PHYS 139/239 Final Presentation

Group 6:

Aditya Sriram

Andy Wan

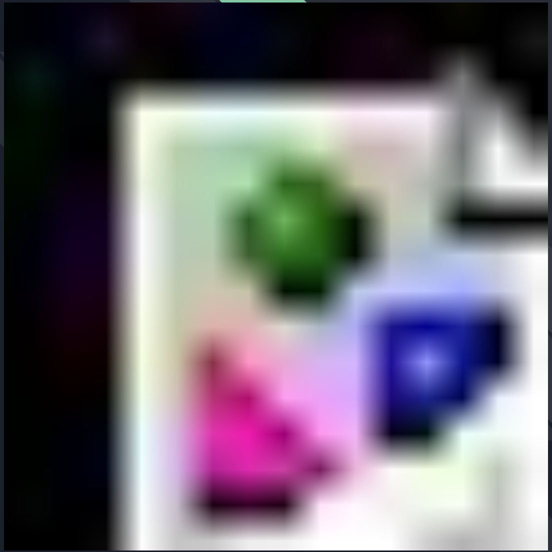
Boyan Stoychev

Donald Dean

Sean Chen

Introduction

Studying dynamics of battery cathode during charging/discharging

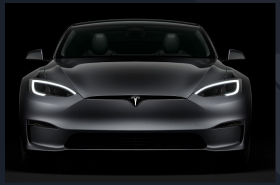


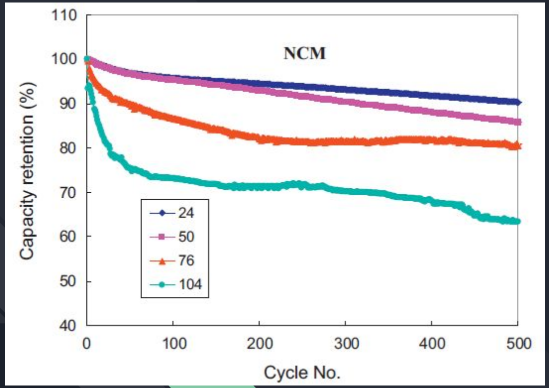
Scientific Motivation

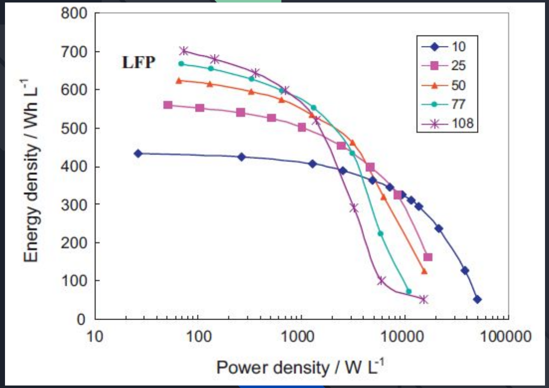
Thick-cathode Li-ion batteries have potential for higher energy density (crucial for EVs)

However, electrochemical performance worsens when using thicker cathodes

Our goal is to understand the cause of these issues on a single-cathode-nanoparticle level







H. Zheng et al. / Electrochimica Acta 71 (2012) 258– 265 263

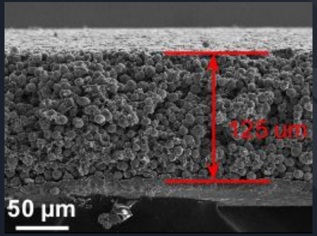
Experimental Details

We used micro-focused X-ray scattering to track the evolution of individual cathode primary particles

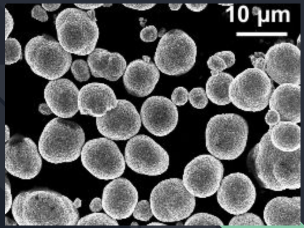
The location of each Bragg peak on our detector is a measure of a primary particle’s lattice parameter

For our project, we aimed to use a CNN to extract peak location with sub-pixel resolution

