Crystallography Peak Detection at CXLS

Eric Everett and Adam Kurth

May 8, 2024

Abstract

This paper explores the application of deep learning techniques at Arizona State University's (ASU) Compact X-ray Light Source (CXLS) to analyze experimental data from various modalities, primarily focusing on X-ray crystallography using the Dectris Eiger 4M detector. Traditional methods of predicting photon energy and sample-detector distance are challenged by dynamic scattering, intrinsic noise, and the CXLS low flux X-ray beam, prompting the need for more advanced solutions. Utilizing the CrystFEL[7] software, we simulate diffraction images for protein 1IC6.pdb across a matrix of nine variable combinations involving photon energies and camera length. Our approach employs convolutional neural networks (CNNs), testing various architectures for binary classification of peak detection and prediction of experimental parameters. The scope of this research wishes to further expand this with modifications in the architecture to accommodate for spectroscopy data, although this is beyond the extent of this manuscript. By integrating different experimental conditions, we anticipate broader applications and improved experimental outcomes.

1 Introduction

1.1 ASU CXFEL Labs

At Arizona State University (ASU), we are at the forefront of developing the world's first Compact X-ray Light Source (CXLS), a scaled-down 10 meter version of the traditionally large X-ray Free-Electron Lasers (XFELs), which typically extend over a kilometer. This revolutionary device emits X-ray pulses at the femtosecond scale and operates at a kilohertz frequency. These X-rays are used to scatter off of a sample where we can collect scatting images and collect data on these samples. Because of the nature of this X-ray beam, samples receive low radiation damage and high resolution scattering images can be collected.

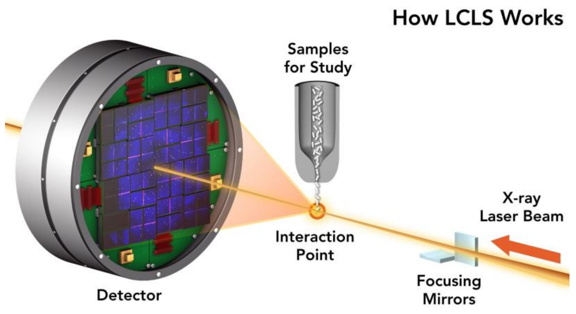


Figure 1: This is an example of the X-ray beam interacting with a sample and the diffraction pattern being detected. This specific illustration is from the Linear Accelerator Coherent Light Source (LCLS). [1]

The ability to scale down our X-ray light source comes from our techniques to generate X-rays. Other sources are so long in part due to the use of magnetic undulator to cause X-ray emission from relativistic electrons. At ASU we align our relativistic electron beam with a high power IR laser in space and time, generating X-rays from a process called Inverse Compton Scattering. The compact nature of the CXLS results in significant cost reductions, making these machines more viable for construction at other facilities. This will push research in fields like material science and pharmaceuticals that greatly benefit from having X-ray light sources for research.

Central to the functionality of the CXLS is the Dectris Eiger 4M detector, known for its high resolution and sensitivity for X-ray detection, which are crucial for capturing intricate diffraction patterns from protein crystals. However, the analysis of these patterns is complex and often labor-intensive, necessitating manual corrections to address data inaccuracies. Additionally, the CXLS has a low flux X-ray beam compared to other XFEL's, which complicates peak detection in both crystallography and spectroscopy images. The presence of scattering noise, largely due to the water content within protein samples, further complicates analyses. These challenges underscore the need for developing advanced methodologies that improve both the precision and efficiency of data processing at the CXLS.

1.2 Project Outline

The goal is this project is to lay the foundations for developing data analysis techniques for the CXLS using deep neural networks. Due to the CXLS still being in the preparation phase for experiments, we will be simulating diffraction data to train our models. This diffraction data is the resulting Bragg peaks from the experiment. In crystallography, Bragg peaks are the result from the constructive interference of

X-rays scattered by the planes of atoms in a crystalline lattice, and they are crucial for determining the crystal structure of materials. Bragg peaks are defined by the following relationship, Bragg's law.

$$n\lambda = 2d\sin(\theta) \tag{1}$$

Where n is the order of the diffracted wave, λ is the wavelength of the X-rays, d is the spacing between the crystal planes in the material, and θ is the angle at which the X-rays strike the crystal planes.

For our simulations, we use the protein designated as 1IC6.pdb, sourced from the RCSB Protein Data Bank [4]. According to the RCSB Protein Data Bank, the *proteinase K from Tritirachium album limber* is characterized by a resolution of 0.98 Å. This enzyme is part of the tetragonal crystal system and belongs to the space group $P4_32_12$[4]. The unit cell dimensions are defined as $a = b = 58.3$ Å, $c = 63.6$ Å, with the point group being $4/mmm$, or 422 [4]. This detailed structural information is fundamental to our analysis and facilitates a more precise understanding of the protein's diffraction patterns. Below is a visual representation of the tetragonal crystal system[3].

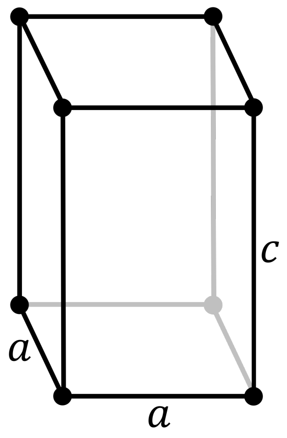


Figure 2: Tetragonal Crystal System.

Using the X-ray diffraction images, we want to be able to determine attributes of 1IC6.pdb. However, before we can do that, we need to use these images to determine attributes of the experimental system. The three attributes we want to determine in this project from our detraction data is are there Bragg peaks present or is it just noise, what the incident photon energy is, and what the camera length is. In addition, we want to be able to detect where Bragg peaks are located. Due to the low photon flux from CXLS, especially compared to other X-ray light sources, the scattered Bragg peaks will appear much closer to the noise in the image, caused by water present in the proteins.

The incident photon energy greatly changes the diffraction data because of the relationship between the wavelength of the incident light and its energy. We can describe this relationship by a form of the Plank-Einstein relation.

$$E = \frac{\hbar\, c}{\lambda} \tag{2}$$

Where E is the energy, $\hbar$ is Planck's constant, c is the speed of light, and λ is the wavelength. We can see the dependence on λ between the relationship with energy and with Bragg Law.

The camera length is another factor that will greatly change the diffraction data. As the camera length grows, there will be an increase angular spread from the scattered X-rays. An increased camera length can increase the spacial resolution of the data by spreading the signal over a greater area, however, this will also result in a weaker signal. Understanding these two parameters is a critical first step for data analysis.

2 Methods

2.1 Data Simulation

To contextualize the methodology employed in our simulations, it is pertinent to discuss the data format utilized—HDF5 (Hierarchical Data Format version 5) [2]. This format is particularly favored in the field of crystallography for its capability to efficiently handle and store large datasets, such as those typically generated during crystallographic experiments [6]. One of the notable features of HDF5 is its support for storing a flexible number of individual snapshots within a single file [6]. This capability is essential for capturing multiple images under varying experimental conditions without the need to switch files, significantly streamlining the analysis process [6].

