gamornet

Release 0.4.3

Jun 30, 2022

Contents

Note: **20th Nov. 2020**: Yale Astronomy's public FTP server is back online and all services should work as usual. If you are still having issues while using our trained models or trying out the tutorials, please let us know.

A tarball of all the public data products is now also available via Google Drive.

The Galaxy Morphology Network (GaMorNet) is a convolutional neural network that can classify galaxies as being disk-dominated, bulge-dominated or indeterminate based on their bulge to total light ratio. GaMorNet doesn't need a large amount of training data and can work across different data-sets. For more details about GaMorNet's design, how it was trained etc., please refer to *Publication & Other Data*.

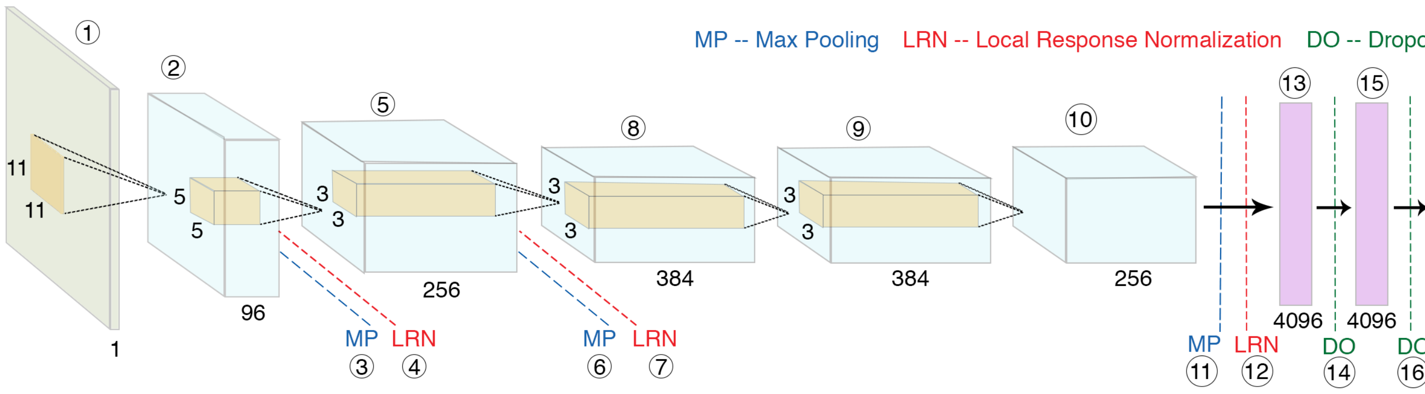

Fig. 1: Schematic diagram of Galaxy Morphology Network.

CHAPTER 1

First contact with GaMorNet

GaMorNet's user-faced functions have been written in a way so that it's easy to start using them even if you have not dealt with convolutional neural networks before. For. eg. to perform predictions on an array of SDSS images using our trained models, the following line of code is all you need.

```
from gamornet.keras_module import gamornet_predict_keras

preds = gamornet_predict_keras(img_array, model_load_path='SDSS_tl', input_shape='SDSS
↪')
```

In order to start using GaMorNet, please first look at the *Getting Started* section for instructions on how to install GaMorNet. Thereafter, we recommend trying out the *Tutorials* in order to get a handle on how to use GaMorNet.

Finally, you should have a look at the *Public Data Release Handbook* for our recommendations on how to use different elements of GaMorNet's public data release for your own work and the *API Documentation* for detailed documentation of the different functions in the module.

CHAPTER 2

Publication & Other Data

You can look at this ApJ paper to learn the details about GaMorNet's architecture, how it was trained, and other details not mentioned in this documentation.

We strongly suggest you read the above-mentioned publication if you are going to use our trained models for performing predictions or as the starting point for training your own models.

All the different elements of the public data release (including the new Keras models) are summarized in *Public Data Release Handbook*.

2.1 Attribution Info.

Please cite the above mentioned publication if you make use of this software module or some code herein.

```
@article{Ghosh2020,
  doi = {10.3847/1538-4357/ab8a47},
  url = {https://doi.org/10.3847/1538-4357/ab8a47},
  year = {2020},
  month = jun,
  publisher = {American Astronomical Society},
  volume = {895},
  number = {2},
  pages = {112},
  author = {Aritra Ghosh and C. Megan Urry and Zhengdong Wang and Kevin Schawinski␣
↪and Dennis Turp and Meredith C. Powell},
  title = {Galaxy Morphology Network: A Convolutional Neural Network Used to Study␣
↪Morphology and Quenching in $\sim$100, 000 {SDSS} and $\sim$20, 000 {CANDELS}␣
↪Galaxies},
  journal = {The Astrophysical Journal}
}
```

Additionally, if you want, please include the following text in the Software/Acknowledgment section.

```
This work uses trained models/software made available as a part of the Galaxy␣
↪Morphology Network public data release.
```

2.2 License

CHAPTER 3

Getting Help/Contributing

If you have a question, please first have a look at the *FAQs* section. If your question is not answered there, please send me an e-mail at this `aritraghsh09+gamornet@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

3.1 Getting Started

GaMorNet is written in Python and uses the Keras and TFLearn deep learning libraries to perform all of the machine learning operations. Both these aforementioned libraries in turn use TensorFlow for their underlying tensor operations. GaMorNet was originally written using TFLearn, but the Keras module was added later as we expect Keras to be better supported and developed going forward.

GaMorNet has two separate packages available via pip. One happens to be the standard `gamornet` package and the other one is a `gamornet-cpu` package meant for users who don't have access to a GPU.

