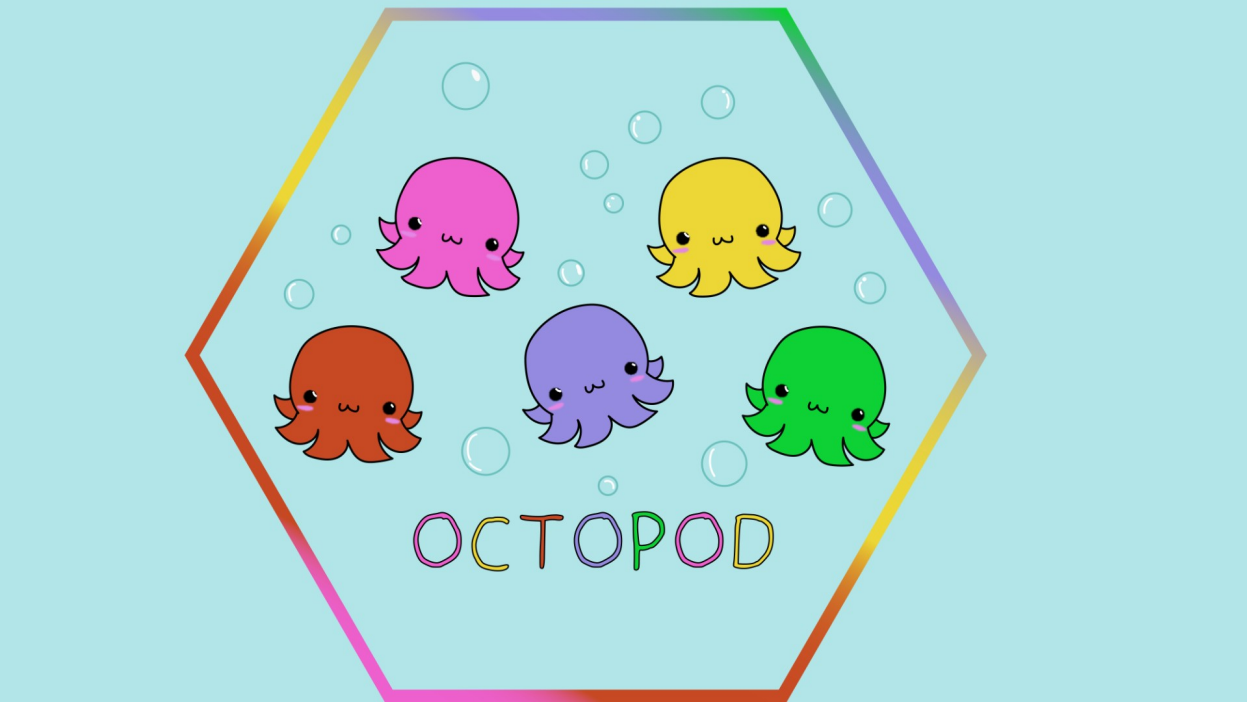
octopod Documentation

Release 3.3.0

ShopRunner data science team

May 01, 2023

CONTENTS



Octopod is a general purpose deep learning library developed by the ShopRunner Data Science team to train multi-task image, text, or ensemble (image + text) models.

What differentiates our library is that you can train a multi-task model with different datasets for each of your tasks. For example, you could train one model to label dress length for dresses and pants length for pants.

See the docs for more details.

To quickly get started, check out one of our tutorials in the `notebooks` folder. In particular, the `synthetic_data` tutorial provides a very quick example of how the code works.

Note 7/08/20: We are renaming this repository Octopod (previously called Tonks). The last version of the PyPI library under the name Tonks will not break but will warn the user to begin installing and using Octopod instead. No further development will continue under the name Tonks.

Note 6/12/20: Our team previously had a tradition of naming projects with terms or characters from the Harry Potter series, but we are disappointed by J.K. Rowling's persistent transphobic comments. In response, we will be renaming this repository, and are working to develop an inclusive solution that minimizes disruption to our users.

CHAPTER

ONE

STRUCTURE

- `notebooks`
  - `fashion_data`: a set of notebooks demonstrating training Octopod models on an open source fashion dataset consisting of images and text descriptions
  - `synthetic_data`: a set of notebooks demonstrating training Octopod models on a set of generated color swatches. This is meant to be an easy fast demo of the library's capabilities that can be run on CPU's.
- `octopod`
  - `ensemble`: code for ensemble models of text and vision models
  - `text`: code for text models with a BERT architecture
  - `vision`: code for vision models with ResNet50 architectures

CHAPTER

TWO

INSTALLATION

```
pip install octopod
```

You may get an error from the `tokenizer` package if you do not have a Rust compiler installed; see https://github.com/huggingface/transformers/issues/2831#issuecomment-592724471.