Moreover, HDF5 files facilitate the clear visualization of Bragg peaks within these snapshots. Bragg peaks are crucial for determining the atomic structure of crystals, as they reflect the positions and intensities of diffracted X-rays. The ability to observe and analyze these peaks across various snapshots enhances our capacity to discern changes and similarities in crystal structures under different experimental conditions. The attributes of HDF5 make it an exemplary choice for crystallography, where precision, efficiency, and meticulous data management are paramount.

Following the selection of the appropriate data format, we proceeded to simulate the diffraction patterns using the `pattern_sim` module of the CrystFEL software suite [8]. These simulations are designed to closely replicate the scattering patterns characteristic of real experimental data, providing a solid foundation for training our deep learning models. An example of the configuration script used for our simulations is depicted in Figure 1, illustrating the setup and parameters that guide our simulation process. This strategic approach not only ensures the accuracy of our simulations but also enhances the effectiveness of the subsequent analysis performed by our deep learning models.

```bash
#!/bin/bash

# Global configurations
NAME="$1"                                       # Experiment or job prefix
TASKS="$2"                                      # Number of tasks to request
    for each job
PHOTON_ENERGY="$"6000                           # Photon energy input
# ...
# pattern_sim specifications
GEOM="Eiger4M.geom"                             # Geometry file
CRYSTAL="1IC6.cell"                             # Crystal file
INPUT="1IC6.pdb.hkl"                            # Constant HKL input file
POINT_GROUP="4/mmm"
CRYSTAL_SIZE_MIN=10000
CRYSTAL_SIZE_MAX=10000
SPECTRUM="tophat"
SAMPLING=7
BANDWIDTH=0.01
N_PHOTONS=3e8
BEAM_RADIUS=5e-6
NUMBER_OF_PATTERNS=10000
# ...
```

Figure 3: Example of BASH script used with `pattern_sim` [8] to simulate diffraction data.

The first fixed parameters in our simulation are the detector geometry of the Dectris Eiger 4M, which produces a 2069×2163 image. Additionally, we have fixed parameters for the protein and its 10 micro crystal environment, as well as for the incident X-ray beam, including photon flux and beam radius. We create 9 total datasets of simulated data by varying the previously mention parameters photon energy and camera length. Table 1 below outlines these datasets.

Table 1: Datasets generated by varying camera length and photon energy.

Dataset	Camera Length (m)	Photon Energy (keV)
01	0.15	6
02	0.15	7
03	0.15	8
04	0.25	6
05	0.25	7
06	0.25	8
07	0.35	6
08	0.35	7
09	0.35	8

All of the resulting images from pattern_sim are pure signal from the diffraction simulations, and will call these *peak* images.

To create the realistic dataset, we must generate the water background images which introduce noise to each respective dataset. These are the diffraction patterns caused by the water within the proteins that make analysis of Bragg peaks difficult. To simulate these water background images we use a software developed by Kirian Lab, `reborn`, which is a Python package for X-ray diffraction simulation and analysis under the Born approximation [5]. Again, like with peak images, we use the Dectris Eiger 4M detector geometry, and very the incident photon energy and camera length. Unlike with peaks data, we only simulate 1 water background image for all 9 data sets. This is because we expect the water background noise to be fairly consistent from shot to shot. We will refer to these images as *water-background* images.

As we see here, the photon energy is held constant at 6keV, while the camera length is 0.15m in Figure 4 and 0.35 in Figure 5.

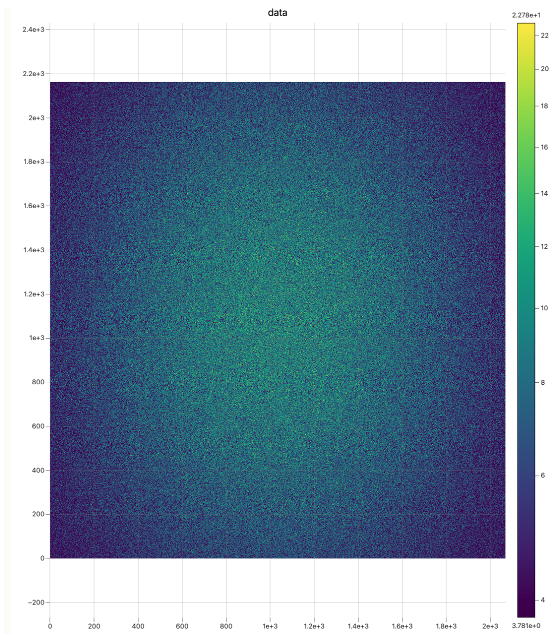


Figure 4: Camera length of 0.15

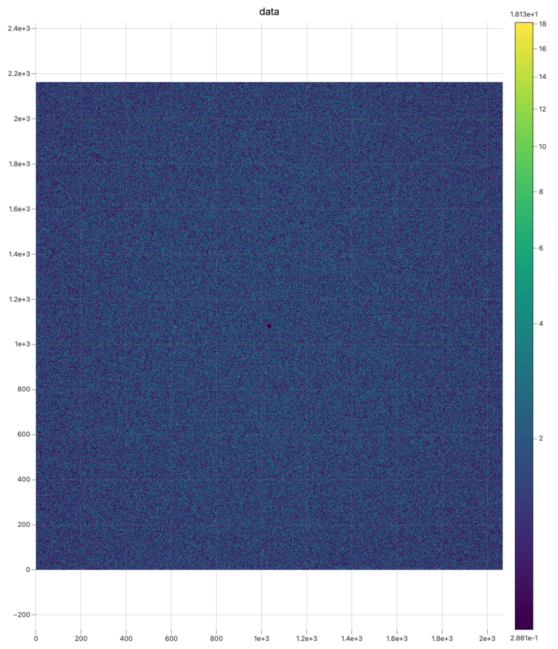


Figure 5: Camera length of 0.35

To create the realistic noisy images from the simulated peak and water-background images. We simply overlayed the water-background images and peak images with the same parameter combination; that is, the same photon energy and camera length values. We will refer to these as *overlay* images.

2.2 Data Management

In handling all of the generated images with different parameter combinations, file management was an important part of the development of this project. Figure 6 shows the base `images/` directory and the structure that is used for the organization of these images. Notably, the `labels` directory holds the binary heatmaps to predict Bragg peak location. Its purpose is to predict Bragg peaks upon testing. However, in the time frame of this project we did not have the time necessary to develop this feature, and it will be discuss later what we used for our labels for the different classification problems.

```
images/
├── peaks/
│   ├── 01/
│   │   ├── 01_6keV_clen01_000001.h5
│   │   └── ...
│   ├── 02/
│   │   ├── 02_7keV_clen01_000001.h5
│   │   └── ...
│   ├── ...
│   └── 09/
│       ├── 09_8keV_clen03_000001.h5
│       └── ...
├── labels/
├── peaks_water_overlay/
└── water/
```

Figure 6: Directory structure for images.