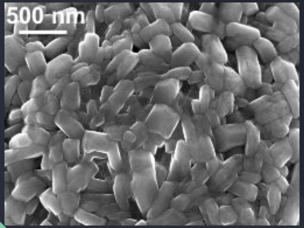
Cathode



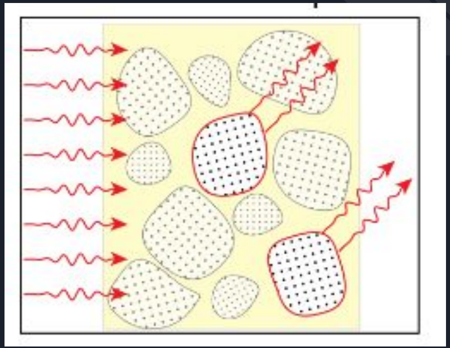
Secondary particles



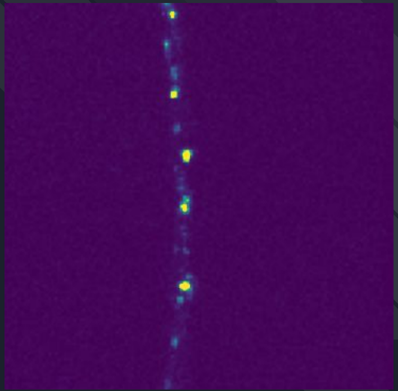
Primary particles



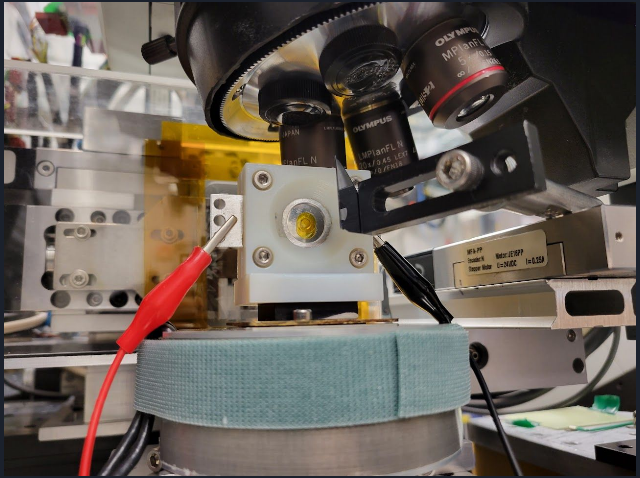
X-ray scattering



Example Data



Experimental Details



Advanced Photon Source, Argonne National Lab

Data

Data frames are a function of charge/discharge state of the battery

The location of each Bragg peak on our detector is a measure of a primary particle’s lattice parameter

For our project, we aimed to use a CNN to extract peak location with sub-pixel resolution

BraggNN

What is BraggNN?

A Deep-learning method for peak position detection

Conventional method involves 2D pseudo-Voigt peak fitting (not ideal for large data)

- BraggNN can determine the center of mass of a diffraction peak with sub-pixel precision.

Advantage of BraggNN:

- It is much faster than conventional 2D pseudo-Voigt peak fitting

  ~200 times faster with consumer hardwares and softwares

- It is more accurate than conventional method

  It yields 15% better results on reconstructions using real data

How does BraggNN work?

Key ideas:

Convolutional Neural Network (CNN)

- Our CNN is acting as feature extractors.
- Each CNN kernel is a neuron that learns to extract certain feature.

Fully Connected Neural Network (FCN)

- Take the result from CNN as input, output the (x,y) coordinates.

