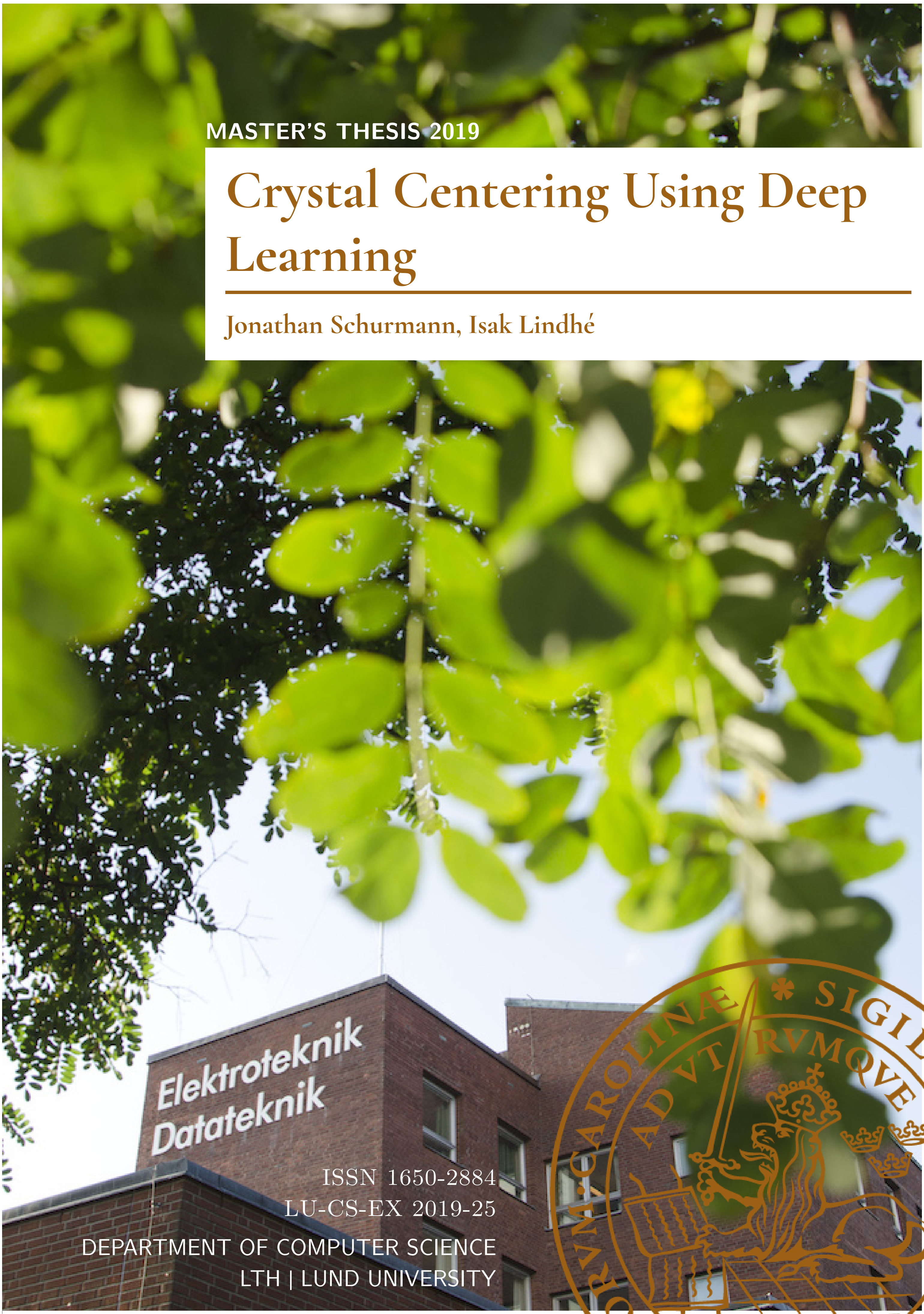


MASTER'S THESIS 2019

Crystal Centering Using Deep Learning

Jonathan Schurmann, Isak Lindhé

ISSN 1650-2884
LU-CS-EX 2019-25

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2019-25

Crystal Centering Using Deep Learning

Jonathan Schurmann, Isak Lindhé

Crystal Centering Using Deep Learning

Jonathan Schurmann
dat12jsc@student.lu.se

Isak Lindhé
dat13ili@student.lu.se

September 16, 2019

Master's thesis work carried out at MAX IV.

Supervisors: Jörn Janneck, jorn.janneck@cs.lth.se
Zdeněk Matěj, zdenek.matej@maxiv.lu.se

Examiner: Flavius Gruian, flavius.gruian@cs.lth.se

Abstract

A problem in X-ray crystallography experiment is to find a good point on the crystal to center in the beam. This problem can be solved by manual aiming or automatically hit the crystal in every possible way and take the best point. When using a more powerful beam, the crystal takes radiation damage after a number of shots which might give unreliable results.