3.1.1 Ways to Use GaMorNet

1. If you have access to a GPU,
   - We recommend installing the `gamornet` package using the instructions in *Installation*
   - However, if you are not familiar with how to enable GPU support for TensorFlow and want to get started quickly, you may consider using Google Colab like we have done in the *Tutorials*
2. If you don't have access to a GPU,
   - and want to use our models for predictions
     - You can install the `gamornet-cpu` package using the instructions in *Installation*
     - You can use Google Colab like we have done in the *Tutorials*
   - and want to train your own models

  - Use the GPUs available via Google Colab as we have done in the *Tutorials*

3.1.2 Installation

It is highly recommended to have a separate Python virtual environment for running GaMorNet as the package has many specific version oriented dependencies on other Python packages. The following instructions are shown using Anaconda, but feel free to go ahead and use any other virtual environment tool you are comfortable using. **Note that GaMorNet only runs on Python >= 3.3 and is recommended to be run on Python 3.6**

1. Using pip
   - Install Anaconda if you don't have it already using the instructions here
   - Create a new Anaconda environment using `conda create -n gamornetenv python=3.6`
   - Activate the above environment using `conda activate gamornetenv`
   - Install GaMorNet using `pip install gamornet` or `pip install gamornet-cpu` depending on your requirements
   - For the GPU installation, if you don't have the proper CUDA libraries, please see *GPU Support*
   - To test the installation, open up a Python shell and type `from gamornet.keras_module import *`. If this doesn't raise any errors, it means you have installed GaMorNet successfully.
   - To exit the virtual environment, type `conda deactivate`
2. From Source
   - Install Anaconda if you don't have it already using the instructions here
   - Create a new Anaconda environment using `conda create -n gamornetenv python=3.6`
   - Activate the above environment using `conda activate gamornetenv`
   - Clone GaMorNet repository from GitHub using `git clone https://github.com/aritraghsh09/GaMorNet.git`
   - To install, do the following based on the package you want
     - For GPU installation,
       * `cd GaMorNet`
       * `python setup.py install`
     - For CPU version,
       * `cd GaMorNet`
       * `git fetch --all`
       * `git checkout cpu_version`
       * `python setup.py install`
   - For the GPU installation, if you don't have the proper CUDA libraries, please see *GPU Support*
   - To test the installation, open up a Python shell and type `from gamornet.keras_module import *`. If this doesn't raise any errors, it means you have installed GaMorNet successfully.
   - To exit the virtual environment, type `conda deactivate`

3.1.3 GPU Support

If you are using a GPU, then you would need to make sure that the appropriate CUDA and cuDNN versions are installed. The appropriate version is decided by the versions of your installed Python libraries. For detailed instructions on how to enable GPU support for Tensorflow, please see this link.

We tested GaMorNet using the following configurations:-

Python	Keras	TFLearn	Tensorflow	CUDA	cuDNN
3.6.10	2.2.4	0.3.2	1.13.1	10.0.130	7.6.0
3.6.10	2.3.1	0.3.2	1.15.3	10.0.130	7.6.2

For more build configurations tested out by the folks at TensorFlow, please see this link

3.2 Tutorials

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

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.

Each Notebook has separate sections on using the Keras and TFLearn modules.

3.2.1 Making Predictions

This tutorial demonstrates how you can use GaMorNet models to make predictions using two images from our SDSS dataset.

3.2.2 Training GaMorNet

This tutorial uses simulated SDSS galaxies to train a GaMorNet model from scratch.

3.2.3 Transfer Learning with GaMorNet

This tutorial uses real SDSS galaxies to perform transfer learning on a GaMorNet model trained only on simulations.

3.3 Public Data Release Handbook

Note: A tarball of all the public data products is now also available via Google Drive.

The folder structure is the same as that of the FTP server mentioned below.

If you are looking for information about the various ways you can use GaMorNet (running on a CPU v/s GPU v/s the cloud) or installation instructions, please have a look at *Getting Started*. This section summarizes different aspects of the public data release and provides some advice on the applicability of GaMorNet for various tasks.

3.3.1 Usage Advice

How you will use the public data release of GaMorNet strongly depends on the task at hand.

- If you are looking for predictions of the SDSS g-band and CANDELS H-band dataset of Ghosh et. al. (2020), please have a look at the *Prediction Tables* section.
- If you have SDSS g-band ($z \sim 0$) and/or CANDELS H-band ($z \sim 1$) data that we haven't classified, please use the final trained models (on simulations + real data) that we have released. You can manually download these models from *Trained Models* or use the `gamornet_predict_keras()` / `gamornet_predict_tflearn()` functions as shown in *Tutorials* and *API Documentation*.
- If you have SDSS and CANDELS data other than g-band at $z \sim 0$ and H-band at $z \sim 1$ that you want to classify:-
  - If the data are in nearby bands *at the same redshifts* (i.e. near g-band for SDSS and H-band for CANDELS), we recommend using the `gamornet_tl_keras()` / `gamornet_tl_tflearn()` functions as shown in *Tutorials* and *API Documentation* to perform transfer learning. We recommend starting the transfer learning process from both our simulation-only and final trained models and choosing the one that maximizes the accuracy on your validation set. In case you want to download the models manually, see *Trained Models*.
  - If you believe that your data is significantly different in redshift, resolution or any other photometric aspect, you could also train a network from scratch using `gamornet_train_keras()` / `gamornet_train_tflearn()` as shown in *Tutorials* and *API Documentation*.
- If you have some other data that you want to classify, train a network from scratch using `gamornet_train_keras()` / `gamornet_train_tflearn()` as shown in *Tutorials* and *API Documentation*.

If you are not sure about something, please look at this documentation carefully and contact us using the information available at *Getting Help/Contributing*.

Important: GaMorNet is best utilized when you a large number of images to analyze. If you only have a handful of images (~ 5) that you want to look at in greater detail, your purposes in all probability will be served better by a standalone light profile fitting code.

3.3.2 Summary of Public Data Release

This section summarizes the different aspects of the data-products released with GaMorNet and how to use them.

Keras v/s TFLearn

Note that all the work in Ghosh et. al. (2020) was originally done using TFLearn. We later used Keras to reproduce the same work. Thus, everything in the Public Data Release is available in two flavors – Keras and TFLearn.