CHAPTER

THREE

NOTES

Currently, this library supports ResNet50 and BERT models.

In some of our documentation the terms `pretrained` and `vanilla` appear. `pretrained` is our shorthand for Octopod models that have been trained at least once already so their weights have been tuned for a specific use case. `vanilla` is our shorthand for base weights coming from `transformers` or `PyTorch` for the out-of-the-box BERT and ResNet50 models.

For our examples using text models, we use the transformers repository managed by huggingface. The most recent version is called `transformers`. The huggingface repo is the appropriate place to check on BERT documentation and procedures.

CHAPTER

FOUR

DEVELOPMENT

Want to add to or fix issues in Octopod? We welcome outside input and have tried to make it easier to test. You can run everything inside a docker container with the following:

```
# to build the container
# NOTE: this may take a while
docker build -t octopod .
# nvidia-docker run : basic startup with nvidia docker to access gpu
# --rm : deletes container when closed
# -p : exposes ports (ex: for jupyter notebook to work)
# bash : opens bash in the container once it starts
# "pip install jupyter && bash" : install requirements-dev and bash
nvidia-docker run \
    -it \
    --rm \
    -v "${PWD}:/octopod" \
    -p 8888:8888 \
    octopod /bin/bash -c "pip install jupyter && bash"
# run jupyter notebook
jupyter notebook --ip 0.0.0.0 --no-browser --allow-root --NotebookApp.token='' --
↪NotebookApp.password=''
```

4.1 Core Octopod

Core Octopod is made up of a learner and dataloader class designed for multi-task multi-dataset learning.

4.1.1 Multitask Learner

class octopod.learner.**MultiInputMultiTaskLearner**(*model*, *train_dataloader*, *val_dataloader*, *task_dict*, *loss_function_dict=None*, *metric_function_dict=None*)

Multi Input subclass of MultiTaskLearner class

Parameters

- **model** (*torch.nn.Module*) – PyTorch model to use with the Learner
- **train_dataloader** (MultiDatasetLoader) – dataloader for all of the training data
- **val_dataloader** (MultiDatasetLoader) – dataloader for all of the validation data

- **task_dict** (*dict*) – dictionary with all of the tasks as keys and the number of unique labels as the values

Notes

Multi-input datasets should return x's as a tuple/list so that each element can be sent to the appropriate device before being sent to the model see octopod.vision.dataset's OctopodImageDataset class for an example

class octopod.learner.**MultiTaskLearner**(*model*, *train_dataloader*, *val_dataloader*, *task_dict*, *loss_function_dict=None*, *metric_function_dict=None*)

Class to encapsulate training and validation steps for a pipeline. Based off the fastai learner.

Parameters

- **model** (*torch.nn.Module*) – PyTorch model to use with the Learner
- **train_dataloader** (MultiDatasetLoader) – dataloader for all of the training data. Set of labels must match *val_dataloader* or a ValueError will be thrown.
- **val_dataloader** (MultiDatasetLoader) – dataloader for all of the validation data. Set of labels must match *train_dataloader* or a ValueError will be thrown.
- **task_dict** (*dict*) – dictionary with all of the tasks as keys and the number of unique labels as the values
- **loss_function_dict** (*dict*) – dictionary where keys are task names (str) and values specify loss functions. A loss function can be specified using the special string value 'categorical_cross_entropy' for a multi-class task or 'bce_logits' for a multi-label task. Alternatively, it can be specified using a Callable that takes the predicted values and the target values as positional PyTorch tensor arguments and returns a float.

  Take care to ensure that the loss function includes a final activation function if needed – for instance, if your model is being used for classification but does not include a softmax layer then calculating cross-entropy properly requires performing the softmax classification within the loss function (this is handled by *nn.CrossEntropyLoss()* loss function). For our exisiting model architecture examples we do not apply final layer activation functions. Instead the desired activation functions are applied when needed. So we use 'categorical_cross_entropy' and 'bce_logits' loss functions which apply softmax and sigmoid activations to their inputs before calculating the loss.
- **metric_function_dict** (*dict*) – dictionary where keys are task names and values are metric calculation functions. If the input is a string matching a supported metric function *multi_class_acc* for multi-class tasks or *multi_label_acc* for multi-label tasks the loss will be filled in. A user can also input a custom metric function as a function for a given task key.

  custom metric functions must take in a *y_true* and *raw_y_pred* and output some *score* and *y_preds*. *y_preds* are the *raw_y_pred* values after an activation function has been applied and *score* is the output of whatever custom metrics the user wants to calculate for that task.