Architecture will be introduced in the next slide

CNN & FCN Architecture

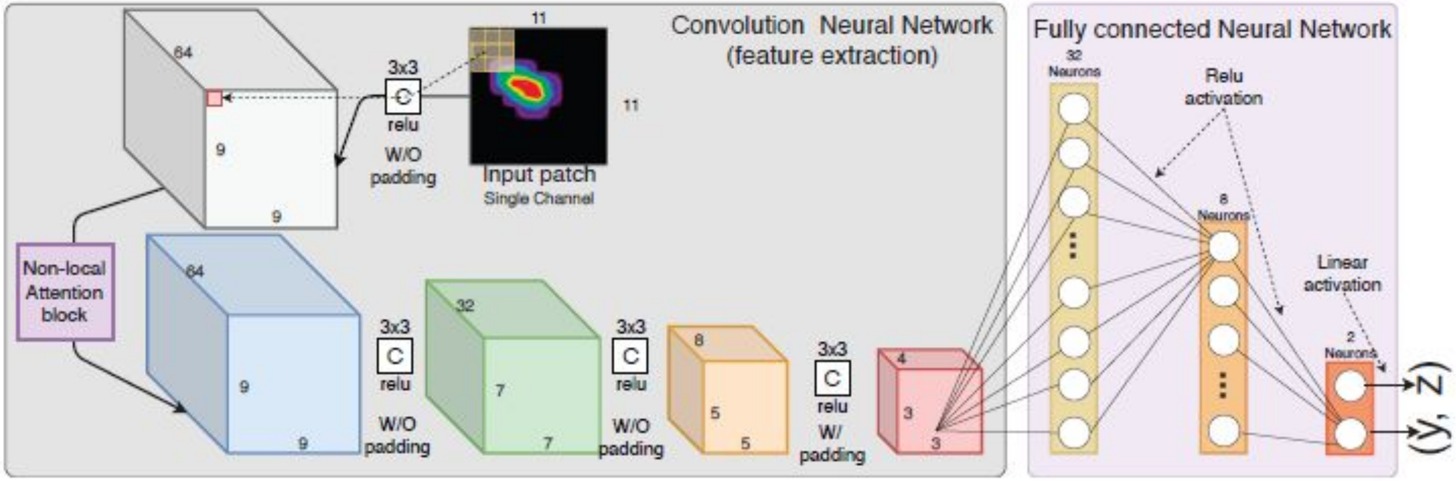


Figure 2: Application of the BraggNN deep neural network to an input patch yields a peak center position (y, z). All convolutions are 2D of size 3×3, with rectifier as activation function. Each fully connected layer, except for the output layer, also has a rectifier activation function.

Liu, Z., Sharma, H., Park, J.-S., Kenesei, P., Miceli, A., Almer, J., Kettimuthu, R. & Foster, I. (2022). BraggNN: fast X-ray Bragg peak analysis using deep learning. IUCrJ, 9, 104–113.

Summary on BraggNN

BraggNN uses fairly common CNN & FC architectures to achieve the desired result. (PyTorch Framework)

Feed-Forward Pass

- Turns the input patch into two floating point numbers
- The model loss is computed between the output and the ground truth

Back propagation

- Compute the gradient of each neuron's weights w.r.t. Loss function using chain rule

Model Training - We train BraggNN model with a collection of input-output pairs.

- Each pair contains the peak patch as input and the peak center position from 2D Voigt fitting as output
- 69,347 Peaks with 80% training and 20% validation and evaluation. .

BraggNN has some pre-trained model which are good building blocks for our project.

Applying BraggNN

- Started by using pre-trained model:
- Find peak roughly and then use BraggNN to precisely determine the location.
- Generate movies to illustrate the peak location.

During our developing:

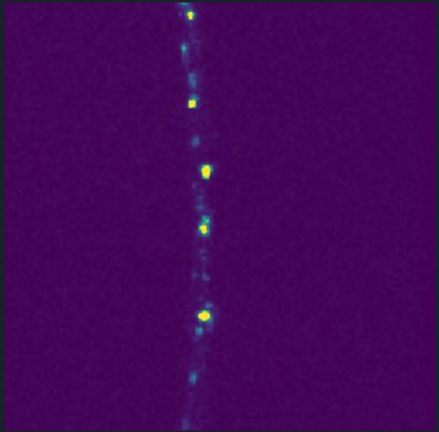
We performed binning and patch filtering to improve model performance