Important: Note that due to the inherent stochasticity involved in training a neural network, the results given by the Keras and TFLearn models are very close, but not exact replicas of one another. If you want to re-create the results in Ghosh et. al. (2020), you should use the TFLearn flavored data products. In all other cases, we recommend using the Keras flavored data products as it will be better supported in the future. Look below to understand how the two flavors are different.

Warning: Note that for the Keras models, the accuracies achieved are slightly different than what was achieved with TFLearn in Ghosh et. al. (2020). Additionally, the recommended probability thresholds are also different. Please read the information below before using the Keras models.

Accuracies

The accuracies achieved with the both the Keras & TFLearn models for the sample of Ghosh et. al. (2020) are shown below. These tables are similar in information content to Tables 5 and 7 in Ghosh et. al. (2020), which were obtained using TFLearn.

Keras on SDSS	Predicted Disks	Predicted Bulges
Actual Disks	99.72%	3.37%
Actual Bulges	0.15%	95.25%

Keras on CANDELS	Predicted Disks	Predicted Bulges
Actual Disks	94.45%	21.74%
Actual Bulges	5.37%	77.88%

TFLearn on SDSS	Predicted Disks	Predicted Bulges
Actual Disks	99.72%	4.13%
Actual Bulges	0.19%	94.83%

TFLearn on CANDELS	Predicted Disks	Predicted Bulges
Actual Disks	91.83%	20.86%
Actual Bulges	7.90%	78.62%

Important: For an additional consistency-check, we counted how many of the galaxies switched classifications between disk-dominated and bulge-dominated, when predictions were performed separately using the Keras and TFLearn models. For both the SDSS and CANDELS samples, this number is $\leq 0.04\%$

Indeterminate Fraction

The table below shows the number of galaxies in the Ghosh et. al. (2020) sample that are classified by the various models of GaMorNet to be indeterminate. This includes galaxies which have intermediate bulge-to-total light ratios ($0.45 \leq L_B/L_T \leq 0.55$) and those for which the network is not confident enough to make a prediction. For more information, please refer to Section 4 of the paper.

	Keras SDSS	Keras CANDELS	TFLearn SDSS	TFLearn CANDELS
Indeterminate Galaxies	31%	46%	33%	39%

Thresholds Used

To turn GaMorNet's output probability values into class predictions, we use probability thresholds. The probability thresholds that were used to generate the prediction tables as well as the tables above are shown below.

Keras on SDSS

1. Disk-dominated if disk-probability $\geq 70\%$
2. Bulge-dominated if bulge-probability $\geq 70\%$
3. Indeterminate otherwise

Keras on CANDELS

1. Disk-dominated if disk-probability > bulge and indeterminate probability
2. Bulge-dominated if bulge-probability $\geq 60\%$
3. Indeterminate otherwise

TFLearn on SDSS

1. Disk-dominated if disk-probability $\geq 80\%$
2. Bulge-dominated if bulge-probability $\geq 80\%$
3. Indeterminate otherwise

TFLearn on CANDELS

1. Disk-dominated if disk-probability > bulge and indeterminate probability and 36%
2. Bulge-dominated if bulge-probability $\geq 55\%$
3. Indeterminate otherwise

Important: The choice of the confidence/probability threshold is arbitrary and should be chosen appropriately for the particular task at hand. Towards this end, Figures 8 and 9 of Ghosh et. al. (2020) can be used to asses the trade-off between accuracy and completeness for both samples.

For more information about the impact of probability thresholds on the results, please refer to Section 4.1 of the paper

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.

Using Linux Command Line

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

If prompted for a username, try `anonymous` and keep the password field blank.

Using a Browser

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

Using Finder on OSX

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

Thereafter, navigate to the appropriate subdirectory.

Google Drive

A tarball of all the data products on the public *FTP Server* is now also available via Google Drive

Prediction Tables

The predicted probabilities (of being disk-dominated, bulge-dominated, or indeterminate) and the final classifications for all of the galaxies in the SDSS and CANDELS test sets of Ghosh et. al. (2020) are made available as `.txt` files. These tables are the full versions of Tables 4 and 6 in the paper. The appropriate sub-directories of the *FTP Server* are mentioned below:-

TFLearn

- SDSS dataset predictions ⇒ */gamornet/pred_tables/pred_table_sdss.txt*
- CANDELS dataset predictions ⇒ */gamornet/pred_tables/pred_table_candels.txt*

Keras

- SDSS dataset predictions ⇒ */gamornet_keras/pred_tables/pred_table_sdss.txt*
- CANDELS dataset predictions ⇒ */gamornet_keras/pred_tables/pred_table_candels.txt*

Trained Models

Note that the functions `gamornet_predict_keras()`, `gamornet_predict_tflearn()` automatically download and use the trained models when the correct parameters are passed to them. However, in case you want to just download the model files for yourself, navigate to the appropriate sub-directories on the *FTP Server* as mentioned below. For more information about these models, please refer to Ghosh et. al. (2020) and see *Usage Advice*.

TFLearn

- SDSS model trained only on simulations ⇒ */gamornet/trained_models/SDSS/sim_trained/*
- SDSS model trained on simulations and real data ⇒ */gamornet/trained_models/SDSS/tl/*
- CANDELS model trained only on simulations ⇒ */gamornet/trained_models/CANDELS/sim_trained/*
- CANDELS model trained on simulations and real data ⇒ */gamornet/trained_models/CANDELS/tl/*

Keras

- SDSS model trained only on simulations ⇒ */gamornet_keras/trained_models/SDSS/sim_trained/*
- SDSS model trained on simulations and real data ⇒ */gamornet_keras/trained_models/SDSS/tl/*
- CANDELS model trained only on simulations ⇒ */gamornet_keras/trained_models/CANDELS/sim_trained/*
- CANDELS model trained on simulations and real data ⇒ */gamornet_keras/trained_models/CANDELS/tl/*

3.4 API Documentation

Both the Keras and TFLearn modules have similarly named functions with very similar parameters. Use the `_predict_` functions to perform predictions using our trained models or a model you trained from scratch. Use the `_train_` functions to train a model from scratch. Use the `_tl_` functions to perform transfer learning on a previously trained model – this can be our pre-trained models or a model that you trained.