  See *learner_utils.loss_utils_config* for examples.

fit(*num_epochs*, *scheduler*, *step_scheduler_on_batch*, *optimizer*, *device='cuda:0'*, *best_model=False*, *smooth_loss_alpha=0.2*)

Fit the PyTorch model

Parameters

- **num_epochs** (*int*) – number of epochs to train

- **scheduler** (*torch.optim.lr_scheduler*) – PyTorch learning rate scheduler
- **step_scheduler_on_batch** (*bool*) – flag of whether to step scheduler on batch (if True) or on epoch (if False)
- **optimizer** (*torch.optim*) – PyTorch optimzer
- **device** (*str (defaults to 'cuda:0')*) – device to run calculations on
- **best_model** (bool (defaults to *False*)) – flag to save best model from a single *fit* training run based on validation loss The default is *False*, which will keep the final model from the training run. *True* will keep the best model from the training run instead of the model from the final epoch of the training cycle.
- **smooth_loss_alpha** (*float*) – Training loss values displayed during fitting and at the end of each epoch are exponentially weighted moving averages over batches. Using an exponentially weighted average over batches is a compromise between reporting the value from the most recent batch, which is highly relevant but noisy, and reporting a simple average over batches, which is more stable but reflects the value of the loss at the beginning of the epoch as much as at the end. *smooth_loss_alpha* controls how much weight is given to the current batch. It must be in the *(0, 1]* interval. Higher values are more like reporting only the most recent batch, while lower values are more like giving all batches equal weight, so this value controls the tradeoff between stability and relevance.

get_val_preds(*device='cuda:0'*)

Return true labels and predictions for data in self.val_dataloaders

Parameters **device** (*str (defaults to 'cuda:0')*) – device to run calculations on

Returns 'y_true': numpy array of true labels, shape: (num_rows,) 'y_pred': numpy of array of predicted probabilities: shape (num_rows, num_labels)

Return type Dictionary with dictionary for each task type

validate(*device='cuda:0'*, *pbar=None*)

Evaluate the model on a validation set

Parameters

- **loss_function** (*function*) – function to calculate loss with in model
- **device** (*str (defaults to 'cuda:0')*) – device to run calculations on
- **pbar** (*fast_progress progress bar (defaults to None)*) – parent progress bar for all epochs

Returns

- **overall_val_loss** (*float*) – overall validation loss for all tasks
- **val_loss_dict** (*dict*) – dictionary of validation losses for individual tasks
- **metrics_scores** (*dict*) – scores for individual tasks

4.1.2 Multitask Dataloader

class octopod.dataloader.**MultiDatasetLoader**(*loader_dict*, *shuffle=True*)
Load datasets for multiple tasks

Parameters

- **loader_dict** (*dict*) – dictonary of DataLoaders
- **shuffle** (*Boolean (defaults to True)*) – Flag for whether or not to shuffle the data

4.2 Octopod Ensemble

The ensemble aspects of Octopod are housed here. This includes sample model architectures, dataset class, and helper functions.

4.2.1 Model Architectures

class octopod.ensemble.models.multi_task_ensemble.**BertResnetEnsembleForMultiTaskClassificat**

PyTorch ensemble class for multitask learning consisting of a text and image models

This model is made up of multiple component models: - for text: Google's BERT model - for images: multiple ResNet50's (the exact number depends on how the image model tasks were split up)

You may need to train the component image and text models first before combining them into an ensemble model to get good results.

Note: For explicitness, *vanilla* refers to the *transformers* BERT or *PyTorch* ResNet50 weights while *pretrained* refers to previously trained Octopod weights.