We implemented scoring to better identify the peaks.

```python
def findPeaks(binned_image, minval = 5, minscore = 10):
    '''Function that uses findpeaks module to roughly locate peaks in detector image

    Arguments:
    binned_image - 2D array w/ detector image
    minval - minimum value below which background is set to 0 (default 20)

    Returns:
    x, y - arrays of coordinates for peaks'''

    # Initialize
    fp = findpeaks(method='topology',verbose = 1)
    #Mask off any values < minval
    mask = binned_image > minval
    X = binned_image * mask
    # Fit topology method on the image
    results = fp.fit(X)
    # fp.plot()
    #Select only peaks from result
    peak_mask = (results['persistence']['peak'].values) & (results['persistence']['score'].values>minscore)
    scores = results['persistence']['score'][peak_mask]
    x = results['persistence']['x'][peak_mask]
    y = results['persistence']['y'][peak_mask]

    return x, y, scores
```

```python
def runBraggNN(binned_image, x, y, scores, plot = True):
    '''Function that takes an entire binned detector image and approximate [x,y] coordinates
        of peaks within that image, and creates 11x11 patches and runs BragNN to determine precise peak location

    Arguments:
    binned_image - 2D array w/ detector image
    x, y - arrays of coordinates for peaks
    plot - whether to plot individual patches and fitted peak locations

    Returns:
    NN_fit - array of precise peak coordinates

    '''
    NN_fit = []
    scores_filtered = []
    for i in range(len(x)):
        x_c, y_c = x[i], y[i]
        try:
            #Define and normalize patch
            patch = binned_image[y_c-5:y_c+6 , x_c-5:x_c+6]
            patch = (patch - patch.min()) / (patch.max()-patch.min())

            #Check that max value within patch is roughly in the center
            x_max, y_max = np.unravel_index(patch.argmax(), patch.shape)

            if (x_max > 3) & (x_max < 7) & (y_max > 3) & (y_max < 7):
                #Run BragNN on patch
                input_tensor = torch.from_numpy(patch[np.newaxis, np.newaxis].astype('float32'))
                with torch.no_grad():
                    pred = model.forward(input_tensor).cpu().numpy()
                pred = pred * 11
                if plot:
                    plt.figure()
                    plt.imshow(patch)
                    plt.scatter(*pred[0], color='k', marker = 'x', s = 50)
                NN_fit.append(pred[0] + [x_c-5, y_c-5])
                scores_filtered.append(scores[i])
        except:
            print('Unable to generate/fit patch')
    return np.array(NN_fit), scores_filtered
```

- findpeaks() roughly identifies peak locations (pixel precision) using topological methods
- For each peak found, we generate an 11x11 patch and use it as input to BraggNN
- BraggNN determines peak location with sub-pixel precision

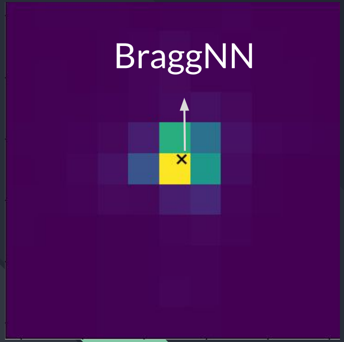


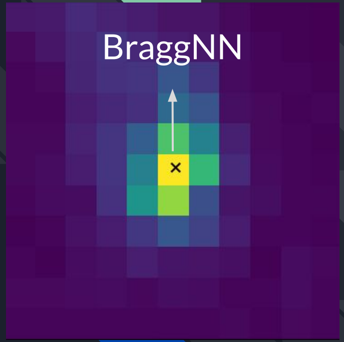
Bin data by (2x2)

