Manual

Version: 0.8. Released on: 27.03.2020. Published by: Bastian Th. Eichenberger.

0. Introduction

This guide will provide you with enough information to use and train a fluffily certified model for biomedical image segmentation. Read everything **carefully** and if you find things to be unclear or outdated let me know by mail – bastian.eichenberger(at)fmi.ch.

What is deep learning?

I might add more detail here. For now, if you're interested you can read this introduction, this paper or this video.

What is my workflow doing?

The entirety of my workflow is divided into four main steps:

1. Setting up the environment

Initially, the programming and application environment must be set up. This is important as programming applications constantly change. A piece of code is only guaranteed to run if the same dependency versions (code written and published by other people) are used.

2. Generating labeled data

For a deep learning model to train it needs data. Ideally the more the better. As some wise old man once said – "garbage in, garbage out" – the higher quality your input is, the better your output will be. So please don't rush this step and take your time. The number of images depends on how variable the images to predict will be. For example, if you try to segment a pattern in a high throughput assay which always looks the same you will need significantly fewer images than if you try to predict DAPI stained nuclei across multiple microscopes, magnifications, and samples. A good starting point for simpler models is 20-30 images.

3. Training and testing a model

This step automatically trains and tests a model. There is a lot to be said here and I will add more information soon.

4. Using a pre-trained model

Quite often there might already be a model available to solve your problem. In that case, one simply has to use the model on the data to be analyzed.

The command-line isn't a pick-up line for computers…

All subsequent code has to be executed in the command-line. Below is a short tutorial to get you going. Note the commands are only for Unix systems (Linux / Mac). All Windows users… Please consider your choices and use a real operating system.

To access the command-line open the application “Terminal”. Type the commands below in the text field and execute them by simply pressing [enter].

A directory is nothing but a folder. All you need to know is how to find which directory you are in and how to change the current directory. There are three commands you will use:

```
pwd
ls
cd
```

- “pwd” – small P, small W, and small D. This stands for “present working directory” and tells you the complete path of your directory. For Windows – “echo %cd%”.
- “ls” – small L and small S. This stands for “list” and lists all files in the current directory. Use this to find out what directories are available for you to go to. For Windows – “dir”.
- “cd” – small C and small D. This stands for “change directory” and is used to navigate to other directories. There are two directions of travel:
    - “cd ..” – Upwards / out of the current directory
    - “cd folder_xy” – Into the specified folder, here “folder_xy”

Always use the complete path given by “pwd” in the commands below. This ensures that no mix-ups between two equally named directories can happen. Watch this video for a more visual experience.

Note – the easiest way to get to your desired folder simply use the “cd” command and drag and drop the folder you want to access into the terminal window. This will automatically generate the right path.

```
cd # Drag and drop the desired folder
```

What should I do if things fail?

Don’t panic. There might still be some undetected bugs. Try to solve the problem **on your own** for at least 15 minutes by reading and understanding what went wrong. Also, don’t forget to try turning it off and on. If things haven’t solved themselves feel free to contact me at the mail address above.

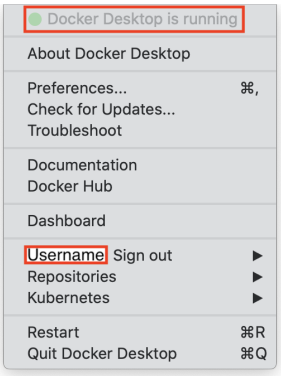
1. Setting up the environment

Docker

Docker is a handy tool that allows us to securely build and share any application. Because docker packages all its applications in “containers” (think of them as self-sufficient applications) the old problem with “it runs on my machine” is no longer applicable. To install it you will need to create a docker account and download the “Docker Desktop” program.

Start by creating an account here. Once the account was created and the email was verified, go ahead and download “Docker Desktop” here. All Linux users can use apt-get (see detailed instructions here).

