPEN be Used for?

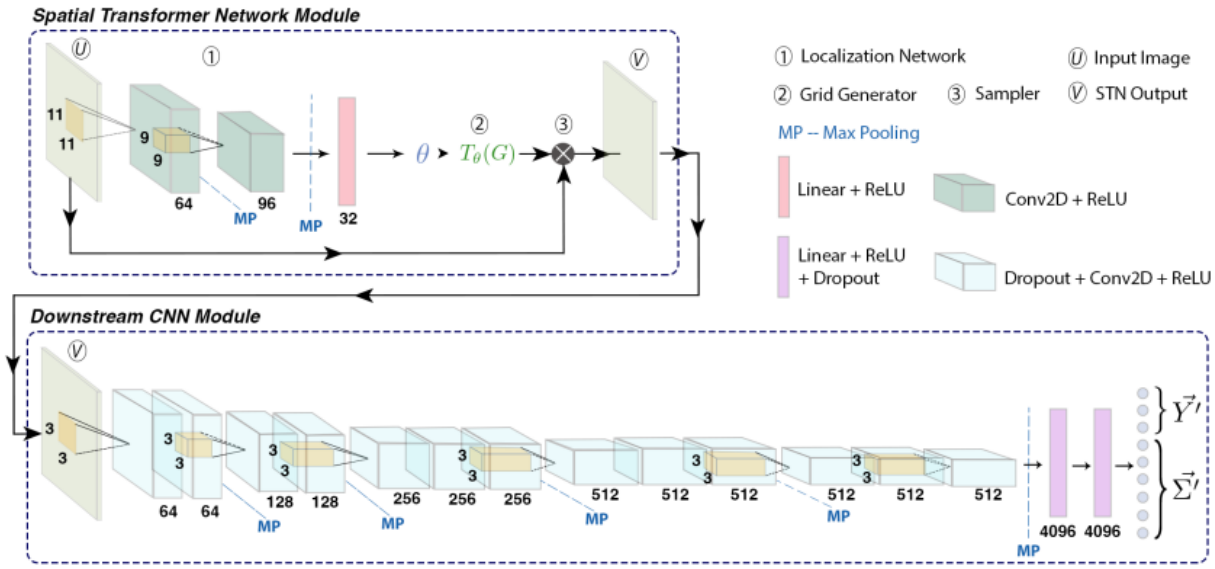
The publicly released GaMPEN models are turned to predict specific parameters for specific surveys. For example, our Hyper Suprime-Cam (HSC) models can be used to estimate the bulge-to-total light ratio, effective radius, and flux of HSC galaxies till $z < 0.75$.

However, GaMPEN models can be trained from scratch to determine any combination of morphological parameters (even different from the ones mentioned above – e.g. Sersic Index) **for any space or ground-based imaging survey**. Please check out our FAQs page for our recommendations if you want to train a GaMPEN model tuned to a specific survey. Also, don't hesitate to contact us if you want our help/advice in training a GaMPEN model for your survey/parameters.

8.3 More Details About GaMPEN

8.3.1 GaMPEN's Architecture

GaMPEN consists of a two sequential neural network modules – a Spatial Transformer Network (STN) and a Convolutional Neural Network (CNN). The image below shows the detailed architecture of both these networks. Note that both the networks are trained simulataneously using the same loss function and optimizer. For further details about the architecture, please refer to Ghosh et. al. 2022 or the `vgg16_w_stn_oc_drp.py` file in the `GaMPEN/ggt/models/` directory.



8.3.2 GaMPEN's Posteriors/Uncertainties

To predict posteriors, GaMMPEN takes into account both aleatoric and epistemic uncertainties. It uses the negative log-likelihood of the output parameters as the loss function combined with the Monte Carlo Dropout technique. GaMPEN also incorporates the full covariance matrix in the loss function, using a series of algebraic manipulations.

The uncertainties/posteriors produced by GaMPEN have been shown to be extremely well-calibrated ($\lesssim 5\%$ deviation. As shown in Ghosh et. al. 2022b this represents a significant improvement over state-of-the-art light profile fitting tools which underestimate uncertainties by $\sim 15\% - 60\%$ depending on the brightness of the source.

8.3.3 Predictional Stabiltily Against Rotational Transformations

The video below shows the stability of predictions made by trained GaMPEN HSC models when an input galaxy image is rotated through various angles. As can be seen, GaMPEN's predictions of all three output parameters are fairly stable against rotations.

8.4 Publications

GaMPEN was initially introduced in 2022 in this ApJ paper.

Since then, GaMPEN has been used in a number of other publications. We always try to maintain an updated record of GaMPEN's trained models and catalogs produced on this page

8.5 Attribution Info.

Please cite the below mentioned publication if you make use of GaMPEN or some code herein.

```
  @article{Ghosh2022,
  author = {Aritra Ghosh and C. Megan Urry and Amrit Rau and Laurence Perreault-
↪Levasseur and Miles Cranmer and Kevin Schawinski and Dominic Stark and Chuan Tian and␣
↪Ryan Ofman and Tonima Tasnim Ananna and Connor Auge and Nico Cappelluti and David B.␣
↪Sanders and Ezequiel Treister},
  doi = {10.3847/1538-4357/ac7f9e},
  issn = {0004-637X},
  issue = {2},
  journal = {The Astrophysical Journal},
  month = {8},
  pages = {138},
  title = {GaMPEN: A Machine-learning Framework for Estimating Bayesian Posteriors of␣
↪Galaxy Morphological Parameters},
  volume = {935},
  year = {2022},
  }
```

Additionally, if you are using publicly released GaMPEN models or catalogs for a specific survey, please cite the relevant publication(s) in which the data was released. For example, if you are using the GaMPEN HSC models, please cite this article.

8.6 License

8.7 Getting Help/Contributing

We always welcome contributions to GaMPEN! If you have any questions about using GaMPEN, please feel free to send me an e-mail at this `aritraghsh09@xxxxx.com` GMail address.

If you have spotted a bug in the code/documentation or you want to propose a new feature, please feel free to open an issue/a pull request on GitHub.

PYTHON MODULE INDEX

g

t

INDEX

Symbols

A

B

C

D

E

F

G

I

L

M

P

R

S

T

U

V