Please have a look at the *Tutorials* for examples of how to use these functions effectively.

3.4.1 Keras Module

The three major user oriented functions happen to be *`gamornet_predict_keras()`*, *`gamornet_train_keras()`* and *`gamornet_tl_keras()`* and are documented here. For the remainder of the functions, please have a look at the source code on GitHub.

`gamornet.keras_module.`**`gamornet_predict_keras`**(*img_array*, *model_load_path*, *input_shape*, *batch_size=64*, *individual_arrays=False*)

Uses a Keras model to perform predictions on supplied images.

Parameters

- **`img_array`** (*`Numpy ndarray [nsamples,x,y,ndim]`*) – The array of images on which the predictions are to be performed. We insist on numpy arrays as many of the underlying deep learning frameworks work better with numpy arrays compared to other array-like elements.
- **`model_load_path`** (*`str`*) – Full path to the saved Keras model.

  Additionally, this parameter can take the following special values

  – `SDSS_sim` – Downloads and uses the GaMorNet model trained on SDSS g-band simulations at z~0 from Ghosh et. al. (2020)

  – `SDSS_tl` – Downloads and uses the GaMorNet model trained on SDSS g-band simulations and real data at z~0 from Ghosh et. al. (2020)

  – `CANDELS_sim` – Downloads and uses the GaMorNet model trained on CANDELS H-band simulations at z~1 from Ghosh et. al. (2020)

  – `CANDELS_tl` – Downloads and uses the GaMorNet model trained on CANDELS H-band simulations and real data at z~1 from Ghosh et. al. (2020)
- **`input_shape`** (*`tuple of ints (x, y, ndim) or allowed str`*) – The shape of the images being used in the form of a tuple.

  This parameter can also take the following special values:-

  – `SDSS` - Sets the input shape to be (167,167,1) as was used for the SDSS g-band images in Ghosh et. al. (2020)

  – `CANDELS` - Sets the input shape to be (83,83,1) as was used for the CANDELS H-band images in Ghosh et. al. (2020)
- **`batch_size`** (*`int`*) – This variable specifies how many images will be processed in a single batch. Set this value to lower than the default if you have limited memory availability. This doesn't affect the predictions in any way.
- **`individual_arrays`** (*`bool`*) – If set to True, this will unpack the three returned arrays

Returns

predicted probabilities – The returned array consists of the probability for each galaxy to be disk-dominated, indeterminate and bulge-dominated respectively [disk_prob, indet_prob, bulge_prob]. If `individual_arrays` is set to `True`, the single array is unpacked and returned as three separate arrays in the same order.

The ordering of individual elements in this array corresponds to the array of images fed in.

Return type array_like

`gamornet.keras_module.`**`gamornet_train_keras`**(*training_imgs*, *training_labels*, *validation_imgs*, *validation_labels*, *input_shape*, *files_save_path='./'*, *epochs=100*, *checkpoint_freq=0*, *batch_size=64*, *lr=0.0001*, *momentum=0.9*, *decay=0.0*, *nesterov=False*, *loss='categorical_crossentropy'*, *load_model=False*, *model_load_path='./'*, *save_model=True*, *verbose=1*)

Trains and returns a GaMorNet model using Keras.

Parameters

- **`training_imgs`** (*`Numpy ndarray [nsamples,x,y,ndim]`*) – The array of images which are to be used for the training process. We insist on numpy arrays as many of the underlying deep learning frameworks work better with numpy arrays compared to other array-like elements.
- **`training_labels`** (*`Numpy ndarray [nsamples,label_arrays]`*) – The truth labels for each of the training images. The supplied labels must be in the one-hot encoding format. We reproduce below what each individual label array should look like:-
  - Disk-dominated - `[1,0,0]`
  - Indeterminate - `[0,1,0]`
  - Bulge-dominated - `[0,0,1]`
- **`validation_imgs`** (*`Numpy ndarray [nsamples,x,y,ndim]`*) – The array of images which are to be used for the validation process. We insist on numpy arrays as many of the underlying deep learning frameworks work better with numpy arrays compared to other array-like elements.
- **`validation_labels`** (*`Numpy ndarray [nsamples,label_arrays]`*) – The truth labels for each of the validation images. The supplied labels must be in the one-hot encoding format. We reproduce below what each individual label array should look like:-
  - Disk-dominated - `[1,0,0]`
  - Indeterminate - `[0,1,0]`
  - Bulge-dominated - `[0,0,1]`
- **`input_shape`** (*`tuple of ints (x, y, ndim) or allowed str`*) – The shape of the images being used in the form of a tuple.

  This parameter can also take the following special values:-
  - `SDSS` - Sets the input shape to be (167,167,1) as was used for the SDSS g-band images in Ghosh et. al. (2020)
  - `CANDELS` - Sets the input shape to be (83,83,1) as was used for the CANDELS H-band images in Ghosh et. al. (2020)

- **files_save_path** (*str*) – The full path to the location where files generated during the training process are to be saved. This includes the `metrics.csv` file as well as the trained model.

  Set this to `/dev/null` on a unix system if you don't want to save the output.
- **epochs** (*int*) – The number of epochs for which you want to train the model.
- **checkpoint_freq** (*int*) – The frequency (in terms of epochs) at which you want to save models. For eg. setting this to 25, would save the model at its present state every 25 epochs.
- **batch_size** (*int*) – This variable specifies how many images will be processed in a single batch. This is a hyperparameter. The default value is a good starting point
- **lr** (*float or schedule*) – 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.

  Instead of setting it at a single value, you can also set a schedule using `keras.optimizers.schedules.LearningRateSchedule`
- **momentum** (*float*) – The value of the momentum to be used in the gradient descent optimizer that is used to train GaMorNet. This must always be ≥ 0. This accelerates the gradient descent process. This is a hyperparameter. The default value is a good starting point.
- **decay** (*float*) – The amount of learning rate decay to be applied over each update.
- **nesterov** (*bool*) – Whether to apply Nesterov momentum or not.
- **loss** (*allowed str or function*) – The loss function to be used. If using the string option, you need to specify the name of the loss function. This can be set to be any loss available in `keras.losses`
- **load_model** (*bool*) – Whether you want to start the training from a previously saved model.