Once installed on your local machine, open up the application and log in using your newly created account. After authorization you should be greeted with something like follows:

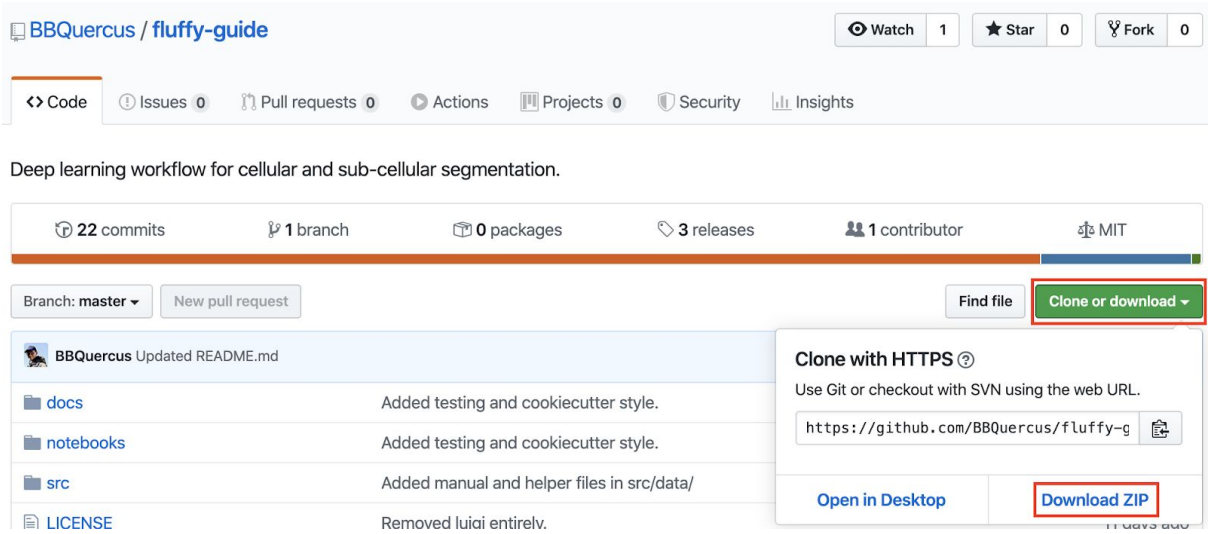


As further verification that everything is working as intended, open up the command-line and type “docker”. If a short usage manual is displayed you did everything according to plan.

Note – if all you care about is to run the “fluffy” interface you do not need any further software and can directly skip ahead to section 4. Using a pre-trained model.

Code Repository

All code that will be used is found in the Github repository here. Download its contents and unzip.



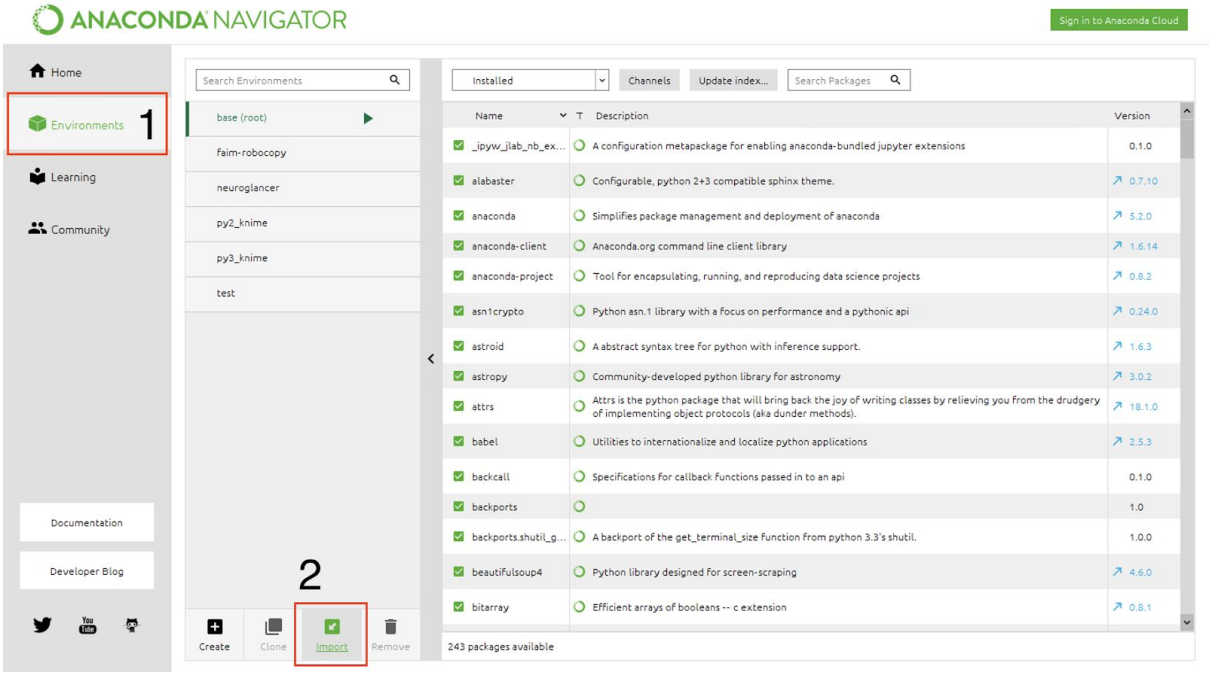
Practice your command-line skills by navigating to the just downloaded repository.