Multiple Python scripts were developed to create a robust file management system. These files are `process_directory.py` (Figure 7), `process.py` (Figure 8), `path.py` (Figure 9), `data.py` (Figure 10). The development of a robust file management system is fundamental to this project, especially as it continues to scale with future development.

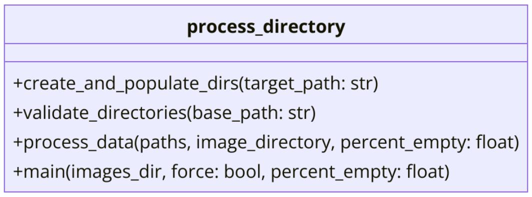

process_directory
+create_and_populate_dirs(target_path: str)
+validate_directories(base_path: str)
+process_data(paths, image_directory, percent_empty: float)
+main(images_dir, force: bool, percent_empty: float)

Figure 7: process_directory.py UML diagram

Upon running `process_directory.py`, the script calls classes from `process.py`, `path.py`. Namely, the `Processor` class which is responsible for generating the overlay images, and the `PathManager` class which keeps track of all the generated files, by selecting the datasets. But `process_directory.py` takes the peaks data (`images/peaks`) and casting every pixel into a binary value (0 or 1) based on a threshold. It's important to note that since the peak images were only signal, this threshold was arbitrary because everything diffracted in the simulation were photons. The script then saves the labeled images in `images/labels` which will be used later after the scope of this course. This script also takes the water-background image (`images/water`) of the respective dataset, and adds the two matrices from the peak image and water-background image. These overlay images are saving in `mages/peaks_water_overlay`.

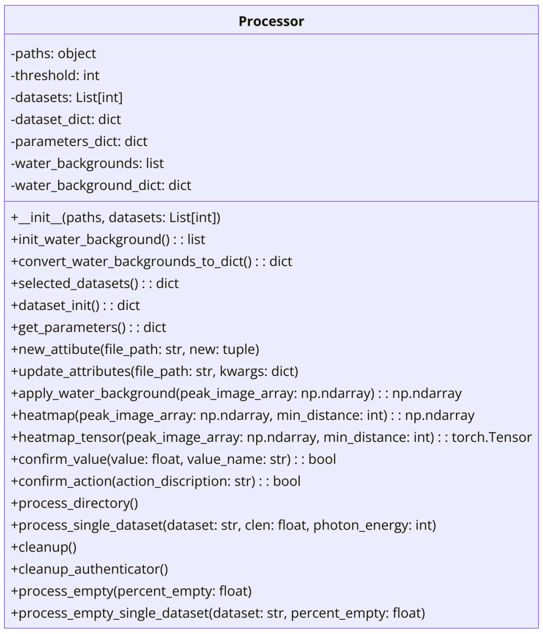
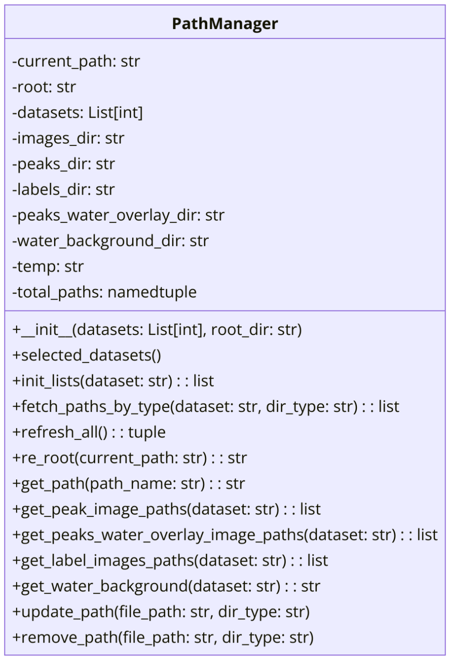

Processor
-paths: object
-threshold: int
-datasets: List[int]
-dataset_dict: dict
-parameters_dict: dict
-water_backgrounds: list
-water_background_dict: dict
+__init__(paths, datasets: List[int])
+init_water_background() : : list
+convert_water_backgrounds_to_dict() : : dict
+selected_datasets() : : dict
+dataset_init() : : dict
+get_parameters() : : dict
+new_attibute(file_path: str, new: tuple)
+update_attributes(file_path: str, kwargs: dict)
+apply_water_background(peak_image_array: np.ndarray) : : np.ndarray
+heatmap(peak_image_array: np.ndarray, min_distance: int) : : np.ndarray
+heatmap_tensor(peak_image_array: np.ndarray, min_distance: int) : : torch.Tensor
+confirm_value(value: float, value_name: str) : : bool
+confirm_action(action_discription: str) : : bool
+process_directory()
+process_single_dataset(dataset: str, clen: float, photon_energy: int)
+cleanup()
+cleanup_authenticator()
+process_empty(percent_empty: float)
+process_empty_single_dataset(dataset: str, percent_empty: float)

Figure 8: process.py UML diagram

With adequate training and testing in mind, the incorporation of ”empty” images are necessary into the datasets. This is easily done by adding an argument that takes some input percent, say 50% of the data in `images/peaks`, (say 50 images from the original 100) then adds that number of image back into the directories (to get 150). `images/peaks` and `images/labels` both will receive simply 0 matrices, `images/peak_water_overlay` will receive the water-background image of the respective dataset. Since we will be testing with the overlay images, its important for peak detection to distinguish between images with Bragg peaks, and without.

As mentioned the `Processor` class in `process.py`, works with `process_directory.py`. In addition to generating new overlay images, and ”empty” images, we also incorporate a validation check. This check ensures all the images have correct attributes assigned to them. This involves; the presence of Bragg peaks, the camera length, and photon energy. We prompt the user to select the datasets to be used for training and testing, in the `select_datasets` function. This simply calls every dataset across the directories in `images/` and gets an updated number of images in each directory, and then calls the `get_counts` function to count every selected directory. All of these steps in the script ensures homogeneous data to be used in training and testing.



```
PathManager
-current_path: str
-root: str
-datasets: List[int]
-images_dir: str
-peaks_dir: str
-labels_dir: str
-peaks_water_overlay_dir: str
-water_background_dir: str
-temp: str
-total_paths: namedtuple
+__init__(datasets: List[int], root_dir: str)
+selected_datasets()
+init_lists(dataset: str) : : list
+fetch_paths_by_type(dataset: str, dir_type: str) : : list
+refresh_all() : : tuple
+re_root(current_path: str) : : str
+get_path(path_name: str) : : str
+get_peak_image_paths(dataset: str) : : list
+get_peaks_water_overlay_image_paths(dataset: str) : : list
+get_label_images_paths(dataset: str) : : list
+get_water_background(dataset: str) : : str
+update_path(file_path: str, dir_type: str)
+remove_path(file_path: str, dir_type: str)
```

Figure 9: path.py UML diagram

The next step, is the load this into the PyTorch DataLoader. This functionality is incorporated in the class `DataManager` in `data.py`. This does the standard DataLoader instantiation from `torch.utils.datasets`, where the file paths from `PathManager` (therefore `images/`) are parsed according to their peak, overlay, and labels images. A partition of the data occurs where 80% is used for training, and 20% used for testing.

