

Figure 1: Base SEM Image

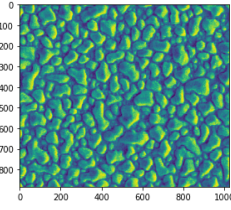


Figure 2: Filtered with 5x5 averaging

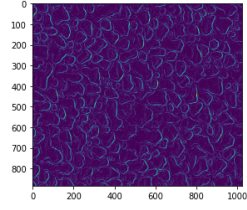


Figure 3: Gradient amplitude of the filtered image

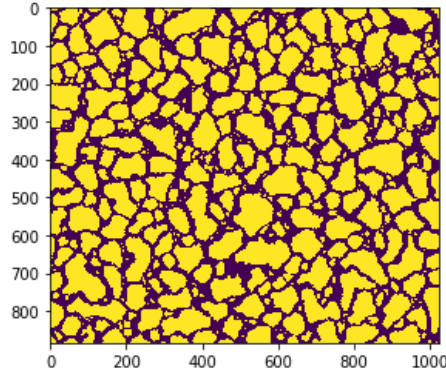


Figure 4: The hand-labeled grain mask for Fig. 1.

1 Grain Programs: Training and Using the Neural Net

1.1 Part 1: Split Original Image and Mask into Train and Test Files

Main Program: `ClusterPFM_Format.py`

In order to train and test on subsets of a main image without converting that image to an array each time, I split the image into a series of arrays, each stored in their own file folder. The **`ClusterPFM_Format.py`** program starts with the base image (Fig. 1), and makes a filtered version (Fig. 2), and an array of the gradient amplitude of the filtered version (Fig. 3). The user then defines the number N of subimages to take and the partition that will be saved for testing p . This turns the base image and mask (Fig. 4) into a $N \times N$ grid of subimages indexed 0 to $N^2 - 1$. If the image index is a multiple of p , it is considered a testing subimage. Four copies of each sub-image are saved which are flipped horizontally, vertically, or both. Bounding boxes are also defined for each subimage.

To summarize the input and output files:

- Input: Base image and mask. Saved in “base_img” and “image_path” variable locations respectively.
- Outputs: Subimages saved in:
 - Base image subimages saved in *PFD_Train_Img* folder or *PFD_Test_Img* folder
 - Filtered image subimages saved in *PFD_Train_Img2* folder or *PFD_Test_Img2* folder
 - Gradient image subimages saved in *PFD_Train_Img3* folder or *PFD_Test_Img3* folder
 - Mask image subimages saved in *PFD_Train_Masks* folder or *PFD_Test_Masks* folder
 - Bounding box 2d arrays (shape of $N_{obj} \times 4$) image subimages saved in *PFD_Train_boxes* folder or *PFD_Test_boxes* folder

1.2 Part 2: File Organization in Sherlock

It is feasible to run the neural net on a home computer without a GPU, but not to train it. To that end, I set the code up to run on the Sherlock computer cluster.

To run the training code, you will need:

- The output files of **ClusterPFM.Format.py** organized into folders as described in Sec. 1.1 and shown in Fig. 5.
- Helper python code, provided in the same GitHub repo and seen in Fig. 6: **coco_eval.py**, **utils.py**, **coco_utils.py**, **engine.py**, and **transforms.py**
- A slurm file to run the code, mine is **PFMTest.sh** and is shown in Fig. 7. This can be written with the nano text editor in Sherlock.
- The main training program, **PFM_MyImages_Sherlock.py**
- A python environment in your Sherlock login node with new versions of torch and torchvision installed (you cannot use the pre-installed version of torch or torchvision in Sherlock, they are too old). You can do this with pip.

1.3 Part 3: Training the Mask/Slurm Job

Main Program: **PFM_MyImages_Sherlock.py**

The training program, **PFM_MyImages_Sherlock.py**, uses the outputs **ClusterPFM.Format.py** to train a NN. This program trains a neural network to recognize the bounding boxes and outlines of detected images based

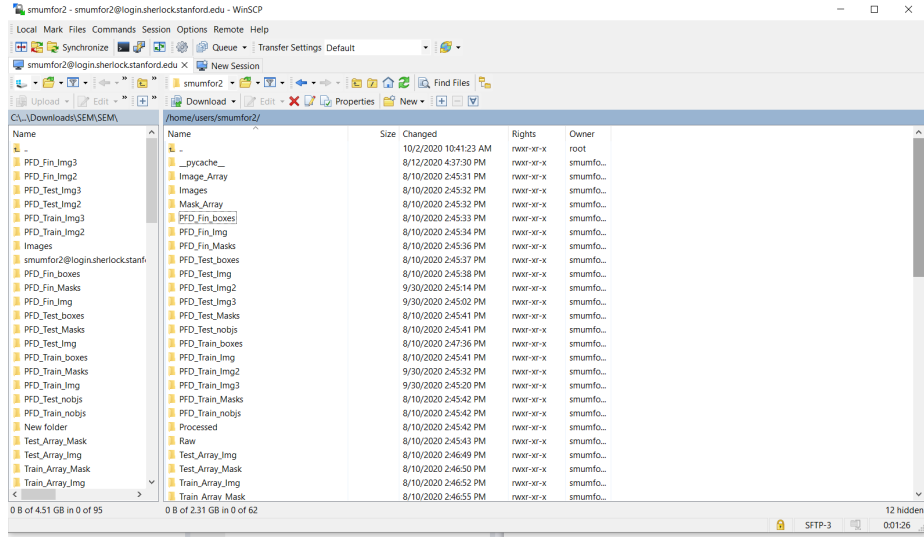


Figure 5: The folders holding base image and mask information in the remote computer (Sherlock), shown through WinSCP.