Fiji

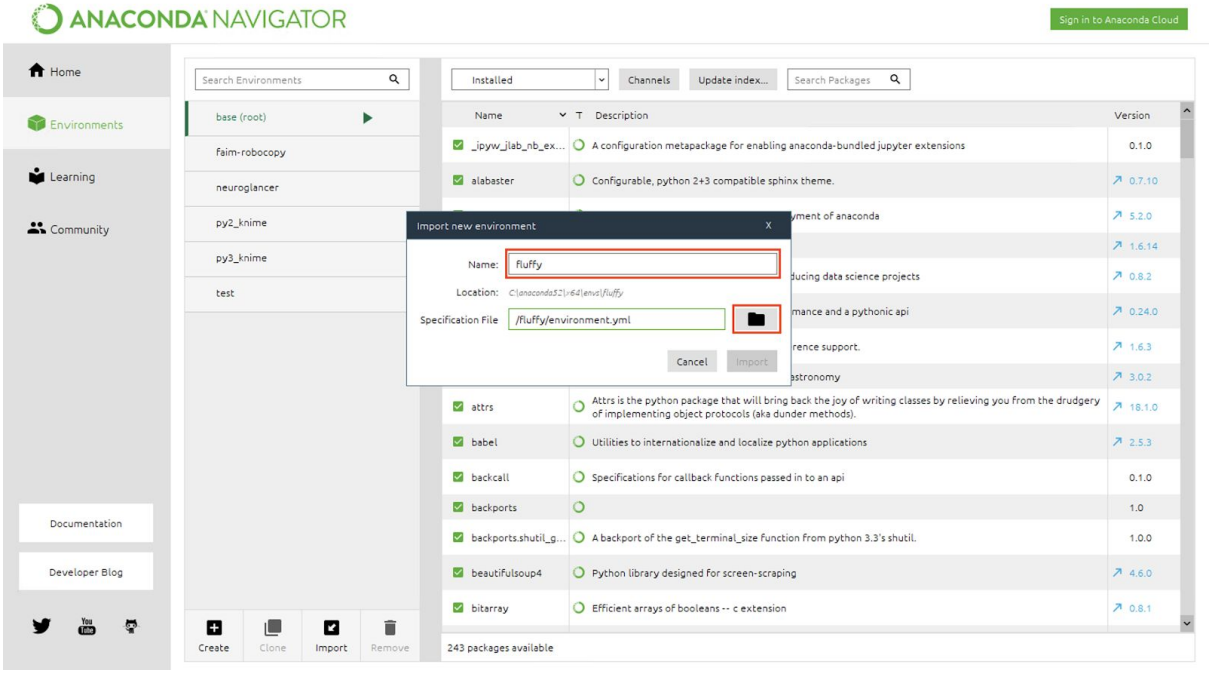
To annotate data we will need the free image processing program Fiji. Please download and install the latest version for your operating system here.

Anaconda

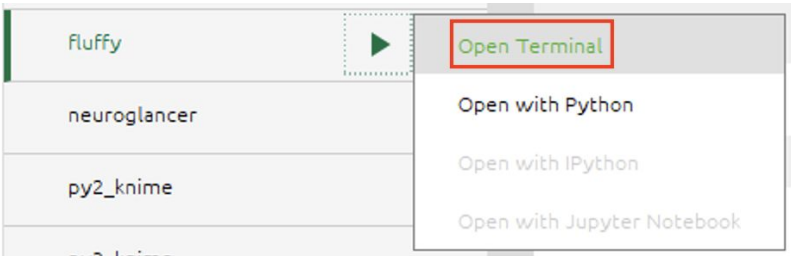
Anaconda is a package manager for python. We will use it to install all packages needed to run the workflow. Please follow the graphical installation manual for Python 3.x for your operating system here. Once installed, open up the Anaconda navigator. To create a new environment (set of python packages) go to the “Environments” tab and click on “Import” (see next page).