Threshold low pixel counts to 0

findpeaks

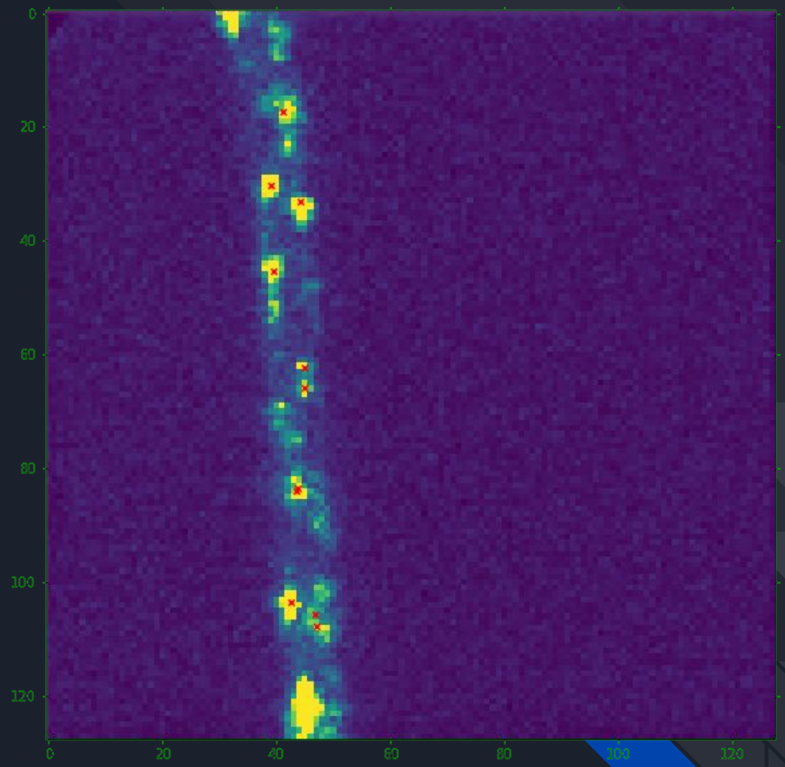
Filter peaks by prominence scoring



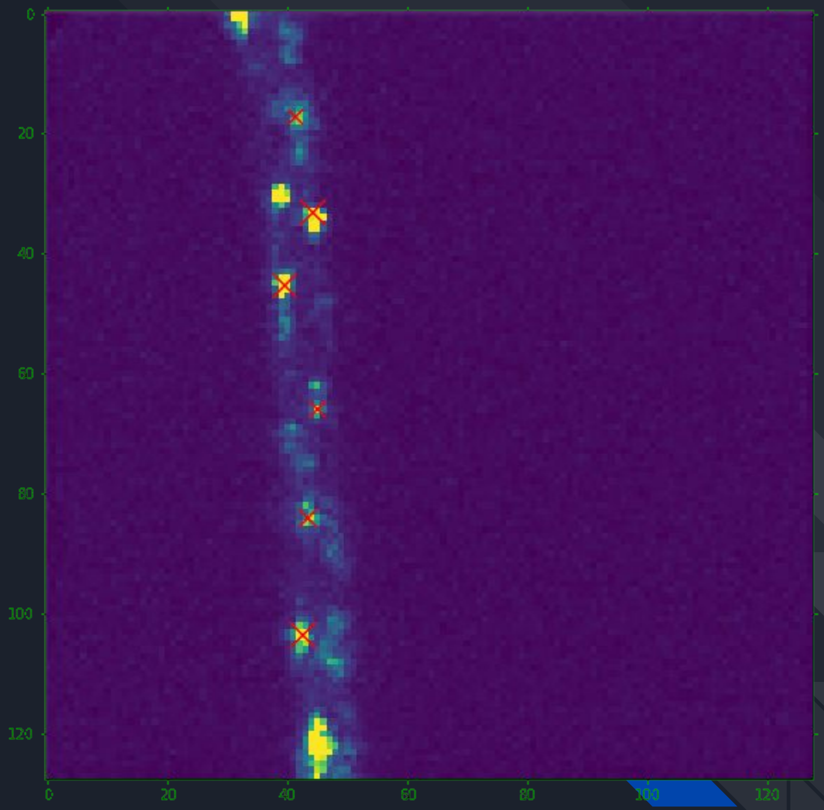


Normalize patch, feed to BraggNN

- Peak location shown as (small) red cross)
- Without careful thresholding and filtering, noise gets picked up as peak