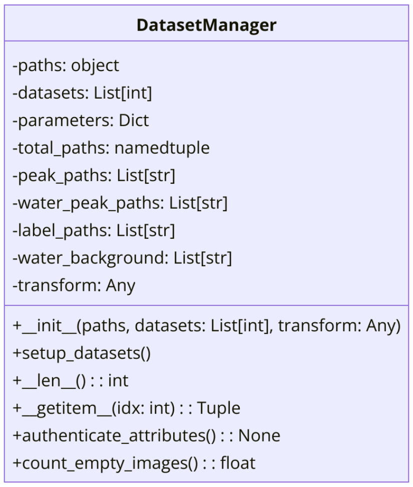

DatasetManager
-paths: object
-datasets: List[int]
-parameters: Dict
-total_paths: namedtuple
-peak_paths: List[str]
-water_peak_paths: List[str]
-label_paths: List[str]
-water_background: List[str]
-transform: Any
+__init__(paths, datasets: List[int], transform: Any)
+setup_datasets()
+__len__() : : int
+__getitem__(idx: int) : : Tuple
+authenticate_attributes() : : None
+count_empty_images() : : float

Figure 10: data.py UML diagram

2.3 Model Architectures

As mention earlier, there are three parameters that we are looking to identify in each diffraction image, are Bragg peaks present, what is the incident photon energy, and what is the camera length. This can be defined as three-classification problem. In addition, we want to be able to locate were Bragg peaks are in a diffraction image. Instead of building one large model that the give an output probability for every combination of the key parameters, three models are used for classification. This approach was agreed to be the best for further development, as these models are only functioning as initial test for classification with simulated data. We will later want to classify photon energy and camera length values more precisely and add models to find different parameters. Due to the nature of having multiple models, we implement a model pipeline for images to pass through to gather information. This will be elaborated on in the latter half of this section.

Each of these attributes we are classifying require different models, labels, optimizers, learning rates, loss functions, and other important parameters. To help keep concise and modular code we created classes in our `eval.py` file for each attribute to take advantage of Pythons object oriented capabilities. This helped the code greatly by only needing to instantiate an attribute we want to classify and pass the object around to relevant code like the training loop, evaluation, and the model pipeline.

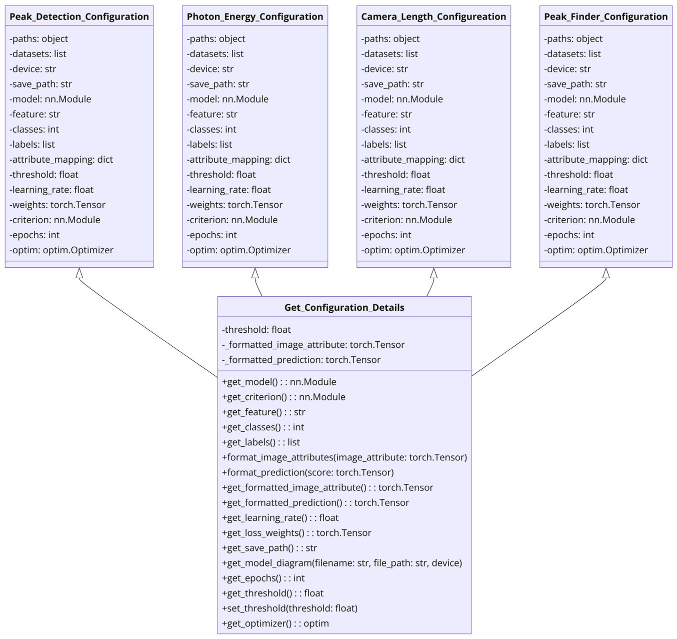


Figure 11: eval.py UML diagram

The first model, as seen in Figure 12 is the first in the pipeline. This being a CNN designed for binary classification; this being peak detection. However, this was by far the most difficult model to get adequate results from due to the low flux nature of the CXLS data. The Bragg peak detection was a very difficult due the closeness in value of the signal and noise. This is why the novel approach we introduce is necessary. Figure 12 highlights the two convolutional layers optimized for extracting varying scales of features from images. The first convolutional layer uses a larger 10×10 kernel to capture broad features, followed by a smaller 3×3 kernel in the second layer for finer details. Each convolutional layer incorporates group normalization with four groups to ensure stable and efficient training by normalizing group activations. After the initial convolutional layer, a max pooling step reduces the spatial dimensions by half, which decreases the computational load and helps mitigate overfitting. The network utilizes the ReLU activation function after each group normalization to introduce non-linearity, enhancing the model's ability to learn intricate patterns. The output from the convolutional layers is dynamically computed to ensure accurate dimensionality for flattening before it reaches the fully connected layer. This layer is crucial as it integrates all the learned features to make the final classification decision. The model's dynamic architecture, including calculated output dimensions and the subsequent flattening of these dimensions, enables efficient processing and robust performance in classifying complex image data.

For binary classification peak detection, It was found that implementing a learning rate scheduler helped the problem converge on a solution faster. The scheduler used is ReduceLROnPlateau. The parameters we used where reducing the learning rate by a factor of 10 every 3 epochs there was not a change in the learning rate at a threshold of 0.1. It was also found that using weights was beneficial to the peak detection problem. This is likely because we are generating empty images based of a percent of the simulated data. Therefore, weights would be helpful due to the imbalanced nature of the dataset. The weight used for binary classification was calculated by dividing the total number of empty images by the number of peak overlay images, giving a weight of about 0.4. The loss function used for binary classification was BCEWithLogits which combines the sigmoid activation function with binary cross entropy. This loss function is very numerically stable and computationally effective. The optimizer used for binary classification is Adam. This optimizer is computationally expensive compared to other optimizers, like stochastic gradient descent, but outcome from using Adam was always superior and thus warranted incorporation.

