

MiniScrub Documentation

Version 0.1.0

By: Nathan LaPierre

August 28, 2017

Table of Contents

Overview and Rationale.....	2
Installation and Dependencies.....	3
Basic Usage.....	4
Major Scripts and Usage.....	5
cigarToPctId.py.....	5
miniscrub.py.....	6
pileup.py.....	7
vgg.py.....	8
References.....	9

Overview and Rationale

MiniScrub is a method for *de novo* scrubbing low-quality portions of long sequencing reads with deep learning. New sequencing technologies such as Oxford Nanopore MinION [1] and PacBio SMRT [2] have made high throughput long read sequencing feasible. These longer read lengths (1kbp to >100kbp) are better than short reads for analysis tasks such as assembly, alignment, and clustering due in part to their ability to span longer repeats [3]. However, these technologies are currently affected by high error rates, which can range for example from 5% to 40% per read [4].

MiniScrub provides several benefits:

- Entirely *de novo* (does not require a reference database) and thus can be used on unknown or ambiguous sequences
- Does not require an additional set of short reads, unlike the common hybrid error correction method
- Leverages Deep Learning, a powerful class of machine learning algorithms that have achieved record-breaking results in many fields such as computer vision and natural language processes, and is increasingly being applied in life sciences. Convolutional Neural Networks (CNNs), which we use in this project, have shown particularly promising results. See for example:
 - DeepVariant [5], a record breaking variant calling method using CNNs
 - Classifying skin lesion types from images with dermatologist-level accuracy using CNNs [6]
 - Using CNNs to detect mitosis in breast cancer histology images [7]

MiniScrub is most effective when there is medium to high sequencing depth, and when cutting off very low quality read segments (i.e. >20% error rate). Running it will require a fairly significant amount of compute time, storage, and memory, and is thus best suited for use on cloud, cluster, grid, or supercomputer platforms that have these resources.

Installation and Dependencies

The MiniScrub files can be downloaded from GitHub: <https://github.com/nlapier2/MiniScrub>

If you are using git and would like to clone the repository, navigate to your command line (Mac, Linux, or Windows git client) and run the following line:

```
git clone https://github.com/nlapier2/MiniScrub.git
```

To download as a zip, use this link: <https://github.com/nlapier2/MiniScrub/archive/master.zip>
Then you can unzip it using either a file explorer window or on the command line using the “unzip” command (Mac or Linux).

MiniScrub is written in Python 2 and Python 3 and has the following dependencies:

- Standard python data science libraries: numpy, scipy, pandas, and scikit-learn
- Minimap (<https://github.com/lh3/minimap>)
- Keras (<https://github.com/fchollet/keras>)
- TensorFlow (<https://github.com/tensorflow/tensorflow>)
- If you are training the network (as opposed to just using an existing model), you must have SAM files with alignment information for the ground truth

If you are on the NERSC Cori supercomputer, you can load these dependencies by running the following:

```
module load python  
module load deeplearning
```

If you already have a python model loaded, you may need to unload it first:

```
module unload python
```

Note that pileup.py uses Python 3 instead of Python 2. On Cori you would instead have to run:

```
module load python/3.6-anaconda-4.4
```

Basic Usage

Here we cover a basic sample usage of MiniScrub. Consult the “Major Scripts and Usage” section for a full list of options. Please note that these steps will take a very long time (up to several days) to run if you use the default settings and have many reads (i.e. over a million).

As a prerequisite to running MiniScrub, you must have a fastq file containing your reads. If you are also training the network, you must have a SAM file mapping those reads to reference sequences, so that this ground truth information can be used to train the network.

The lines below represent a typical run through of the MiniScrub pipeline. Please note that the step of training a network is optional, as a pretrained model will be made available online soon. We use the example names “reads.fastq” and “mapped-reads.sam” for the prerequisite files.

Running our modified version of minimap (generates PAF file with minimizer locations listed):
Temp

Generating pileup images and saving them in plots/ folder (please note that this script uses Python 3; on some computers, run python command instead of python3 command):

```
python3 pileup.py --mapping minimapped-reads.paf --reads  
reads.fastq --saveplots --plotdir plots/
```

Loading a model and using it to scrub all 100bp segments with predicted 25% or higher error rate (--cutoff 0.75) using pileup images in plots/ folder:

```
python miniscrub.py --reads reads.fastq --cutoff 0.75  
--load model.hd5 --input plots/ --output  
scrubbed-reads.fastq
```

If training the network, execute these lines after generating the pileup images.

Generating labels for each 100bp segment of each read using SAM file:

```
python cigarToPctId.py --sam mapped-reads.sam --paf  
minimapped-reads.paf --outfile labels.txt
```

Training a network on all images in plots/ folder:

```
python vgg.py --input plots/ --labels labels.txt --output  
model.hd5
```

Major Scripts and Usage

Below are the help outputs for the major scripts that make up MiniScrub, which can each be generated in the terminal by running “python [scriptname].py -h”. Many of the options (debug, limit_*, etc) have to do with speeding up the runtime by using only some reads, and are helpful for doing test runs. Other options provide for the ability to run a naive SVM baseline, based on minimizer counts, to test whether a newly-trained network is more effective.

cigarToPctId.py

```
usage: cigarToPctId.py [-h] [--amount AMOUNT] [--compression {none,gzip}]
                        [--limit_paf LIMIT_PAF] [--limit_sam LIMIT_SAM]
                        [--mode {minimizers,bases}] [--outfile OUTFILE]
                        [--paf PAF] --sam SAM
```

Use cigar strings in sam files to compute percent identities.