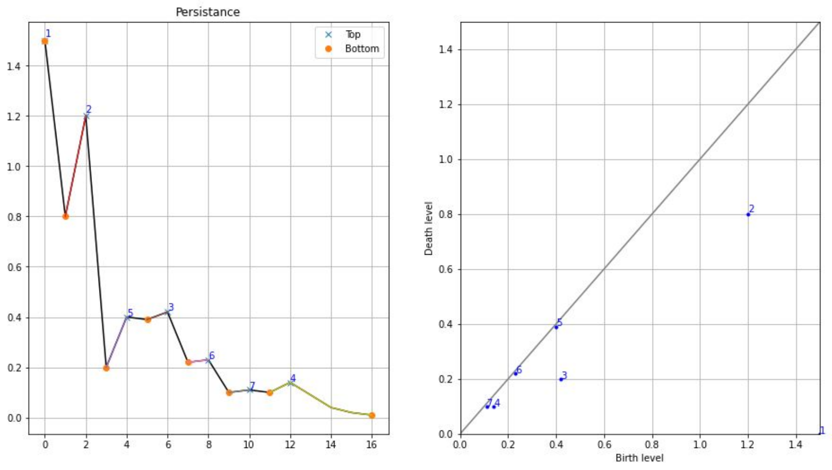


- Refining pixel threshold values and choosing only peaks with prominence >10 (on a scale from 0-255) yielded significantly better results
- Size of cross corresponds to peak prominence



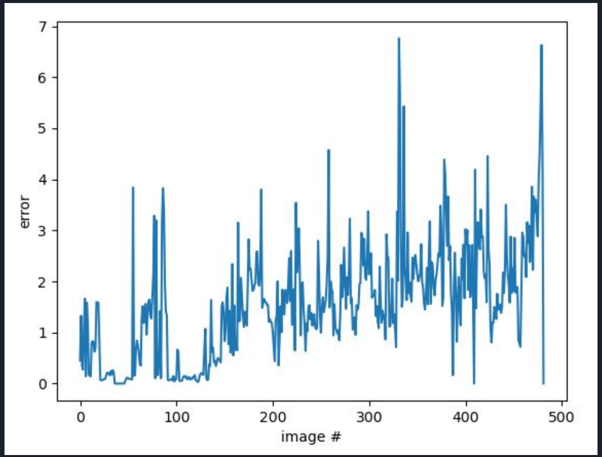
findpeaks

- Uses topological data analysis to detect the peaks
- Persistent Homology
  - Quantitative way of determining the most significant peaks
  - Compares “birth” (peaks) vs “death” (located at saddle points, counts for lower peak)
  - Persistence: difference between birth and death level
  - Sorts by persistence levels

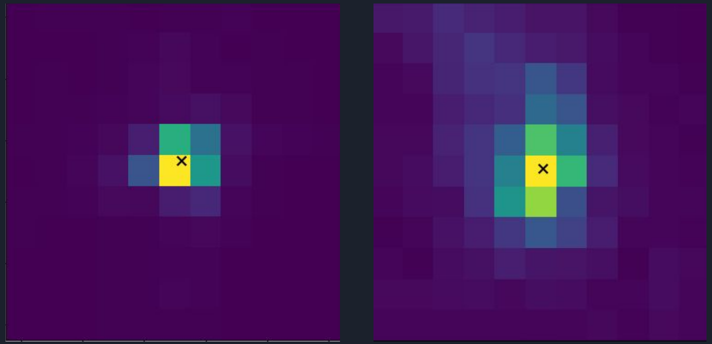


Model Performance

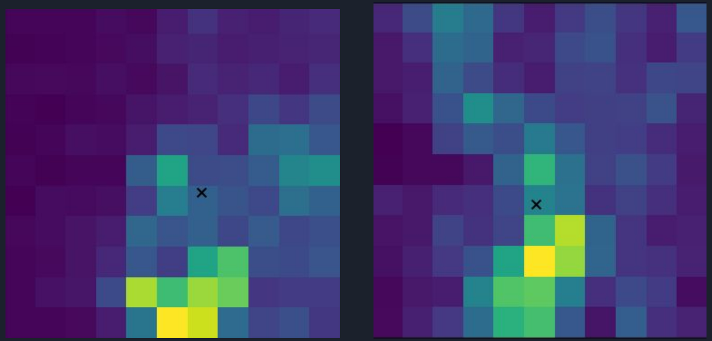
Errors computed by implementing 2D Gaussian fitting to patches and evaluating distance between Gaussian fit and BraggNN fit