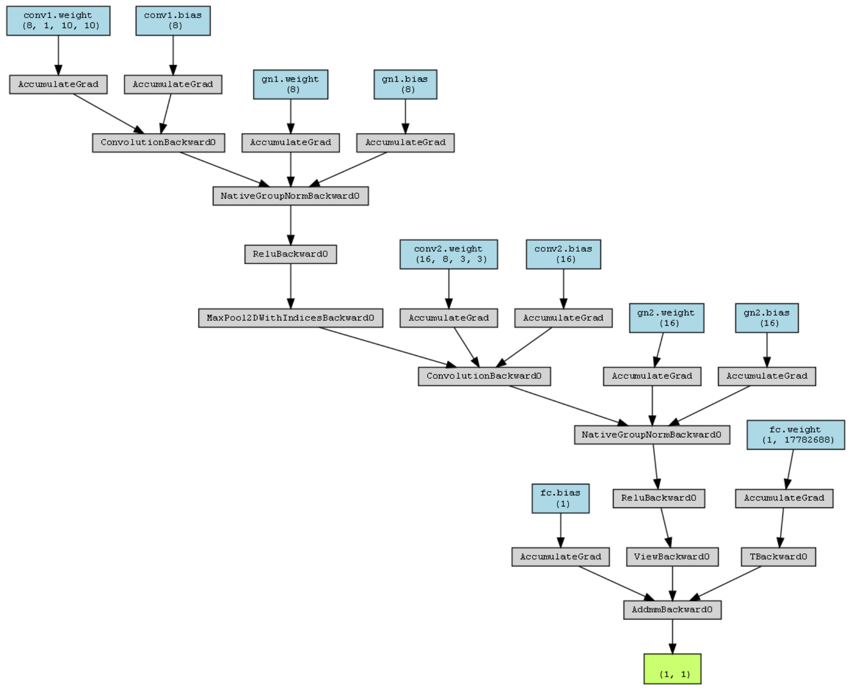


Figure 12: Peak binary classification model made by torchviz.

To increase the complexity of the binary classification problem in a realistic way, we used data were the beam size was not optimized to the sample protein, causing many of the photons to miss the target, lowering the flux even more. Interestingly, we got the best results not by adding more layers to increase complexity, but by using the next model, dual input CNN, where it would train with both overlay images and signal images.

The next model, seen in Figure 13, uses two separate but identical branches. Each branch consists of a convolutional layer with a 5×5 kernel, a stride of 2, and padding of 1, followed by a ReLU activation and a max pooling layer. This design helps in extracting and reducing feature dimensions effectively for both overlay and respective peak images. Output dimensions from both branches are calculated dynamically to ensure consistency before the features are flattened and concatenated. This combined feature set is then fed into a classifier that includes a linear layer with 128 units, a ReLU activation, and a final linear layer that outputs a single decision value. Like for the normal flux classification, we used BCEwithLogits, Adam, and ReduceLROnPlateau.

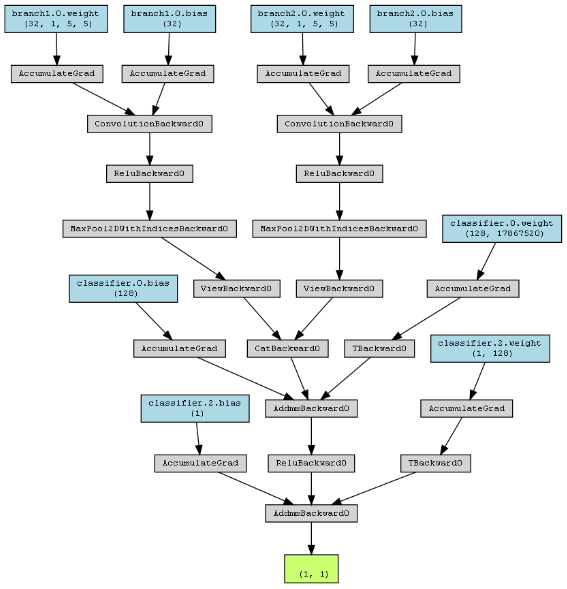


Figure 13: Model of dual input CNN.

The photon energy and camera length are both multi-class classification problems with three output classes for each. The camera length and photon energy effect the overlay and peak images so much that this is a problem that could be addressed with just a linear model. We used stochastic gradient descent (SGD) for these models because they would produce the same results with less computational cost than Adam, and there was no need for a learning rate scheduler. For the loss function, we used cross entropy loss.

The state dictionaries for the models for peak detection (normal flux), photon energy classification, and camera length classification were saved after training. These saved parameters would then be used in the model pipeline. This pipeline is created by instantiating the pipeline class from `pipe.py` and passing in the attribute objects and file paths to the saved parameters. We then would use the `run` function, which takes in an image tensor, and send it through the binary classifier. If peaks were detected, then the image would then be passed to the other multi-class classifiers, for the prediction of camera length and photon energy. At the end of this pipeline, the output value would either be false, ending in termination after peak detection, or return true with the camera length and photon energy predictions. This serves as a prototype for what the final product will act like when being used for the CXLS.

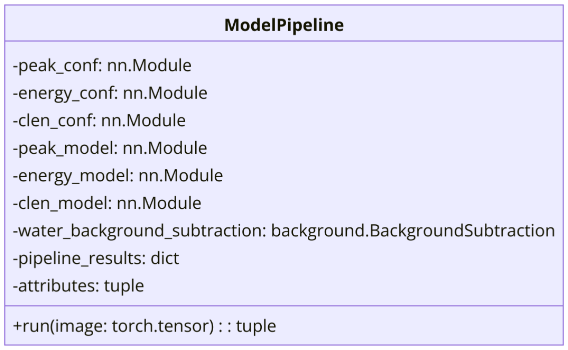


Figure 14: pipe.py UML diagram

The last model we worked on was a model for finding peak locations. This model is by far the most complex model, seen in Figure 24. This model is mentioned after the pipeline because it is not to a satisfactory level yet, and therefore excluded from the pipeline.

This model is a convolutional neural network designed to generate heatmaps from image inputs. It starts with an initial convolutional layer that expands the input to 16 channels, followed by batch normalization and a max pooling layer to reduce dimensionality. A second convolutional layer increases the channel depth to 32, followed by another batch normalization. The network then incorporates advanced attention mechanisms to refine the feature maps; ChannelAttention, which focuses on important features across the channel dimension by processing average and max pooled signals through separate fully connected layers, and SpatialAttention, which focuses on critical spatial information by processing the maximum and average projections of the feature maps.

These attention mechanisms enhance the network's ability to focus on relevant features, both channel-wise and spatially, allowing for more precise heatmap generation. After processing through these attention modules, the feature map is passed through another convolutional layer specifically designed to produce the heatmap output. Finally, the heatmap is upscaled to the original input size using bilinear interpolation and a dropout layer is applied to prevent overfitting.