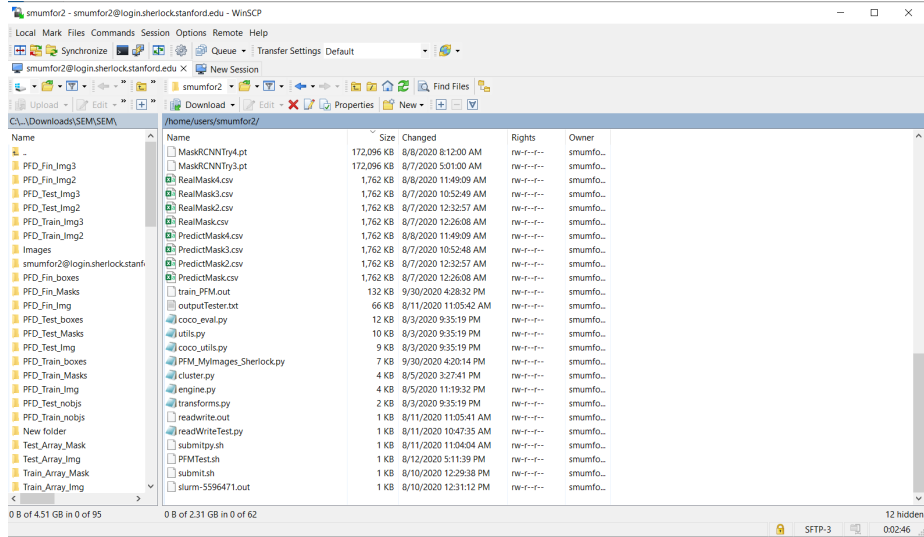
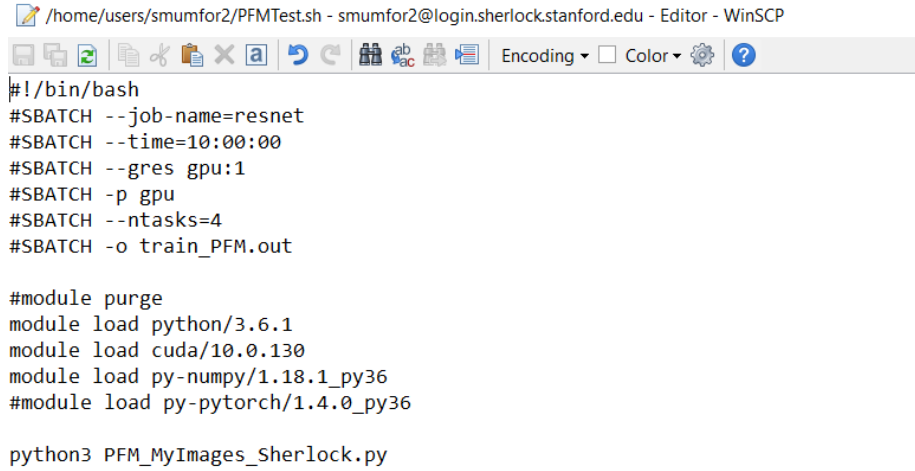


Figure 6: The files in the home directory in Sherlock.



```
#!/bin/bash
#SBATCH --job-name=resnet
#SBATCH --time=10:00:00
#SBATCH --gres gpu:1
#SBATCH -p gpu
#SBATCH --ntasks=4
#SBATCH -o train_PFM.out

#module purge
module load python/3.6.1
module load cuda/10.0.130
module load py-numpy/1.18.1_py36
#module load py-pytorch/1.4.0_py36

python3 PFM_MyImages_Sherlock.py
```

Figure 7: Slurm file with instructions on how to run the training job on a GPU to Sherlock.

on Mask R-CNN (<https://arxiv.org/abs/1703.06870>), itself based on Faster R-CNN (<https://arxiv.org/abs/1506.01497>).

You will need to edit the training program slightly to change the name of the trained neural network if you don't want to overwrite an existing NN. The NN name for the full trained model and just fit parameters are at the end of the file. Once done, run it with a slurm file. Instructions on running a job using a slurm file such as **PFMTest.sh** and monitoring job progress are provided at <https://www.sherlock.stanford.edu/docs/getting-started/submitting/>.

1.4 Part 4a: Using The Neural Net With a Known Grain Mask

Main Programs: ClusterFinFormat.py, DataStitch.py, CompareMasks.py

In order to evaluate the mask-generating network performance, we need to convert a full image to a predicted mask image. The first program, **ClusterFinFormat.py**, does the same operation as **ClusterPFMFormat.py**, except instead of splitting the train and test data and saving flipped copies of each, all $N \times N$ sub-images are saved in the folders *PFD_Fin_Img*, *PFD_Fin_Img2*, and *PFD_Fin_Img3* and the masks and boxes in *PFD_Fin_Masks* and *PFD_Fin_boxes*. The program **DataStitch.py** uses the neural network and the saved files from **ClusterFinFormat.py** to generate one predicted mask image. Finally, **CompareMasks.py** repairs errors caused by stitching together the sub-images and compares the predicted grain mask to the base mask used for training.

1.5 Part 4b: Using the Neural Net Without a Grain Mask

Main Programs: `ClusterFinFormat_noMask.py`, `DataStitchJustModel.py`

The programs `ClusterFinFormat_noMask.py` and `DataStitchJustModel.py` do the same things as the programs described in Sec. 1.4, but they do not require mask files and thus only generate a predicted mask.