  We strongly recommend using the `gamornet_tl_keras` function for more control over the process when starting the training from a previously saved model.
- **model_load_path** (*str*) – Required iff `load_model == True`. The path to the saved model.
- **save_model** (*bool*) – Whether you want to save the model in its final trained state.

  Note that this parameter does not affect the models saved by the `checkpoint_freq` parameter
- **verbose** (*{0, 1, 2}*) – The level of verbosity you want from Keras during the training process. 0 = silent, 1 = progress bar, 2 = one line per epoch.

Returns **Trained Keras Model**

Return type Keras `Model` class

`gamornet.keras_module.`**`gamornet_tl_keras`**`(`*training_imgs, training_labels, validation_imgs, validation_labels, input_shape, load_layers_bools=[True, True, True, True, True, True, True, True], trainable_bools=[True, True, True, True, True, True, True, True], model_load_path='./', files_save_path='./', epochs=100, checkpoint_freq=0, batch_size=64, lr=1e-05, momentum=0.9, decay=0.0, nesterov=False, loss='categorical_crossentropy', save_model=True, verbose=1*`)`

Performs Transfer Learning (TL) using a previously trained GaMorNet model.

Parameters

- **`training_imgs`** (*`Numpy ndarray [nsamples,x,y,ndim]`*) – The array of images which are to be used for the TL process. We insist on numpy arrays as many of the underlying deep learning frameworks work better with numpy arrays compared to other array-like elements.
- **`training_labels`** (*`Numpy ndarray [nsamples,label_arrays]`*) – The truth labels for each of the TL images. The supplied labels must be in the one-hot encoding format. We reproduce below what each individual label array should look like:-
  - Disk-dominated - `[1,0,0]`
  - Indeterminate - `[0,1,0]`
  - Bulge-dominated - `[0,0,1]`
- **`validation_imgs`** (*`Numpy ndarray [nsamples,x,y,ndim]`*) – The array of images which are to be used for the validation process. We insist on numpy arrays as many of the underlying deep learning frameworks work better with numpy arrays compared to other array-like elements.
- **`validation_labels`** (*`Numpy ndarray [nsamples,label_arrays]`*) – The truth labels for each of the validation images. The supplied labels must be in the one-hot encoding format. We reproduce below what each individual label array should look like:-
  - Disk-dominated - `[1,0,0]`
  - Indeterminate - `[0,1,0]`
  - Bulge-dominated - `[0,0,1]`
- **`input_shape`** (*`tuple of ints (x, y, ndim) or allowed str`*) – The shape of the images being used in the form of a tuple.

  This parameter can also take the following special values:-
  - `SDSS` - Sets the input shape to be (167,167,1) as was used for the SDSS g-band images in Ghosh et. al. (2020)
  - `CANDELS` - Sets the input shape to be (83,83,1) as was used for the CANDELS H-band images in Ghosh et. al. (2020)
- **`load_layers_bools`** (*`array of bools`*) – This variable is used to identify which of the 5 convolutional and 3 fully-connected layers of GaMorNet will be loaded during the transfer learning process from the supplied starting model. The rest of the layers will be initialized from scratch.

  The order of the bools correspond to the following layer numbers [2, 5, 8, 9, 10, 13, 15, 17] in GaMorNet. Please see Figure 4 and Table 2 of Ghosh et. al. (2020) to get more details.

The first five layers are the convolutional layers and the last three are the fully connected layers.

This parameter can also take the following special values which are handy when you are using our models to perform predictions:-

  - `load_bools_SDSS` - Sets the bools according to what was done for the SDSS data in Ghosh et. al. (2020)
  - `load_bools_CANDELS`- Sets the bools according to what was done for the CANDELS data in Ghosh et. al. (2020)

- **`trainable_bools`** (*`array of bools`*) – This variable is used to identify which of the 5 convolutional and 3 fully-connected layers of GaMorNet will be trainable during the transfer learning process. The rest are frozen at the values loaded from the previous model.

  The order of the bools correspond to the following layer numbers [2, 5, 8, 9, 10, 13, 15, 17] in GaMorNet. Please see Figure 4 and Table 2 of Ghosh et. al. (2020) to get more details. The first five layers are the convolutional layers and the last three are the fully connected layers.

  This parameter can also take the following special values which are handy when you are using our models to perform predictions:-

  - `train_bools_SDSS` - Sets the bools according to what was done for the SDSS data in Ghosh et. al. (2020)
  - `train_bools_CANDELS`- Sets the bools according to what was done for the CANDELS data in Ghosh et. al. (2020)

- **`model_load_path`** (*`str`*) – Full path to the saved Keras model, which will serve as the starting point for transfer learning.

  Additionally, this parameter can take the following special values

  - `SDSS_sim` – Downloads and uses the GaMorNet model trained on SDSS g-band simulations at z~0 from Ghosh et. al. (2020)
  - `SDSS_tl` – Downloads and uses the GaMorNet model trained on SDSS g-band simulations and real data at z~0 from Ghosh et. al. (2020)
  - `CANDELS_sim` – Downloads and uses the GaMorNet model trained on CANDELS H-band simulations at z~1 from Ghosh et. al. (2020)
  - `CANDELS_tl` – Downloads and uses the GaMorNet model trained on CANDELS H-band simulations and real data at z~1 from Ghosh et. al. (2020)

- **`files_save_path`** (*`str`*) – The full path to the location where files generated during the training process are to be saved. This includes the `metrics.csv` file as well as the trained model.

  Set this to `/dev/null` on a unix system if you don't want to save the output.