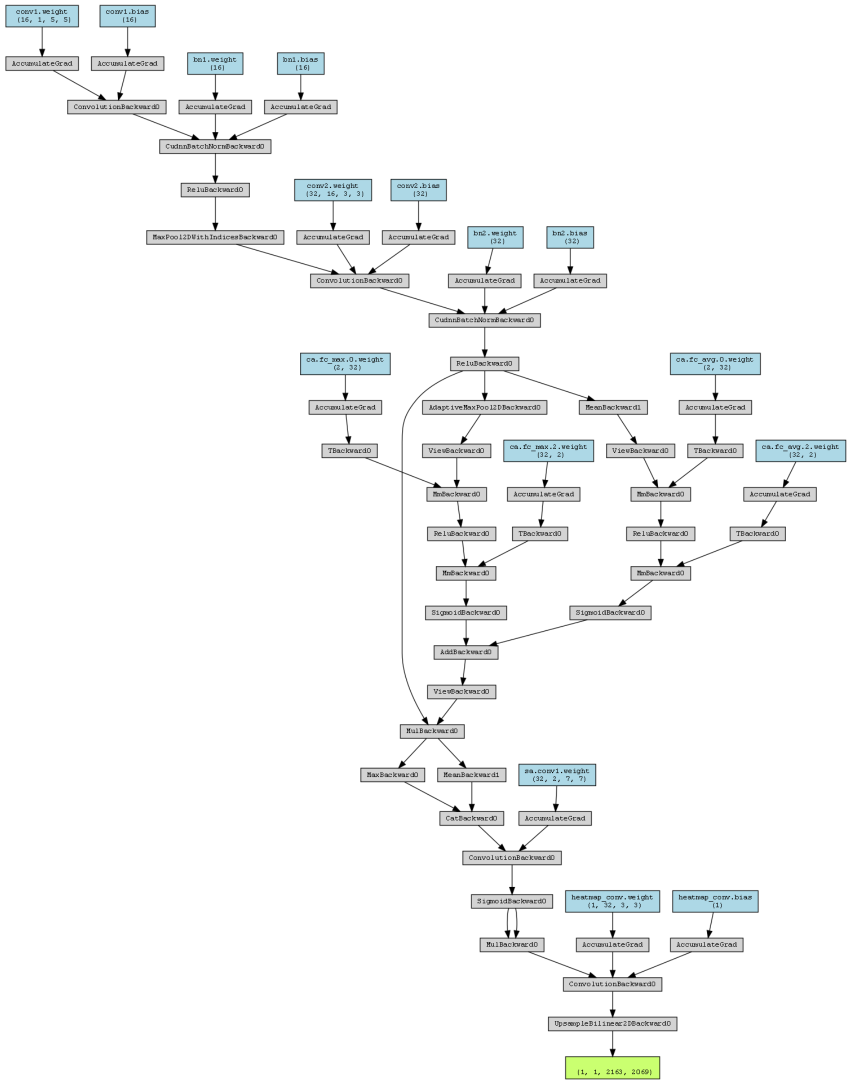


Figure 15: Current status of the peak finder model.

3 Results

The results for both photon energy and camera length classification resulted in a model that could perfectly classify the images. Each model produced a confusion matrix of diagonal ones. As well, by the end of the fifth epoch on each model the train and test accuracy was one and the train and test loss was zero. We can even see in Figure 18 that if we change the data sets being used in training that the model will appropriately respond.

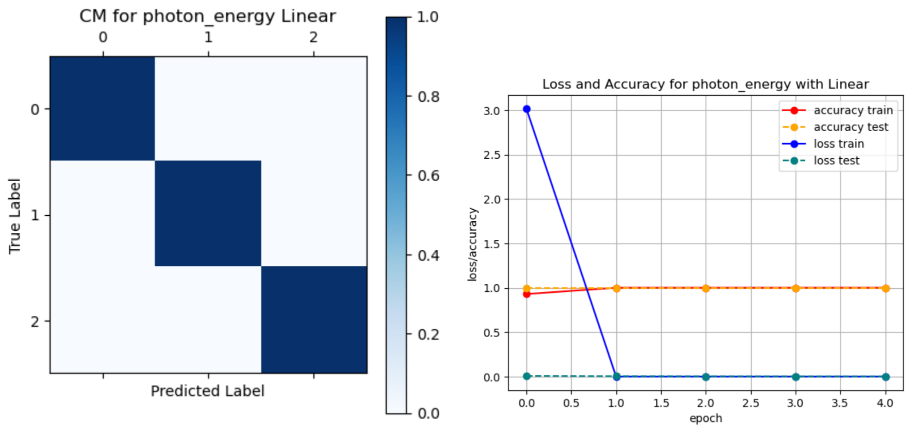


Figure 16: Results from training the linear model for classifying photon energy.

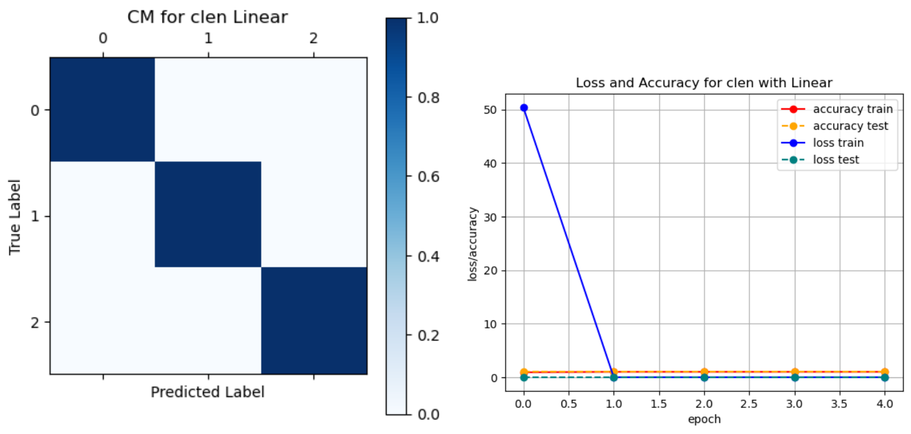


Figure 17: Results from training the linear model for classifying camera length.

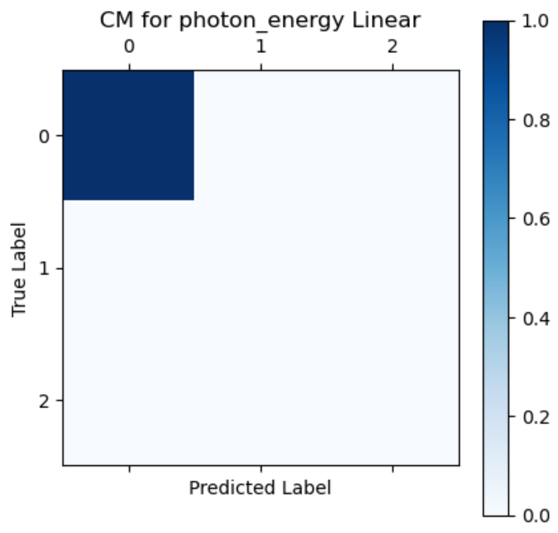


Figure 18: This is an example of selecting specific data sets, here `01` and `02`, and seeing that the model properly classifies the data.