Good patches -



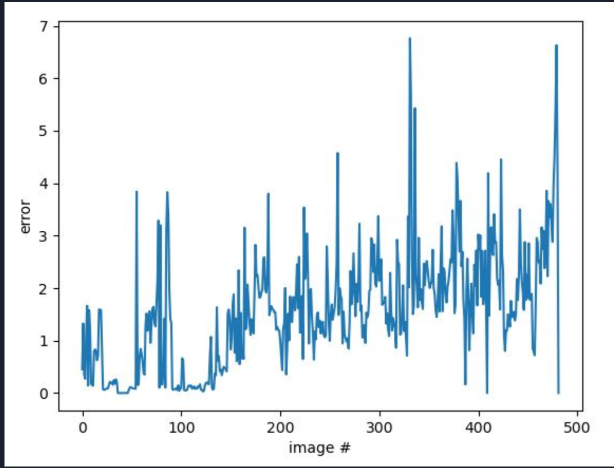
Bad patches -



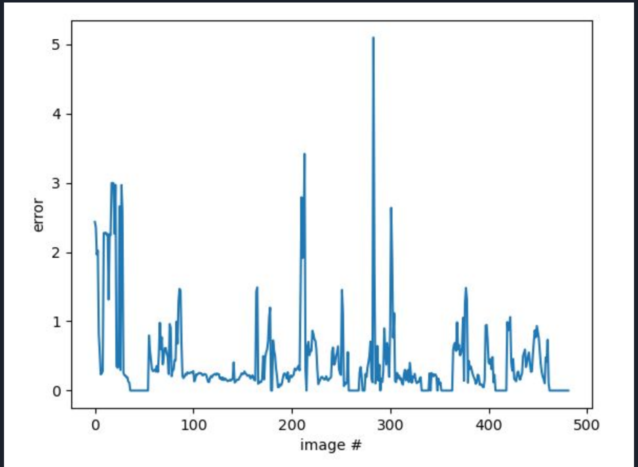
BraggNN is trained only on “Good” patches, while our data has many instances of “Bad” patches

Binning

128 x 128



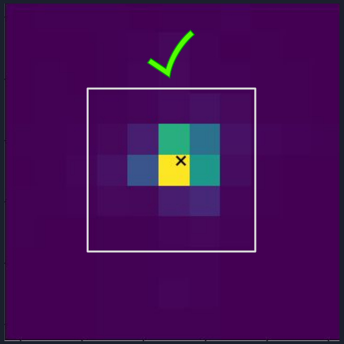
64 x 64

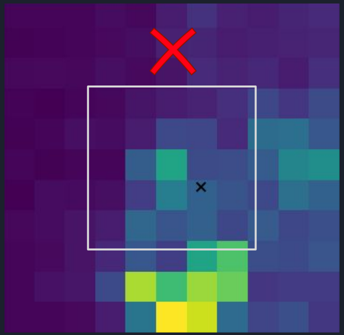


Binning can decrease the number of “bad” patches, but at the cost of resolution

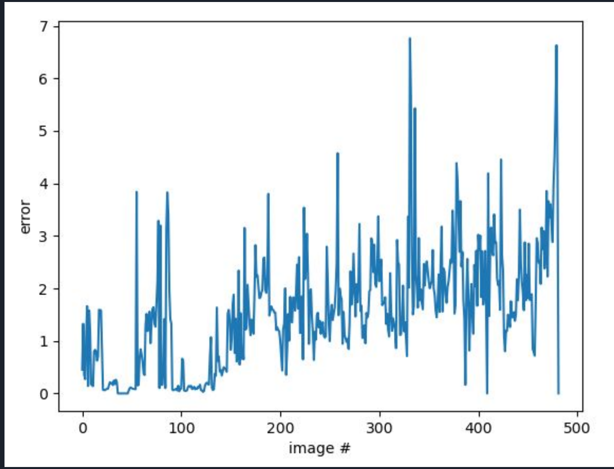
Patch Filtering

Exclude patches where maximum value is farther than 2 pixels from center in both **x** and **y**