In the pop-up window select a name for your environment such as “fluffy”. Select the “environment.yml” file which can be found in the code repository you downloaded above.



Follow the installation procedure and you should have a new “environment” (set of python packages) called “fluffy” (or your name of choice). Click on the arrow symbol next to the environment name and try to open a terminal from within the Anaconda navigator (see next page).



This completes your installation. Don’t forget to always open a new terminal window through the Anaconda navigator.

2. Generating labeled data

Note – the ALL CAPS words used below must be changed to suit your specific situation. The names are selected to make sense if read in context.

File setup

Before we get to labeling images, we need to set up a proper file structure. Please run the following script in the command-line after selecting a base directory (e.g. the Desktop) and the folder name (e.g. nucleus_labeling). Don't forget you can drag and drop folders to automatically get the path.

```
# From within the fluffy Github directory
cd src/data/
python make_directories.py

# Path to the base directory: PATH/TO/BASE/DIRECTORY
# Folder name: NAME_OF_FOLDER
```

Once run, you should have the basic file structure at your specified location. There should be three folders ("Raw", "Labeling", and "Processed") which will be described in more detail. For now, add all images that you want to label to the "Raw" directory. Please only use one of the supported file types: ".stk", ".czi", ".tiff", ".tif", ,".jpeg", ".jpg" or ".png". Only images in the "Raw" directory will be used – any subdirectories will be ignored.

We will convert all files into a uniform format to provide the basis to start labeling. The trained model will only take 2D input. Therefore, images will be converted to 2D – specify whether you want to annotate every slice/time frame of each image in the "Raw" directory. If not, state if images are Z-stacks (performs an automatic maximum projection) or time-lapses (only selects a single time frame). Lastly, if your images are already 2D you can simply say N (no) every time. Please run:

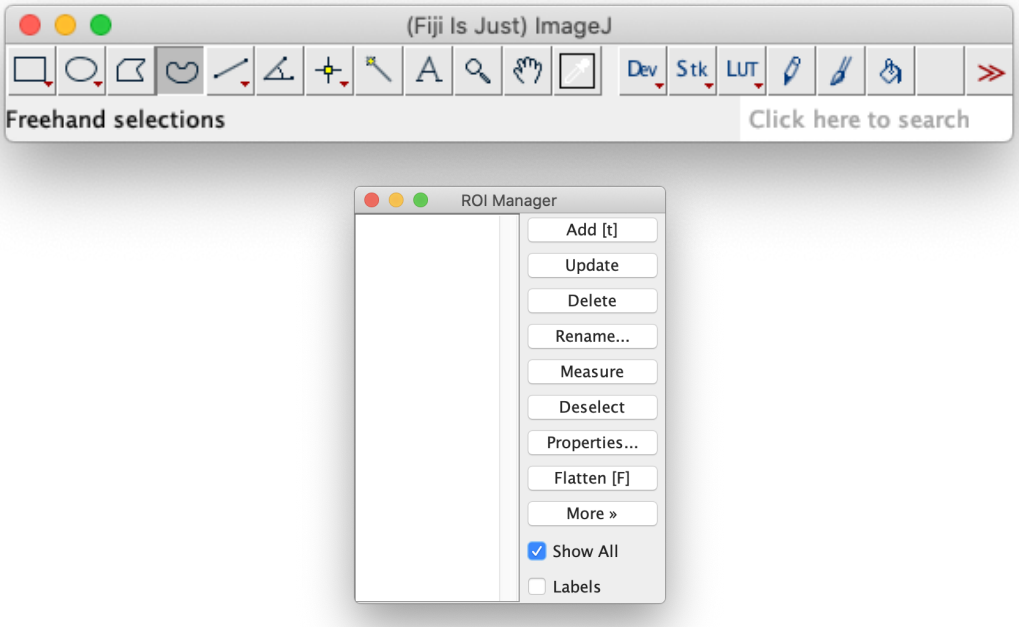
```
# From within the fluffy Github directory
cd src/data/
python make_labelling.py

# Path to the base directory: PATH/TO/BASE/DIRECTORY
# Save all slices [y/N]: CHOOSE
# Timelapse [y/N]: CHOOSE
# ZStack [y/N]: CHOOSE
```