Below is the training for the normal flux binary classification, which was almost perfect. With 30 epochs, the final confusion matrix came out to be,

$$\begin{bmatrix} 1 & 0 \\ 0.025 & 0.975 \end{bmatrix}$$

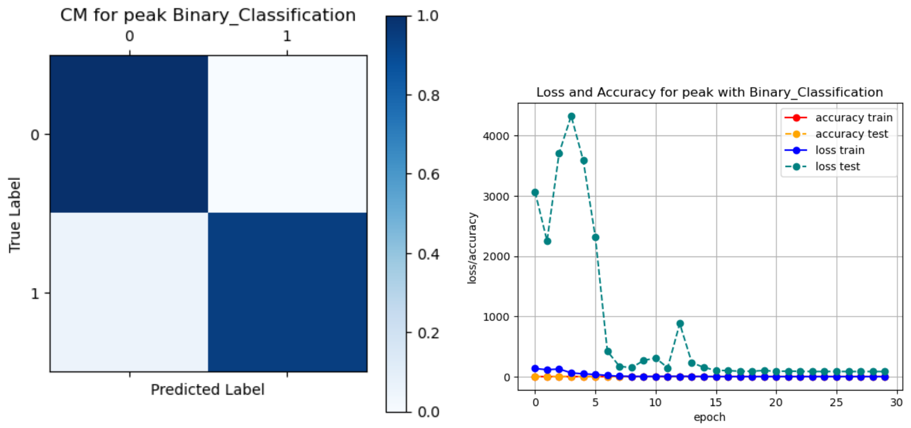


Figure 19: Results from training binary model for classifying normal flux photon peaks.

Below is the comparison base CNN used to justify a more complex architecture. The base CNN is composed of a convolutional layer, the ReLu activation function, and a fully connected layer. Over the same number of epochs, the outcome confusion matrix was

$$\begin{bmatrix} 1 & 0 \\ 0.26 & 0.74 \end{bmatrix}$$

The improvement on peak detection here is critical, as misidentifying 26% of peak data would be detrimental in data analysis. Thus warranting the basic CNN as not suited for our purposes.

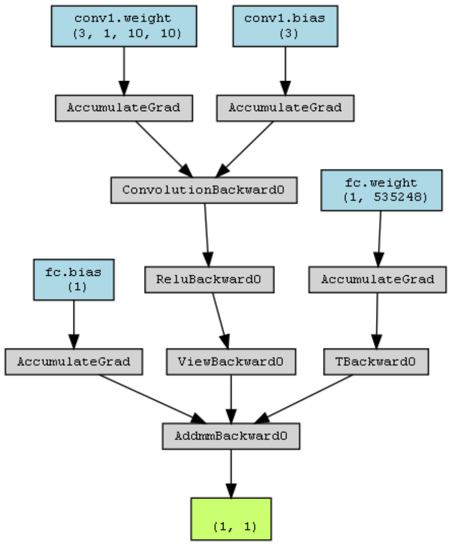


Figure 20: This is the base binary classification model used to comparison made by torchviz.

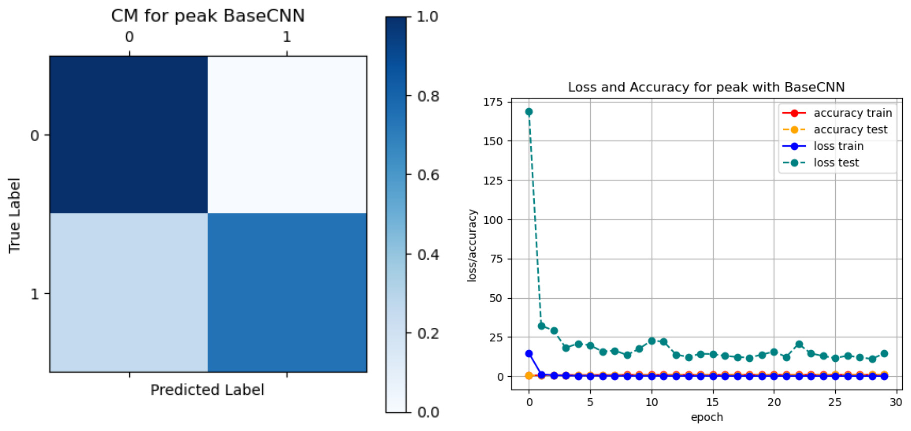
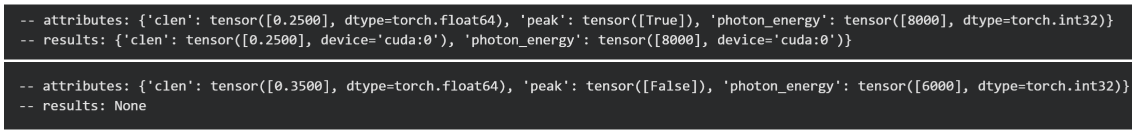


Figure 21: Base comparison model for binary classification.

Using the train models, we loaded that `state_dict` of the models from their respective pt files into the ModelPipeline class. We passed in diffraction images into the run method and it classified the image. Below in Figure 22 are examples of classification of images with and without Bragg peaks.



```
-- attributes: {'clen': tensor([0.2500], dtype=torch.float64), 'peak': tensor([True]), 'photon_energy': tensor([8000], dtype=torch.int32)}
-- results: {'clen': tensor([0.2500], device='cuda:0'), 'photon_energy': tensor([8000], device='cuda:0')}
```

```
-- attributes: {'clen': tensor([0.3500], dtype=torch.float64), 'peak': tensor([False]), 'photon_energy': tensor([6000], dtype=torch.int32)}
-- results: None
```

Figure 22: Example pipeline results.

Below are the results from the extreme low flux from the beam misalignment. These parameters caused classification to be very difficult for the model, producing a confusion matrix of

$$\begin{bmatrix} 0.25 & 0.75 \\ 0 & 1 \end{bmatrix}$$

While the loss and accuracy plot do not show this, the loss and accuracy would plateau for up to 30 epochs, when with the use of a learning rate scheduler. Despite being difficult to classify, this is the only model that would not result in complete classification of either ”peaks” or ”no peaks” (columns in the confusion matrix). Even the model used for normal flux peak detection was producing poor results with this data set in Figure 23.

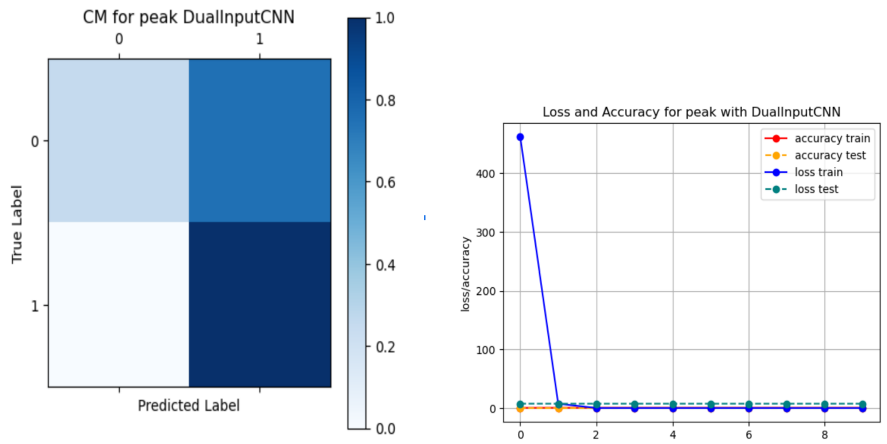


Figure 23: Results from training dual channel model for classifying extreme low flux photon peaks.