- **`epochs`** (*`int`*) – The number of epochs for which you want to train the model.
- **`checkpoint_freq`** (*`int`*) – The frequency (in terms of epochs) at which you want to save models. For eg. setting this to 25, would save the model at its present state every 25 epochs.
- **`batch_size`** (*`int`*) – This variable specifies how many images will be processed in a single batch. This is a hyperparameter. The default value is a good starting point

- **lr** (*float or schedule*) – 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.

  Instead of setting it at a single value, you can also set a schedule using `keras.optimizers.schedules.LearningRateSchedule`
- **momentum** (*float*) – The value of the momentum to be used in the gradient descent optimizer that is used to train GaMorNet. This must always be ≥ 0. This accelerates the gradient descent process. This is a hyperparameter. The default value is a good starting point.
- **decay** (*float*) – The amount of learning rate decay to be applied over each update.
- **nesterov** (*bool*) – Whether to apply Nesterov momentum or not.
- **loss** (*allowed str*) – The loss function to be used. If using the string option, you need to specify the name of the loss function. This can be set to be any loss available in `keras.losses`
- **save_model** (*bool*) – Whether you want to save the model in its final trained state.

  Note that this parameter does not affect the models saved by the `checkpoint_freq` parameter
- **verbose** (*{0, 1, 2}*) – The level of verbosity you want from Keras during the training process. 0 = silent, 1 = progress bar, 2 = one line per epoch.

Returns **Trained Keras Model**

Return type Keras `Model` class

3.4.2 TFLearn Module

The three major user oriented functions happen to be `gamornet_predict_tflearn()`, `gamornet_train_tflearn()` and `gamornet_tl_tflearn()` and are documented here. For the remainder of the functions, please have a look at the source code on GitHub.

`gamornet.tflearn_module.`**`gamornet_predict_tflearn`**(*img_array, model_load_path, input_shape, batch_size=64, individual_arrays=False, trainable_bools=[True, True, True, True, True, True, True, True], clear_session=False*)

Uses a TFLearn model to perform predictions on supplied images.

Parameters

- **img_array** (*Numpy ndarray[nsamples, x, y, ndim]*) – The array of images on which the predictions are to be performed. We insist on numpy arrays as many of the underlying deep learning frameworks work better with numpy arrays compared to other array-like elements.
- **model_load_path** (*str*) – Path to the saved model. Note that tflearn models usually consist of three files in the format `file_name.data`, `file_name.index`, `file_name.meta`. For this parameter, simply specify file_path/file_name.

  This parameter can also take the following special values
  - `SDSS_sim` – Downloads and uses GaMorNet models trained on SDSS g-band simulations at z~0 from Ghosh et. al. (2020)

  - `SDSS_tl` – Downloads and uses GaMorNet models trained on SDSS g-band simulations and real data at z~0 from Ghosh et. al. (2020)
  - `CANDELS_sim` – Downloads and uses GaMorNet models trained on CANDELS H-band simulations at z~1 from Ghosh et. al. (2020)
  - `CANDELS_tl` – Downloads and uses GaMorNet models trained on CANDELS H-band simulations and real data at z~1 from Ghosh et. al. (2020)

- **input_shape** (*tuple of ints (x, y, ndim) or allowed str*) – The shape of the images being used in the form of a tuple.

  This parameter can also take the following special values:-

  - `SDSS` - Sets the input shape to be (167,167,1) as was used for the SDSS g-band images in Ghosh et. al. (2020)
  - `CANDELS` - Sets the input shape to be (83,83,1) as was used for the CANDELS H-band images in Ghosh et. al. (2020)

- **batch_size** (*int*) – This variable specifies how many images will be processed in a single batch. Set this value to lower than the default if you have limited memory availability. This doesn't affect the predictions in any way.
- **individual_arrays** (*bool*) – If set to True, this will unpack the three returned arrays
- **trainable_bools** (*array of bools or allowed str*) – This variable is used to identify which of the 5 convolutional and 3 fully-connected layers of GaMorNet were set to trainable during the training phase of the model (which is now being used for prediction)

  The order of the bools correspond to the following layer numbers [2, 5, 8, 9, 10, 13, 15, 17] in GaMorNet. Please see Figure 4 and Table 2 of Ghosh et. al. (2020) to get more details. The first five layers are the convolutional layers and the last three are the fully connected layers.

  This parameter can also take the following special values which are handy when you are using our models to perform predictions:-

  - `train_bools_SDSS` - Sets the bools according to what was done for the SDSS data in Ghosh et. al. (2020)
  - `train_bools_CANDELS`- Sets the bools according to what was done for the CANDELS data in Ghosh et. al. (2020)

- **clear_session** (*bool*) – If set to True, this will clear the TensorFlow session currently running. This is handy while running GaMorNet in a notebook to avoid variable name confusions. (Sometimes, under the hood, TFLearn & Tensorflow reuse the same layer names leading to conflicts)

  Note that if set to True, you will lose access to any other graphs you may have run before.

Returns

predicted probabilities – The returned array consists of the probability for each galaxy to be disk-dominated, indeterminate and bulge-dominated respectively [disk_prob, indet_prob, bulge_prob]. If `individual_arrays` is set to `True`, the single array is unpacked and returned as three separate arrays in the same order.

The ordering of individual elements in this array corresponds to the array of images fed in.

Return type array_like

`gamornet.tflearn_module.`**`gamornet_train_tflearn`**(*training_imgs*, *training_labels*, *validation_imgs*, *validation_labels*, *input_shape*, *files_save_path='./'*, *epochs=100*, *max_checkpoints=1*, *batch_size=64*, *lr=0.0001*, *momentum=0.9*, *decay=0.0*, *nesterov=False*, *loss='categorical_crossentropy'*, *load_model=False*, *model_load_path='./'*, *save_model=True*, *show_metric=True*, *clear_session=False*)

Trains and return a GaMorNet model using TFLearn.