optional arguments:

-h, --help	show this help message and exit
--amount AMOUNT	Number of bases/minimizers per label.
--compression {none,gzip}	Compression format, or none
--limit_paf LIMIT_PAF	Optionally limit the number of reads from paf file.
--limit_sam LIMIT_SAM	Optionally limit the number of reads from sam file.
--mode {minimizers,bases}	Labels for minimizers or bases.
--outfile OUTFILE	File to output label information to.
--paf PAF	Path to paf file with minimizers.
--sam SAM	Path to the sam file with ground truth labels.

miniscrub.py

```
usage: miniscrub.py [-h] [--compression {none,gzip}] [--cutoff CUTOFF]
                  [--debug DEBUG] [--input INPUT] [--labels LABELS] --load
                  LOAD [--output OUTPUT] [--reads READS]
                  [--segment_size SEGMENT_SIZE] [--window_size WINDOW_SIZE]
```

Use saved keras model to scrub reads.

optional arguments:

-h, --help show this help message and exit

--compression {none,gzip}
 Compression of reads file ("none" or "gzip").

--cutoff CUTOFF Scrub read segments below this percent identity.

--debug DEBUG Number of images to use in debug mode. If <= 0, non-
 debug mode (default).

--input INPUT Directory with png pileup images. Default: current
 directory.

--labels LABELS Path to image labels file. If provided, will NOT trim.
 Labels must correspond with segment_size.

--load LOAD Path to keras model file to load. Required.

--output OUTPUT File to write scrubbed reads to.

--reads READS Path to reads file. Default: do not trim (output
 statsitics instead).

--segment_size SEGMENT_SIZE
 Neural net segment size to predict. Keep as default
 unless network retrained.

--window_size WINDOW_SIZE
 Neural net window size to predict. Keep as default
 unless network retrained.

pileup.py (NOTE: this script uses Python 3)

```
usage: pileup.py [-h] [--avgdepth AVGDEPTH] [--color {bw,rgb}]
                [--compression {none,gzip}] [--debug] [-k K]
                [--limit_fastq LIMIT_FASTQ] [--limit_reads LIMIT_READS]
                --mapping MAPPING [--maxdepth MAXDEPTH]
                [--mode {whole,minimizers}] [--plotdir PLOTDIR]
                [--processes PROCESSES] --reads READS [--saveplots] [--verbose]
```

Create pileups from .paf read-to-read mapping and fastq reads.

optional arguments:

-h, --help show this help message and exit

--avgdepth AVGDEPTH Average coverage depth (not currently used).

--color {bw,rgb} Color mode (bw or rgb).

--compression {none,gzip} Compression format, or none

--debug Debug mode.

-k K K-mer size of minimizers. Required.

--limit_fastq LIMIT_FASTQ Limit number of reads to scan from fastq file.

--limit_reads LIMIT_READS Limit number of reads to generate pileups for.

--mapping MAPPING Path to the .paf file of read-to-read mappings.

--maxdepth MAXDEPTH Maximum number of matched reads per pileup image.

--mode {whole,minimizers} Whole read or minimizers-only.

--plotdir PLOTDIR If --saveplots is used, directory path to save plots in.

--processes PROCESSES Number of multiple processes to run concurrently.

--reads READS Path to the .fastq reads file.

--saveplots If used, will plot the pileups and save them.

--verbose Verbose output option.

vgg.py

```
usage: vgg.py [-h] [--baseline] [--classify CLASSIFY] [--debug DEBUG]
             [--epochs EPOCHS] [--extra EXTRA] [--input INPUT]
             [--labels LABELS] [--load LOAD] [--loadsvm LOADSVM]
             [--output OUTPUT] [--outputsvm OUTPUTSVM]
             [--segment_size SEGMENT_SIZE] [--test_input TEST_INPUT]
             [--test_labels TEST_LABELS] [--window_size WINDOW_SIZE]
```

Create pileups from .paf read-to-read mapping and fastq reads.

optional arguments:

-h, --help show this help message and exit

--baseline Compare with baseline: SVM trained on # of minimizers
 matched.

--classify CLASSIFY Turn into classification problem; specify threshold.

--debug DEBUG Number of examples to use in debug mode. If 0, non-
 debug mode (default).

--epochs EPOCHS Number of epochs to train the network. Default: 5.

--extra EXTRA Number of fully connected layers to add to VGG.
 Default: 0

--input INPUT Directory with png pileup images. Default: current
 directory.

--labels LABELS Path to image labels file.

--load LOAD Path to keras model file to load. Default: do not
 load.

--loadsvm LOADSVM Path to saved pickled SVM file. Default: do not load.

--output OUTPUT File to write model to. Default: no output.

--outputsvm OUTPUTSVM
 File to write svm model to. Default: no output.

--segment_size SEGMENT_SIZE
 Size of read segments to evaluate.

--test_input TEST_INPUT
 Directory of separate set of images to test model on.

--test_labels TEST_LABELS
 Path to file with labels for images in test set.

--window_size WINDOW_SIZE
 Window size for VGG. Window >= segment size.

References

1. <https://nanoporetech.com/products/minion>
2. <http://www.pacb.com/smrt-science/>
3. <https://genomebiology.biomedcentral.com/articles/10.1186/gb-2013-14-6-405>
4. <http://genome.cshlp.org/content/25/11/1750.full>
5. <http://www.biorxiv.org/content/early/2016/12/21/092890>
6. <https://www.nature.com/nature/journal/v542/n7639/abs/nature21056.html>
7. https://link.springer.com/chapter/10.1007/978-3-642-40763-5_51