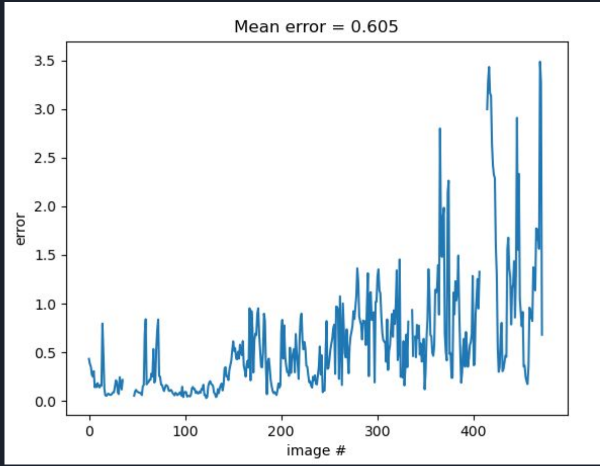




128 x 128, unfiltered



128 x 128, filtered

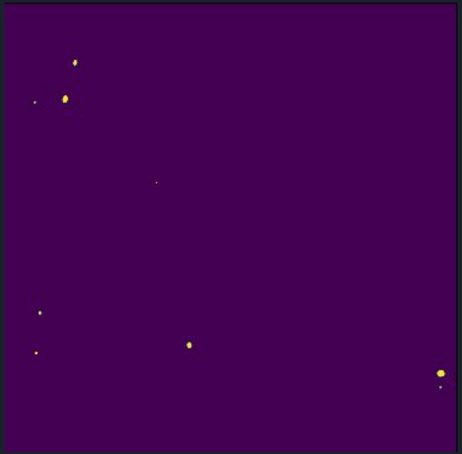


Retraining BraggNN

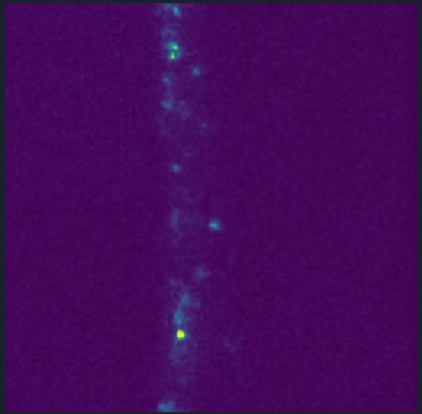
Motivation

- As noted previously, our data is more nebulous compared to the BraggNN training data
  - BraggNN trains of gold particles - sharp, well-defined, spaced peaks
  - Our data is from coin cells - as we charge and discharge, lithium ions disrupt the crystal structure - defects makes the peaks less well-defined

BraggNN Data Sample



Sample from our data



Steps

1. Create the two HDF5 files required to train the model
    a. Frames - 1 file containing the number of frames, and the resolution of the detector images
    b. Peaks - 1 file containing the location of each peak, as well as the frame corresponding to the specific peak
2. Run the model.py code updated with our file names
    a. Error: we consistently ran into the process being killed. As we executed the code, it would buffer, then report “Killed”.
    b. We determined this to be a process error rather than a memory error on our end, since running this on the original training data produced the same result.

```
bstoyche@dsmlp-jupyter-bstoyche:~/teams/group-6/Boyan/BraggNN_Main$ python main.py -mbsz=16 -aug=0
[1679425609.134] loading data into CPU memory, it will take a while ... ...
Killed
```

Re-Write BraggNN

BraggNN, But Using TensorFlow

So far we have been focusing on implementing BraggNN on our own dataset.

Our goals with rewriting BraggNN with our own CNN using TensorFlow were two-fold:

1. Create a CNN to find the diffraction peaks that did trained on our “messier” data, compared to the BraggNN data
2. Use the simpler language of TensorFlow compared to PyTorch to make the model simpler to understand

Result: Here, we ran out of time, and were unable to finish rewriting it with TensorFlow. We will attempt to finish that and submit with our finished code.

Thank you!

Contributions:

Data pre-processing, findpeaks() implementation - Boyan

Pre-trained BraggNN implementation - Andy

Animations of results - Donald

2D Guassian fitting for ground truth comparison - Sean

Efforts towards re-training BraggNN on our dataset - Aditya