Parameters

- **`training_imgs`** (*`Numpy ndarray [nsamples,x,y,ndim]`*) – The array of images which are to be used for the training process. We insist on numpy arrays as many of the underlying deep learning frameworks work better with numpy arrays compared to other array-like elements.
- **`training_labels`** (*`Numpy ndarray [nsamples,label_arrays]`*) – The truth labels for each of the training images. The supplied labels must be in the one-hot encoding format. We reproduce below what each individual label array should look like:-
  - Disk-dominated - `[1,0,0]`
  - Indeterminate - `[0,1,0]`
  - Bulge-dominated - `[0,0,1]`
- **`validation_imgs`** (*`Numpy ndarray [nsamples,x,y,ndim]`*) – The array of images which are to be used for the validation process. We insist on numpy arrays as many of the underlying deep learning frameworks work better with numpy arrays compared to other array-like elements.
- **`validation_labels`** (*`Numpy ndarray [nsamples,label_arrays]`*) – The truth labels for each of the validation images. The supplied labels must be in the one-hot encoding format. We reproduce below what each individual label array should look like:-
  - Disk-dominated - `[1,0,0]`
  - Indeterminate - `[0,1,0]`
  - Bulge-dominated - `[0,0,1]`
- **`input_shape`** (*`tuple of ints (x, y, ndim) or allowed str`*) – The shape of the images being used in the form of a tuple.

  This parameter can also take the following special values:-
  - `SDSS` - Sets the input shape to be (167,167,1) as was used for the SDSS g-band images in Ghosh et. al. (2020)
  - `CANDELS` - Sets the input shape to be (83,83,1) as was used for the CANDELS H-band images in Ghosh et. al. (2020)
- **`files_save_path`** (*`str`*) – The full path to the location where the models generated during the training process are to be saved. The path should end with the name of the file. For eg. `/path/checkpoint`. This will result in model files of the form `checkpoint.meta`, `checkpoint.data` and `checkpoint.index` being saved.

  Set this to `/dev/null` on a unix system if you don't want to save the file(s)

- **epochs** (*int*) – The number of epochs for which you want to train the model.
- **max_checkpoints** (*int*) – TFLearn saves the model at the end of each epoch. This parameter controls how many of the most recent models are saved. For eg. setting this to 2, will save the model state during the most recent two epochs.
- **batch_size** (*int*) – This variable specifies how many images will be processed in a single batch. This is a hyperparameter. The default value is a good starting point
- **lr** (*float*) – 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.
- **momentum** (*float*) – The value of the momentum to be used in the gradient descent optimizer that is used to train GaMorNet. This must always be ≥ 0. This accelerates the gradient descent process. This is a hyperparameter. The default value is a good starting point.
- **decay** (*float*) – The amount of learning rate decay to be applied over each update.
- **nesterov** (*bool*) – Whether to apply Nesterov momentum or not.
- **loss** (*allowed str or function*) – The loss function to be used. If using the string option, you need to specify the name of the loss function. This can be set to be any loss available in `tflearn`
- **load_model** (*bool*) – Whether you want to start the training from a previously saved model.

  We strongly recommend using the `gamornet_tl_tflearn` function for more control over the process when starting the training from a previously saved model.
- **model_load_path** (*str*) – Required iff `load_model == True`. The path to the saved model.

  Note that tflearn models usually consist of three files in the format `file_name.data`, `file_name.index`, `file_name.meta`. For this parameter, simply specify file_path/file_name.
- **save_model** (*bool*) – Whether you want to save the model files at each epoch during training. This parameter should be used in conjunction with `max_checkpoints` to configure how many of the saved model files are preserved till the end.
- **show_metric** (*bool*) – Whether to display the training/testing metrics during training.
- **clear_session** (*bool*) – If set to True, this will clear the TensorFlow session currently running. This is handy while running GaMorNet in a notebook to avoid variable name confusions. (Sometimes, under the hood, TFLearn & Tensorflow reuse the same layer names leading to conflicts)

  Note that if set to True, you will lose access to any other graphs you may have run before.

Returns **Trained TFLearn Model**

Return type TFLearn `models.dnn.DNN` class `gamornet.tflearn_module.`**`gamornet_tl_tflearn`**(*training_imgs, training_labels, validation_imgs, validation_labels, input_shape, load_layers_bools=[True, True, True, True, True, True, True, True], trainable_bools=[True, True, True, True, True, True, True, True], model_load_path='./', files_save_path='./', epochs=100, max_checkpoints=1, batch_size=64, lr=1e-05, momentum=0.9, decay=0.0, nesterov=False, loss='categorical_crossentropy', save_model=True, show_metric=True, clear_session=False*)

Performs Transfer Learning (TL) using a previously trained GaMorNet model.

Parameters

- **`training_imgs`** (*`Numpy ndarray [nsamples,x,y,ndim]`*) – The array of images which are to be used for the TL process. We insist on numpy arrays as many of the underlying deep learning frameworks work better with numpy arrays compared to other array-like elements.
- **`training_labels`** (*`Numpy ndarray [nsamples,label_arrays]`*) – The truth labels for each of the TL images. The supplied labels must be in the one-hot encoding format. We reproduce below what each individual label array should look like:-
  - Disk-dominated - `[1,0,0]`
  - Indeterminate - `[0,1,0]`
  - Bulge-dominated - `[0,0,1]`
- **`validation_imgs`** (*`Numpy ndarray [nsamples,x,y,ndim]`*) – The array of images which are to be used for the validation process. We insist on numpy arrays as many of the underlying deep learning frameworks work better with numpy arrays compared to other array-like elements.
- **`validation_labels`** (*`Numpy ndarray [nsamples,label_arrays]`*) – The truth labels for each of the validation images. The supplied labels must be in the one-hot encoding format. We reproduce below what each individual label array should look like:-
  - Disk-dominated - `[1,0,0]`
  - Indeterminate - `[0,1,0]`
  - Bulge-dominated - `[0,0,1]`
- **`input_shape`** (*`tuple of ints (x, y, ndim) or allowed str`*) – The shape of the images being used in the form of a tuple.