Examples

The ensemble model should be used with pretrained BERT and ResNet50 component models. To initialize a model in this way:

```
image_task_dict = {
    'color_pattern': {
        'color': color_train_df['labels'].nunique(),
        'pattern': pattern_train_df['labels'].nunique()
    },
    'dress_sleeve': {
        'dress_length': dl_train_df['labels'].nunique(),
        'sleeve_length': sl_train_df['labels'].nunique()
    },
    'season': {
        'season': season_train_df['labels'].nunique()
    }
}
model = BertResnetEnsembleForMultiTaskClassification(
    image_task_dict=image_task_dict
)

resnet_model_id_dict = {
```

(continues on next page)

(continued from previous page)

```
    'color_pattern': 'SOME_RESNET_MODEL_ID1',
    'dress_sleeve': 'SOME_RESNET_MODEL_ID2',
    'season': 'SOME_RESNET_MODEL_ID3'
}

model.load_core_models(
    folder='SOME_FOLDER',
    bert_model_id='SOME_BERT_MODEL_ID',
    resnet_model_id_dict=resnet_model_id_dict
)

# DO SOME TRAINING

model.save(SOME_FOLDER, SOME_MODEL_ID)

# OR

model.export(SOME_FOLDER, SOME_MODEL_ID)
```

Parameters

- **image_task_dict** (*dict*) – dictionary mapping each pretrained ResNet50 models to a dictionary of the tasks it was trained on
- **dropout** (*float*) – dropout percentage for Dropout layer

static create_text_dict(*image_task_dict*)
Create a task dict for the text model from the image task dict

export(*folder*, *model_id*, *model_name=None*)
Exports the entire model state dict to a specific folder, along with the image_task_dict, which is needed to reinstantiate the model.

Parameters

- **folder** (*str or Path*) – place to store state dictionaries
- **model_id** (*int*) – unique id for this model
- **model_name** (*str (defaults to None)*) – Name to store model under, if None, will default to *multi_task_ensemble_{model_id}.pth*

Side Effects

saves two files:

- folder / f'multi_task_ensemble_{model_id}.pth'
- folder / f'image_task_dict_{model_id}.pickle'

forward(*x*)
Defines forward pass for ensemble model

Parameters **x** (*dict*) –

dictionary of torch tensors with keys:

- *bert_text*: integers mapping to BERT vocabulary

- *full_img*: tensor of full image
- *crop_img*: tensor of cropped image

Returns

Return type A dictionary mapping each task to its logits

freeze_bert()
Freeze all core BERT layers

freeze_classifiers_and_core()
Freeze pretrained classifier layers and core BERT/ResNet layers

freeze_ensemble_layers()
Freeze all final ensemble layers

freeze_resnets()
Freeze all core ResNet models layers

load(*folder*, *model_id*)
Loads the model state dicts for ensemble model from a specific folder. This will load all the model components including the final ensemble and existing pretrained *classifiers*.

Parameters

- **folder** (*str or Path*) – place where state dictionaries are stored
- **model_id** (*int*) – unique id for this model

Side Effects

loads from six files:

- folder / f'bert_dict_{model_id}.pth'
- folder / f'dropout_dict_{model_id}.pth'
- folder / f'image_resnets_dict_{model_id}.pth'
- folder / f'image_dense_layers_dict_{model_id}.pth'
- folder / f'ensemble_layers_dict_{model_id}.pth'
- folder / f'classifiers_dict_{model_id}.pth'

load_core_models(*folder*, *bert_model_id*, *resnet_model_id_dict*)
Loads the weights from pretrained BERT and ResNet50 Octopod models

Does not load weights from the final ensemble and classifier layers. use case is for loading SR_pretrained component BERT and image model weights into a new ensemble model.

Parameters

- **folder** (*str or Path*) – place where state dictionaries are stored
- **bert_model_id** (*int*) – unique id for pretrained BERT text model
- **resnet_model_id_dict** (*dict*) – dict with unique id's for pretrained image model, e.g. ``` resnet_model_id_dict = {

  'task1_task2': 'model_id1', 'task3_task4': 'model_id2', 'task5': 'model_id3'

Side Effects

loads from four files:

- folder / f'bert_dict_{bert_model_id}.pth'
- folder / f'dropout_dict_{bert_model_id}.pth'
- **folder / f'resnet_dict_{resnet_model_id}.pth'** for each resnet_model_id in the resnet_model_id_dict
- folder / f'dense_layers_dict_{resnet_model_id}.pth'

save(*folder*, *model_id*)

Saves the model state dicts to a specific folder. Each part of the model is saved separately, along with the image_task_dict, which is needed to reinstantiate the model.