In Figure 24 is the results from the peak finder model. The loss and accuracy graph is deceiving in this situation. Since our diffraction images have over 4 million pixels (roughly 2000×2000), and there is about 1 peak pixel per 100,000 noisy pixels, the low loss and high accuracy does not accurately give a good representation of how to model doing in reality. The confusion matrix does however show that the model is over fitting to give that there are never Bragg peaks.

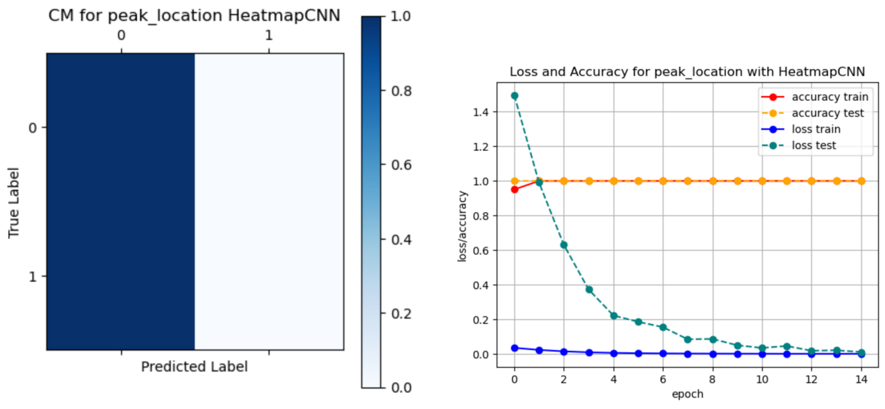


Figure 24: Results from training peak finder model.

4 Discussion

The multi-class classification problems with our parameters ended up being much simpler to classify that previously imaged. We started working with CNNs for this classification and realized that all that was needed was a linear model. This is likely the case because of the values and amount of chosen photon energies and camera lengths to work with. If we had a larger spectrum of photon energies and camera lengths it would likely be much more difficult to classify. The next step in expanding the capabilities of this model now that we know this is an easy task is to generate more data categories, which was too time consuming to reasonably do within the time frame of this project. Ideally we would be able to classify to the nearest keV photon energy and nearest cm camera length.

The binary classification with normal flux was difficult to accomplish because it was mostly in tuning parameters. Creating models with more layers would still only result in less than 75% of correctly identified peak images. We found after creating multiple models and playing with the parameters that the model we choose gave the best results for the lest computational cost. The accuracy seems to plateau at around 95% accuracy for detecting peaks, even with more epochs. This may come down to fine tuning the model even more.

Binary classification for extreme-low flux was incredibly difficult to figure out. Even using methods like transfer learning with models architectures like ResNet50 could not produce results as good as the dual channel input. We also attempted single channel input with both peak and overlay images for training, and testing with only overlay, however this lead to poor results as well. Of course the glaring issue with this implementation would be using this model in a practical sense, due to a channel requiring the need of a signal image. This will need to be explored more because while the lab does not expect to see images this low flux commonly, it would be better if our model could handle a variety of photon fluxes.

The peak detection model was severely over fitting the data to always give the results that there is no Bragg peaks. Even when incorporating a high weight of 100,000 for the loss function, and using a high dropout rate of 0.9, the model eventually starts to guess all zeros over enough epochs. The model also seems to never get any peak identification location correct. We were able to monitor this by watching a printout of the known and guessed peaks. The model seems to learn before 10 epochs that there are only around 10 to 100 peaks present in the image, and it also seems to start to group the guessed peaks in the correct areas. However, it never actually guesses a peak correctly, it over fits before that can happen. We have also played with using a DnCNN (denoising convolutional neural network), however the results of this model did not seem to work as well. This will be the main focus of this project after this class ends. Being able to local the peaks is long term the heart of this project, and will likely be the first thing we tackle during the summer at CXFEL Labs.

In this project data simulation and management was a bottleneck. It was a slow start to get software running for data simulation, and it was a long stretch to get all the correct data we needed. We spent quite some time working with incorrect data until we were able to clear the issues with our P.I. at CXFEL Labs, which lead to us experimenting with the extreme low flux data, since it was already generated. Though this low flux data was not meant to be used at this point, it lead to interesting results for a real issue that was not originally part of this project, but worth mentioning in the report. Issues with file management code would stop the ability to test models for days at a time while troubleshooting. Now that these prospects are in order, we are currently simulating thousands of images to use to use with training, and the file management code is incredibly robust and easy to troubleshoot.

In conclusion, this project has significantly advanced the understanding and application of deep learning techniques in the analysis of X-ray crystallography data for the CXLS at ASU. By successfully integrating CNNs, we have enhanced the accuracy and efficiency of peak detection and the prediction of experimental parameters, demonstrating the potential of advanced computational methods in overcoming the inherent challenges posed by low flux and high noise levels in diffraction data. This research not only serves as a foundational study in applying neural networks to crystallography at CXLS but also sets the stage for future work to expand these techniques to include more complex experimental setups and broader applications in scientific research. Moving forward, it is anticipated that continued refinement of these models and expansion of the data set will yield even more robust tools, capable of transforming the landscape of crystallographic research by significantly reducing the time and resources needed for data analysis.

References

[1] Free-electron lasers reveal detailed architecture of proteins. `https://www6.slac.stanford.edu/sites/default/files/styles/centered/public/lcls_howitworks_boutet.jpg?itok=ShfJEUb_`, 2012. June.

[2] Sustainability of digital formats: Planning for library of congress collections. hdf5, hierarchical data format, version 5. `https://www.loc.gov/preservation/digital/formats/fdd/fdd000229.shtml`, 2022. May 2.

[3] Tetragonal. `https://en.m.wikipedia.org/wiki/File:Tetragonal.svg`, n.d.

[4] RCSB Protein Data Bank. 1ic6: Structure of a serine protease proteinase k from tritirachium album limber at 0.98 Å resolution. RCSB PDB, n.d. `https://www.rcsb.org/structure/1ic6`.

[5] kirianlab. Kirianlab / reborn. GitLab, n.d. `https://gitlab.com/kirianlab/reborn`.

[6] L. A. Wasser. Hierarchical data formats - what is hdf5? NSF NEON — Open Data to Understand our Ecosystems, n.d. `https://www.neonscience.org/resources/learning-hub/tutorials/about-hdf5`.

[7] T. A. White, R. A. Kirian, A. V. Martin, A. Aquila, K. Nass, A. Barty, and H. N. Chapman. Crystfel: a software suite for snapshot serial crystallography. *J. Appl. Cryst.*, 45:335–341, 2012.

[8] T. A. White, R. A. Kirian, A. V. Martin, A. Aquila, K. Nass, A. Barty, and H. N. Chapman. Crystfel: a software suite for snapshot serial crystallography, 2012. `https://www.desy.de/~twhite/crystfel/manual-pattern_sim.html`.