  This parameter can also take the following special values:-
  - `SDSS` - Sets the input shape to be (167,167,1) as was used for the SDSS g-band images in Ghosh et. al. (2020)
  - `CANDELS` - Sets the input shape to be (83,83,1) as was used for the CANDELS H-band images in Ghosh et. al. (2020)
- **`load_layers_bools`** (*`array of bools`*) – This variable is used to identify which of the 5 convolutional and 3 fully-connected layers of GaMorNet will be loaded during the transfer learning process from the supplied starting model. The rest of the layers will be initialized from scratch.

The order of the bools correspond to the following layer numbers [2, 5, 8, 9, 10, 13, 15, 17] in GaMorNet. Please see Figure 4 and Table 2 of Ghosh et. al. (2020) to get more details. The first five layers are the convolutional layers and the last three are the fully connected layers.

This parameter can also take the following special values which are handy when you are using our models to perform predictions:-

- `load_bools_SDSS` - Sets the bools according to what was done for the SDSS data in Ghosh et. al. (2020)
- `load_bools_CANDELS`- Sets the bools according to what was done for the CANDELS data in Ghosh et. al. (2020)

- **`trainable_bools`** (*`array of bools`*) – This variable is used to identify which of the 5 convolutional and 3 fully-connected layers of GaMorNet will be trainable during the transfer learning process. The rest are frozen at the values loaded from the previous model.

  The order of the bools correspond to the following layer numbers [2, 5, 8, 9, 10, 13, 15, 17] in GaMorNet. Please see Figure 4 and Table 2 of Ghosh et. al. (2020) to get more details. The first five layers are the convolutional layers and the last three are the fully connected layers.

  This parameter can also take the following special values which are handy when you are using our models to perform predictions:-

  - `train_bools_SDSS` - Sets the bools according to what was done for the SDSS data in Ghosh et. al. (2020)
  - `train_bools_CANDELS`- Sets the bools according to what was done for the CANDELS data in Ghosh et. al. (2020)

- **`model_load_path`** (*`str`*) – Path to the saved model, which will serve as the starting point for transfer learning. Note that tflearn models usually consist of three files in the format `file_name.data`, `file_name.index`, `file_name.meta`. For this parameter, simply specify file_path/file_name.

  This parameter can also take the following special values

  - `SDSS_sim` – Downloads and uses GaMorNet models trained on SDSS g-band simulations at z~0 from Ghosh et. al. (2020)
  - `SDSS_tl` – Downloads and uses GaMorNet models trained on SDSS g-band simulations and real data at z~0 from Ghosh et. al. (2020)
  - `CANDELS_sim` – Downloads and uses GaMorNet models trained on CANDELS H-band simulations at z~1 from Ghosh et. al. (2020)
  - `CANDELS_tl` – Downloads and uses GaMorNet models trained on CANDELS H-band simulations and real data at z~1 from Ghosh et. al. (2020)

- **`files_save_path`** (*`str`*) – The full path to the location where the models generated during the training process are to be saved. The path should end with the name of the file. For eg. `/path/checkpoint`. This will result in model files of the form `checkpoint.meta`, `checkpoint.data` and `checkpoint.index` being saved.

  Set this to `/dev/null` on a unix system if you don't want to save the output.

- **`epochs`** (*`int`*) – The number of epochs for which you want to train the model.

- **`max_checkpoints`** (*`int`*) – TFLearn saves the model at the end of each epoch. This parameter controls how many of the most recent models are saved. For eg. setting this to 2, will save the model state during the most recent two epochs.

- **batch_size** (*int*) – This variable specifies how many images will be processed in a single batch. This is a hyperparameter. The default value is a good starting point
- **lr** (*float*) – 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.
- **momentum** (*float*) – The value of the momentum to be used in the gradient descent optimizer that is used to train GaMorNet. This must always be ≥ 0. This accelerates the gradient descent process. This is a hyperparameter. The default value is a good starting point.
- **decay** (*float*) – The amount of learning rate decay to be applied over each update.
- **nesterov** (*bool*) – Whether to apply Nesterov momentum or not.
- **loss** (*allowed str or function*) – The loss function to be used. If using the string option, you need to specify the name of the loss function. This can be set to be any loss available in `tflearn`
- **save_model** (*bool*) – Whether you want to save the model files at each epoch during training. This parameter should be used in conjunction with `max_checkpoints` to configure how many of the saved model files are preserved till the end.
- **show_metric** (*bool*) – Whether to display the training/testing metrics during training.
- **clear_session** (*bool*) – If set to True, this will clear the TensorFlow session currently running. This is handy while running GaMorNet in a notebook to avoid variable name confusions. (Sometimes, under the hood, TFLearn & Tensorflow reuse the same layer names leading to conflicts)

  Note that if set to True, you will lose access to any other graphs you may have run before.

Returns **Trained TFLearn Model**

Return type TFLearn `models.dnn.DNN` class

3.5 FAQs

1. Can I run GaMorNet on any galaxy image?

   No! Please see our recommendations in the *Public Data Release Handbook*.

2. I am having difficulty enabling GPU support. What should I do?

   Try using Google Colab like we have done in the *Tutorials*.

   Note that the underlying package that we use to interact with a GPU is TensorFlow. Look at these detailed instructions for enabling GPU support for TensorFlow. Alternatively, if you are running this on a supercomputer, ask the administrators for detailed instructions on installing TensorFlow.

3. I am getting an import error involving `GLIBC` or `libcudas.so` or `libm.so`.

   In all probability, you are getting these errors because TensorFlow cannot find the appropriate CUDA libraries. Please follow the instructions here. Alternatively if you are running this on a supercomputer, ask the administrators for detailed instructions on installing TensorFlow.

4. Should I use the Keras or TFLearn module if I myself don't have a preference?

   We recommend using the Keras module as we expect it to be better supported going forward. However, you may wish to take a look at the *Public Data Release Handbook* for differences between the two modules. It should be noted that the results in the original paper was obtained using TFLearn.

5. Is it worth enabling GPU support?

   We highly recommend running GaMorNet on a GPU if you are going to train your own models.

6. What if my question is not answered here?

   Please send me an e-mail at this `aritraghsh09+gamornet@xxxxx.com` GMail address. Additionally, 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

Index