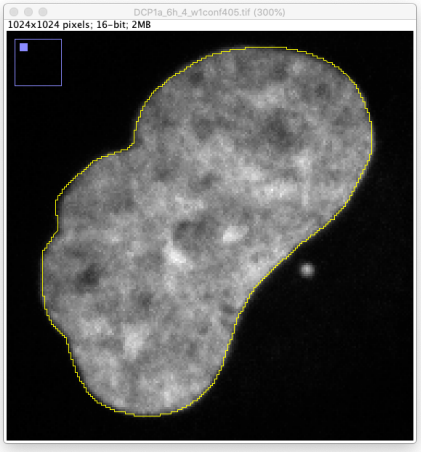
This just converted all files in the “Raw” directory to the 2D images of the “.tif” file format inside the “Labelling” directory.

File annotation

To annotate images, we will use Fiji. Select the “Freehand selections tool. Furthermore, open up the “ROI Manager” which can be found under “Analyze > Tools > ROI Manager”. Select the “Show All” option in the ROI Manager. Your setup should now look like this.



Open the first image from within the “Labeling/IMAGE_NAME/images” directory and start drawing around the border of one object in your image. Here is an example of a nucleus.



After circling one object click “Add” or [t] to add the annotated object to the ROI manager. An item with a numeric name should appear in the ROI manager. Continue circling and adding the objects until all annotations have been added to the manager. Save all annotations by selecting “More > Save…” in the ROI manager. Navigate to the directory “Labeling/IMAGE_NAME” in the just-opened pop-up window. Use the default name “RoiSet.zip” select “Save”. To clear the cache of the ROI Manager, select all annotations and press “Delete”. Repeat the same procedure until all your images have been annotated.

File post-processing

If you followed the steps above you should now have “RoiSet.zip” files in all “Labeling/IMAGE_NAMES” directories corresponding to that image. We will now use a Fiji macro to convert every set of annotations into label masks. Open up the “make_masks.ijm” file found in the “src/data” directory inside fluffy by dragging it to the Fiji menu bar. At the very top of the document change “root” to your base directory (specify the directory without a trailing backslash). Execute the macro by selecting “Run” right below the document.

All “Labeling/IMAGE_NAME/masks” directories should now contain a bunch of files corresponding to each annotation labeled “mask_NUMBER.png”. Check if your file structure looks the same:

```
# Inside the Processed directory
├──── File_Name_1
│     ├──── RoiSet.zip
│     ├──── images
│     │     └──── File_Name_1.tif
│     └──── masks
│          ├──── mask_0.png
│          ├──── mask_....png
│          └──── mask_n.png
│
├──── File_Name_2
│     ├──── RoiSet.zip
│     ├──── images
│     │     └──── File_Name_2.tif
│     └──── masks
│          ├──── mask_0.png
│          ├──── mask_....png
│          └──── mask_n.png
│
└──── etc.
```

If this is the case, the last step is to merge all of these annotations into a single label map. You can do this by running:

```
# From within the fluffy Github directory
cd src/data/
python make_label_maps.py

# Path to base directory: PATH/TO/BASE/DIRECTORY
```

You should be greeted by a populated “Processed” directory. Inside, the image files and merged label maps are within the corresponding folders. You can now use the “Processed” directory for training without any further manipulation.

3. Training and testing a model

“Houston we have a problem” – Training a model, especially if the process is fully automated, takes a lot of computational power. To avoid waiting forever for training to finish one has to use GPUs. Unfortunately setting up GPUs isn’t as straightforward as one could hope for. Currently, the easiest option is to contact me once you have finished preparing enough labeled training data.

The automated training and testing process consist of the following steps:

1. Splitting of training and testing data

We split data into two main sets. One for training and one for testing. This step is important to evaluate model performance. Our model learns from the training data set and is evaluated data it has never seen (testing data).