Parameters

- **folder** (*str or Path*) – place to store state dictionaries
- **model_id** (*int*) – unique id for this model

Side Effects

saves six files:

- folder / f'bert_dict_{model_id}.pth'
- folder / f'dropout_dict_{model_id}.pth'
- folder / f'image_resnets_dict_{model_id}.pth'
- folder / f'image_dense_layers_dict_{model_id}.pth'
- folder / f'ensemble_layers_dict_{model_id}.pth'
- folder / f'classifiers_dict_{model_id}.pth'

unfreeze_classifiers()

Unfreeze pretrained classifier layers

unfreeze_classifiers_and_core()

Unfreeze pretrained classifiers and core BERT/ResNet layers

4.2.2 Dataset

class octopod.ensemble.dataset.**OctopodEnsembleDataset**(*text_inputs*, *img_inputs*, *y*, *tokenizer*, *max_seq_length=128*, *transform='train'*, *crop_transform='train'*)

Load image and text data specifically for an ensemble model

Parameters

- **text_inputs** (*pandas Series*) – the text to be used
- **img_inputs** (*pandas Series*) – the paths to images to be used
- **y** (*list*) – A list of dummy-encoded categories or strings, which will be encoded using a sklearn label encoder

- **tokenizer** (*pretrained BERT Tokenizer*) – BERT tokenizer likely from *transformers*
- **max_seq_length** (*int (defaults to 128)*) – Maximum number of tokens to allow
- **transform** (*str or list of PyTorch transforms*) – specifies how to preprocess the full image for a Octopod image model To use the built-in Octopod image transforms, use the strings: *train* or *val* To use custom transformations supply a list of PyTorch transforms.
- **crop_transform** (*str or list of PyTorch transforms*) – specifies how to preprocess the center cropped image for a Octopod image model To use the built-in Octopod image transforms, use strings *train* or *val* To use custom transformations supply a list of PyTorch transforms.

class octopod.ensemble.dataset.**OctopodEnsembleDatasetMultiLabel**(*text_inputs*, *img_inputs*, *y*, *tokenizer*, *max_seq_length=128*, *transform='train'*, *crop_transform='train'*)

Multi label subclass of OctopodEnsembleDataset

Parameters

- **text_inputs** (*pandas Series*) – the text to be used
- **img_inputs** (*pandas Series*) – the paths to images to be used
- **y** (*list*) – a list of lists of binary encoded categories or strings with length equal to number of classes in the multi-label task. For a 4 class multi-label task a sample list would be [1,0,0,1], A string example would be ['cat','dog'], (if the classes were ['cat','frog','rabbit','dog]), which will be encoded using a sklearn label encoder to [1,0,0,1].
- **tokenizer** (*pretrained BERT Tokenizer*) – BERT tokenizer likely from *transformers*
- **max_seq_length** (*int (defaults to 128)*) – Maximum number of tokens to allow
- **transform** (*str or list of PyTorch transforms*) – specifies how to preprocess the full image for a Octopod image model To use the built-in Octopod image transforms, use the strings: *train* or *val* To use custom transformations supply a list of PyTorch transforms.
- **crop_transform** (*str or list of PyTorch transforms*) – specifies how to preprocess the center cropped image for a Octopod image model To use the built-in Octopod image transforms, use strings *train* or *val* To use custom transformations supply a list of PyTorch transforms.

4.3 Learner Utils

This section contains helper code for the Octopod learner pipelines for supporting multiple loss functions and metrics for individual tasks.

4.3.1 Metric Utils

octopod.learner_utils.metrics_utils.**multi_class_accuracy**(*y_true*, *y_raw_preds*)

Takes in raw outputs from Octopod task heads and outputs an accuracy metric and the processed predictions after a softmax as been applied

Parameters

- **y_true** (*np.array*) – Target labels for a specific task for the predicted samples in *y_raw_preds*
- **y_raw_preds** (*np.array*) – predicted values for the validation set for a specific task

Returns

- **acc** (*float*) – Output of a sklearn accuracy score function
- **y_preds** (*np.array*) – array of predicted values where a softmax has been applied

octopod.learner_utils.metrics_utils.**multi_label_accuracy**(*y_true*, *y_raw_preds*)

Takes in raw outputs from Octopod task heads and outputs an accuracy metric and the processed predictions after a sigmoid as been applied

Parameters

- **y_true** (*np.array*) – Target labels for a specific task for the predicted samples in *y_raw_preds*
- **y_raw_preds** (*np.array*) – predicted values for the validation set for a specific task

Returns

- **acc** (*float*) – Output of a sklearn accuracy score function
- **y_preds** (*np.array*) – array of predicted values where a sigmoid has been applied

4.4 Octopod Text

The text aspects of Octopod are housed here. This includes sample model architectures and a dataset class.

4.4.1 Model Architectures

class octopod.text.models.multi_task_bert.**BertForMultiTaskClassification**(*config*, *pretrained_task_dict=None*, *new_task_dict=None*, *dropout=0.1*)

PyTorch BERT class for multitask learning. This model allows you to load in some pretrained tasks in addition to creating new ones.

Examples

To instantiate a completely new instance of BertForMultiTaskClassification and load the weights into this architecture you can use the *from_pretrained* method of the base class by specifying the name of the weights to load, e.g.:

```
model = BertForMultiTaskClassification.from_pretrained(
    'bert-base-uncased',
    new_task_dict=new_task_dict
)

# DO SOME TRAINING

model.save(SOME_FOLDER, SOME_MODEL_ID)
```

To instantiate an instance of BertForMultiTaskClassification that has layers for pretrained tasks and new tasks, you would do the following:

```
model = BertForMultiTaskClassification.from_pretrained(
    'bert-base-uncased',
    pretrained_task_dict=pretrained_task_dict,
    new_task_dict=new_task_dict
)

model.load(SOME_FOLDER, SOME_MODEL_DICT)

# DO SOME TRAINING
```

Parameters

- **config** (*json file*) – Defines the BERT model architecture. Note: you will most likely be instantiating the class with the *from_pretrained* method so you don't need to come up with your own config.
- **pretrained_task_dict** (*dict*) – dictionary mapping each pretrained task to the number of labels it has
- **new_task_dict** (*dict*) – dictionary mapping each new task to the number of labels it has
- **dropout** (*float*) – dropout percentage for Dropout layer

export (*folder*, *model_id*, *model_name=None*)

Exports the entire model state dict to a specific folder.

Note: if the model has *pretrained_classifiers* and *new_classifers*, they will be combined into the *pretrained_classifiers* attribute before being saved.

Parameters

- **folder** (*str or Path*) – place to store state dictionaries
- **model_id** (*int*) – unique id for this model
- **model_name** (*str (defaults to None)*) – Name to store model under, if None, will default to *multi_task_bert_{model_id}.pth*

Side Effects

saves one file:

- folder / model_name

forward(*tokenized_input*)
Defines forward pass for Bert model

Parameters **tokenized_input** (*torch tensor of integers*) – integers represent tokens for each word

Returns

Return type A dictionary mapping each task to its logits

freeze_bert()
Freeze all core Bert layers

freeze_pretrained_classifiers_and_bert()
Freeze pretrained classifier layers and core Bert layers

import_model(*folder*, *file*)
Imports the entire model state dict from a specific folder.

Note: to export a model based on the import_model from this method, use the export method

Parameters

- **folder** (*str or Path*) – place to store state dictionaries
- **file** (*str*) – filename for the exported model object

load(*folder*, *model_id*)
Loads the model state dicts from a specific folder.

Parameters

- **folder** (*str or Path*) – place where state dictionaries are stored
- **model_id** (*int*) – unique id for this model

Side Effects

loads from three files:

- folder / f'bert_dict_{model_id}.pth'
- folder / f'dropout_dict_{model_id}.pth'
- folder / f'pretrained_classifiers_dict_{model_id}.pth'

save(*folder*, *model_id*)
Saves the model state dicts to a specific folder. Each part of the model is saved separately to allow for new classifiers to be added later.

Note: if the model has *pretrained_classifiers* and *new_classifers*, they will be combined into the *pretrained_classifiers_dict*.

Parameters

- **folder** (*str or Path*) – place to store state dictionaries
- **model_id** (*int*) – unique id for this model

Side Effects

saves three files:

- folder / f'bert_dict_{model_id}.pth'
- folder / f'dropout_dict_{model_id}.pth'
- folder / f'pretrained_classifiers_dict_{model_id}.pth'

unfreeze_pretrained_classifiers()
Unfreeze pretrained classifier layers

unfreeze_pretrained_classifiers_and_bert()
Unfreeze pretrained classifiers and core Bert layers

4.4.2 Dataset

class octopod.text.dataset.**OctopodTextDataset**(*x*, *y*, *tokenizer*, *max_seq_length=128*)
Load data for use with a BERT model

Parameters

- **x** (*pandas Series*) – the text to be used
- **y** (*list*) – A list of dummy-encoded or string categories will be encoded using an sklearn label encoder
- **tokenizer** (*pretrained BERT Tokenizer*) – BERT tokenizer likely from *transformers*
- **max_seq_length** (*int (defaults to 128)*) – Maximum number of tokens to allow

class octopod.text.dataset.**OctopodTextDatasetMultiLabel**(*x*, *y*, *tokenizer*, *max_seq_length=128*)
Multi label subclass of OctopodTextDataset