2. Bayesian probability based hyperparameter optimization using cross-validation

No set of training data is equal. Similarly, there is no one-size-fits-all neural network. Instead, a variety of parameters (hyperparameters) have to be tweaked and tuned to find suitable ones for the problem at hand. Bayesian probability improves this process by iteratively searching across a hyperparameter search space. At the end of this process, one should have somewhat suitable hyperparameters for the following steps.
For each optimization step, we use cross-validation. Cross-validation splits up the training dataset into multiple training and validation datasets. Training and validation are performed on each subset. The mean accuracy is then used for optimization. This gives us a good feeling on how well the model generalizes (performs on other data). To learn more about this procedure you can read up here.

3. Training one final model

Using the found hyperparameters and enough confidence provided by cross-validation, we train a final model. This model is trained for longer to maximize the capability of the model to learn important features. This training step is performed on the entirety of training and validation data. The testing data is now used as the validation set to minimize overfitting.

Two main types are available – “binary” and “categorical”:

- The binary model outputs a True/False mask telling us if an object is there or not. If we take nuclei as an example the output tells us if the pixel is a nucleus (blue) or background/something else (yellow).

- The categorical model can be used to predict instances (individual objects) by also predicting an “object border” class (green). These borders help to separate nearby or partially touching masks which would be classified as one large nucleus using the binary model. Use this only if you care about counting objects with higher accuracy.

If running on a GPU, change the “tensorflow” requirement in the “environments.yml” file to “tensorflow-gpu”. Start the training regiment by running:

```
# From within the fluffy Github directory
cd src/training/
python train_model.py \
    --dir_in 'PATH/TO/DATA/DIRECTORY' \
    --dir_out 'PATH/TO/SAVE/MODEL/FILES' \
    --name 'NAME_OF_MODEL' \
    --binary True/False \  # Select model type
    --use_defaults True/False  # If hyperparameters should be tuned
```

This will take around 2-5 hours on a decent GPU. Once Training is complete you will have the following files in the output directory:

- train_val / test.pkl – Pickled list of all train_val / test files used.
- .gz – Skopt file to look at hyperparameter optimization in more detail.
- .h5 – Final and best model file with the selected hyperparameters.
- .csv – Log of training progress (does not include hyperparameter optimization).
- Folder – Tensorboard callback to visualize training progress (does not include hyperparameter optimization).

4. Using a pre-trained model

There are two ways of using pre-trained models. Either through the command-line or the visual interface “fluffy”. Both of them are discussed below.

Fluffy interface

To open up our interface we will use Docker. First, the latest version has to be downloaded from Docker Hub. Due to regular updates being released, it is always a good idea to check which version we currently use here.

Open up docker and make sure “Docker Desktop” is running. Open up the command-line, replace “VERSION” with the latest version and run the following. Alternatively, using the link above you can simply copy the complete command.

```
# From anywhere in your system
docker pull bbquercus/fluffy:VERSION  # Replace version with the latest
```

Once complete, check if all images (blueprints) were downloaded. Do this using:

```
# From anywhere in your system
docker images
```

If “bbquercus/fluffy” is one of them, the download succeeded. Every time you want to use “fluffy” you have to tell docker to start running the container (convert the image/blueprint into a container):

```
# From anywhere in your system
docker run \
      -p 5000:5000 \
      bbquercus/fluffy:VERSION
```

Execute and open up a browser of choice and go to the URL:

```
localhost:5000
```

You will be greeted with the fluffiest of image processing software. Once the analysis is completed and “fluffy” is no longer required it's time to stop the container. To do this you have to find the unique “CONTAINER ID” assigned to any running container. The “CONTAINER ID” is the alphanumeric ID displayed by the first command below.

```
# To list currently running containers
docker container ls

# To stop the desired container
```

```
docker stop CONTAINER ID
```

You just stopped the container from running, freeing up some disk space, and allowing you to enjoy cotton candy in real life.

Command-line interface

Simply have a model file and image ready and run the following:

```
# From within the fluffy Github directory
cd src/inference/
python predict_model.py

# Model file: PATH/TO/MODEL_FILE.h5
# Image file/folder: PATH/TO/FOLDER/OR/IMAGE.png
```

About

If you find this helpful for your research please cite:

```
@misc{Fluffy,
      author = {Bastian Th., Eichenberger},
      title = {Fluffy},
      year = {2020},
      publisher = {GitHub},
      journal = {GitHub repository},
      howpublished = {\\url{https://github.com/bbquercus/fluffy}}
```

For assistance or to report bugs, please raise an issue on GitHub. A thank you to the Friedrich Miescher Institute's bioinformatics team that provides GPU access for easy model training.