Parameters

- **x** (*pandas Series*) – the text to be used
- **y** (*list*) – a list of lists of binary encoded categories or string categories with length equal to number of classes in the multi-label task. For a 4 class multi-label task a sample list would be [1,0,0,1], A string example would be ['cat','dog'], (if the classes were ['cat','frog','rabbit','dog]), which will be encoded using a sklearn label encoder to [1,0,0,1].
- **tokenizer** (*pretrained BERT Tokenizer*) – BERT tokenizer likely from *transformers*
- **max_seq_length** (*int (defaults to 128)*) – Maximum number of tokens to allow

4.5 Octopod Vision

The computer vision aspects of Octopod are housed here. This includes sample model architectures, dataset class, and helper functions.

4.5.1 Model Architectures

class octopod.vision.models.multi_task_resnet.**ResnetForMultiTaskClassification**(*pretrained_task_*
new_task_dict=N
*load_pretrained_*

PyTorch image attribute model. This model allows you to load in some pretrained tasks in addition to creating new ones.

Examples

To instantiate a completely new instance of ResnetForMultiTaskClassification and load the weights into this architecture you can set *pretrained* to True:

```
model = ResnetForMultiTaskClassification(
    new_task_dict=new_task_dict,
    load_pretrained_resnet = True
)

# DO SOME TRAINING

model.save(SOME_FOLDER, SOME_MODEL_ID)
```

To instantiate an instance of ResnetForMultiTaskClassification that has layers for pretrained tasks and new tasks, you would do the following:

```
model = ResnetForMultiTaskClassification(
    pretrained_task_dict=pretrained_task_dict,
    new_task_dict=new_task_dict
)

model.load(SOME_FOLDER, SOME_MODEL_DICT)

# DO SOME TRAINING
```

Parameters

- **pretrained_task_dict** (*dict*) – dictionary mapping each pretrained task to the number of labels it has
- **new_task_dict** (*dict*) – dictionary mapping each new task to the number of labels it has
- **load_pretrained_resnet** (*boolean*) – flag for whether or not to load in pretrained weights for ResNet50. useful for the first round of training before there are fine tuned weights

export(*folder*, *model_id*, *model_name=None*)

Exports the entire model state dict to a specific folder. Note: if the model has *pretrained_classifiers* and *new_classifiers*, they will be combined into the *pretrained_classifiers* attribute before being saved.

Parameters

- **folder** (*str or Path*) – place to store state dictionaries
- **model_id** (*int*) – unique id for this model
- **model_name** (*str (defaults to None)*) – Name to store model under, if None, will default to *multi_task_bert_{model_id}.pth*

Side Effects

saves one file:

- folder / model_name

forward(*x*)
Defines forward pass for image model

Parameters

- **x** (*dict of image tensors containing tensors for*) –
- **and cropped images. the full image tensor** (*full*) –
- **the key 'full_img' and the cropped tensor has** (*has*) –
- **key 'crop_img'** (*the*) –

Returns

Return type A dictionary mapping each task to its logits

freeze_all_pretrained()
Freeze pretrained classifier layers and core model layers

freeze_core()
Freeze all core model layers

freeze_dense()
Freeze all core model layers

load(*folder*, *model_id*)
Loads the model state dicts from a specific folder.

Parameters

- **folder** (*str or Path*) – place where state dictionaries are stored
- **model_id** (*int*) – unique id for this model

Side Effects

loads from three files:

- folder / f'resnet_dict_{model_id}.pth'
- folder / f'dense_layers_dict_{model_id}.pth'
- folder / f'pretrained_classifiers_dict_{model_id}.pth'

save(*folder*, *model_id*)
Saves the model state dicts to a specific folder. Each part of the model is saved separately to allow for new classifiers to be added later.

Note: if the model has *pretrained_classifiers* and *new_classifers*, they will be combined into the *pretrained_classifiers_dict*.

Parameters

- **folder** (*str or Path*) – place to store state dictionaries
- **model_id** (*int*) – unique id for this model

Side Effects

saves three files:

- folder / f'resnet_dict_{model_id}.pth'
- folder / f'dense_layers_dict_{model_id}.pth'
- folder / f'pretrained_classifiers_dict_{model_id}.pth'

unfreeze_pretrained_classifiers()
Unfreeze pretrained classifier layers

unfreeze_pretrained_classifiers_and_core()
Unfreeze pretrained classifiers and core model layers

4.5.2 Dataset

class octopod.vision.dataset.**OctopodImageDataset**(*x*, *y*, *transform='train'*, *crop_transform='train'*)
Load data specifically for use with a image models

Parameters

- **x** (*pandas Series*) – file paths to stored images
- **y** (*list*) – A list of dummy-encoded categories or strings For instance, y might be [0,1,2,0] for a 3 class problem with 4 samples, or strings which will be encoded using a sklearn label encoder
- **transform** (*str or list of PyTorch transforms*) – specifies how to preprocess the full image for a Octopod image model To use the built-in Octopod image transforms, use the strings: *train* or *val* To use custom transformations supply a list of PyTorch transforms
- **crop_transform** (*str or list of PyTorch transforms*) – specifies how to preprocess the center cropped image for a Octopod image model To use the built-in Octopod image transforms, use strings *train* or *val* To use custom transformations supply a list of PyTorch transforms

class octopod.vision.dataset.**OctopodImageDatasetMultiLabel**(*x*, *y*, *transform='train'*, *crop_transform='train'*)
Subclass of OctopodImageDataset used for multi-label tasks

Parameters

- **x** (*pandas Series*) – file paths to stored images

- **y** (*list*) – a list of lists of binary encoded categories or strings with length equal to number of classes in the multi-label task. For a 4 class multi-label task a sample list would be [1,0,0,1], A string example would be ['cat','dog'], (if the classes were ['cat','frog','rabbit','dog]), which will be encoded using a sklearn label encoder to [1,0,0,1].
- **transform**(*str or list of PyTorch transforms*) – specifies how to preprocess the full image for a Octopod image model To use the built-in Octopod image transforms, use the strings: *train* or *val* To use custom transformations supply a list of PyTorch transforms
- **crop_transform**(*str or list of PyTorch transforms*) – specifies how to preprocess the center cropped image for a Octopod image model To use the built-in Octopod image transforms, use strings *train* or *val* To use custom transformations supply a list of PyTorch transforms

4.5.3 Helper Functions

octopod.vision.helpers.**center_crop_pil_image**(*img*)

Helper function to crop the center out of images.

Utilizes the centercrop function from *wildebeest*

Parameters **img** (*array*) – PIL image array

Returns **PIL.Image**

Return type Slice of input image corresponding to a cropped area around the center

4.6 Contributing and Making PRs

4.6.1 How to Contribute

We welcome contributions in the form of issues or pull requests!

We want this to be a place where all are welcome to discuss and contribute, so please note that this project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms. Find the Code of Conduct in the `CODE-OF-CONDUCT.md` file on GitHub or in the Code of Conduct section of read the docs.

If you have a problem using Octopod or see a possible improvement, open an issue in the GitHub issue tracker. Please be as specific as you can.

If you see an open issue you'd like to be fixed, take a stab at it and open a PR!

4.6.2 Pull Requests

To create a PR against this library, please fork the project and work from there.

Steps

1. Fork the project via the Fork button on Github
2. Clone the repo to your local disk.
3. Create a new branch for your PR.

```
git checkout -b my-awesome-new-feature
```

1. Install requirements (probably in a virtual environment)

```
virtualenv venv
source venv/bin/activate
pip install -r requirements-dev.txt
pip install -e .
```

1. Develop your feature
2. Submit a PR to main! Someone will review your code and merge your code into main when it is approved.

PR Checklist

- Ensure your code has followed the Style Guidelines below
- Run the linter on your code

```
source venv/bin/activate
flake8 octopod tests
```

- Make sure you have written unittests where appropriate
- Make sure the unittests pass

```
source venv/bin/activate
pytest -v
```

- Update the docs where appropriate. You can rebuild them with the commands below.

```
cd docs/
make html
open build/html/index.html
```

- Update the CHANGELOG

Style Guidelines

For the most part, this library follows PEP8 with a couple of exceptions.

- Indent with 4 spaces
- Lines can be 100 characters long
- Docstrings should be numpy style docstrings.
- Your code should be Python 3 compatible
- When in doubt, follow the style of the existing code

- We prefer single quotes for one-line strings unless using double quotes allows us to avoid escaping internal single quotes.

4.7 Contributor Covenant Code of Conduct

4.7.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

4.7.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

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

4.7.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

4.7.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

4.7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement by submitting this anonymous form or by sending an email to opensource@shoprunner.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

4.7.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the project community.

4.7.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

PYTHON MODULE INDEX

O

INDEX

O

R

S

U

V