We present a machine learning-based solution by training a neural network with labeled data. This approach does not rely on either brute force nor manual supervision to determine where to aim the beam.

Based on our experiments we can conclude that machine learning is a potential solution to this problem. Our result shows that machine learning and more specifically deep neural networks have the capability to learn where the crystal is. Further research might improve on this and detect a more specific point which is guaranteed to be a good point of where to aim the beam.

Keywords: MSc, Machine learning, X-ray crystallography, Deep learning, Convolutional Neural Networks

Acknowledgements

First and foremost, none of this would be possible without MAX IV and all the amazing people there. MAX IV offered the opportunity for us to write our thesis and have spent time and resources on our work as an ambition for potential use in the future. Thanks to our supervisor at MAX IV, Zdeněk Matěj for all ideas and helping us plan and execute our work.

We would like to thank our supervisor Jörn Janneck at the Faculty of Engineering at Lund University for his irreplaceable and coherent feedback as well as the thought-provoking conversations.

Special thanks to Gustavo Lima for assisting in collecting data and the late evenings in the beamline.

Contents

Chapter 1

Introduction

An artificial neural network borrows inspiration from the biological equivalent found in the field of neuroscience. It consists of neurons connected by synapses that carry out a function when activated. The same idea is applied to the artificial network by connecting layers where each layer is a collection of neurons. Each neuron is activated based on a mathematical condition which in turn might activate neurons in the next layer. When adding a bunch of these layers, the network can learn highly complex functions that might classify or predict a value based on the input. Neural networks have existed for decades but did not take off until recent years due to a lack of computation power and access to large data sets. The reason one might use a neural network is the ability to learn arbitrary features without explicitly programming them.

Machine learning methods are applied here in the domain of crystallography, in particular to the optimization of single crystal X-ray diffraction experiments. This class of experiments is is the most common and well-established method for an experimental determination of a 3D atomic structure of molecules including important biological molecules as proteins. It is an essential component for the development of new pharmaceutical molecules used in medicine.

In such a single crystal diffraction experiment a high-power X-ray beam is fired at some sort of crystal sample. A detector behind the crystal detects the pattern (Figure 1.1) of how the beam is scattered by the crystal. See Figure 1.2. The recorded interference pattern, called diffraction pattern, is then used for the analysis of 3D atomic structure of the crystal. The crystal is usually around $100\mu m$ across and the beam (in our case) has a diameter of $20\mu m$. The crystal is fixed by surface tension in a small metal loop, as seen in Figure 2.2 at page 18.

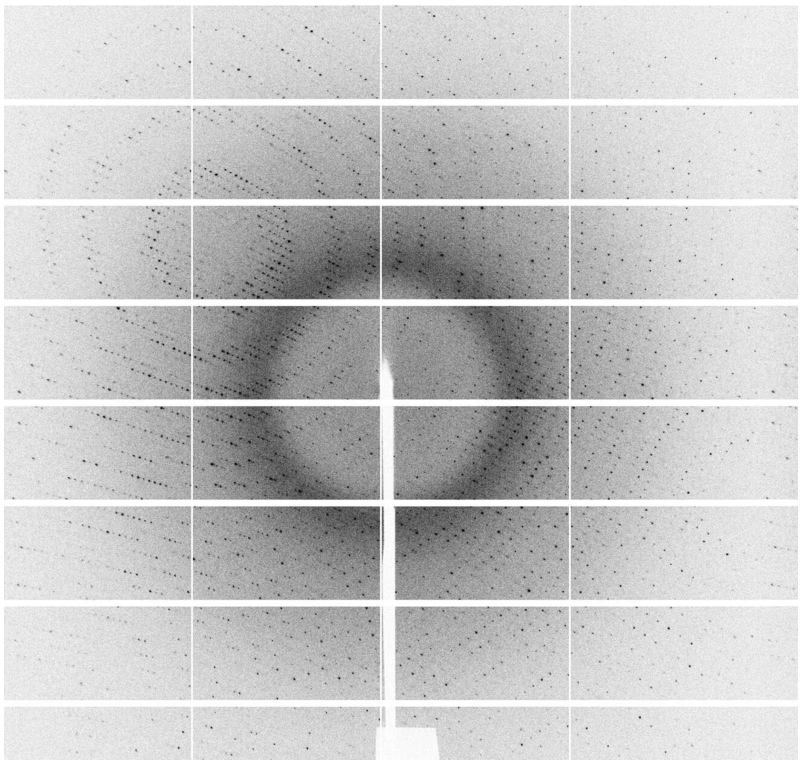


Figure 1.1: An example of a diffraction pattern from a crystal.
© Gustavo Lima

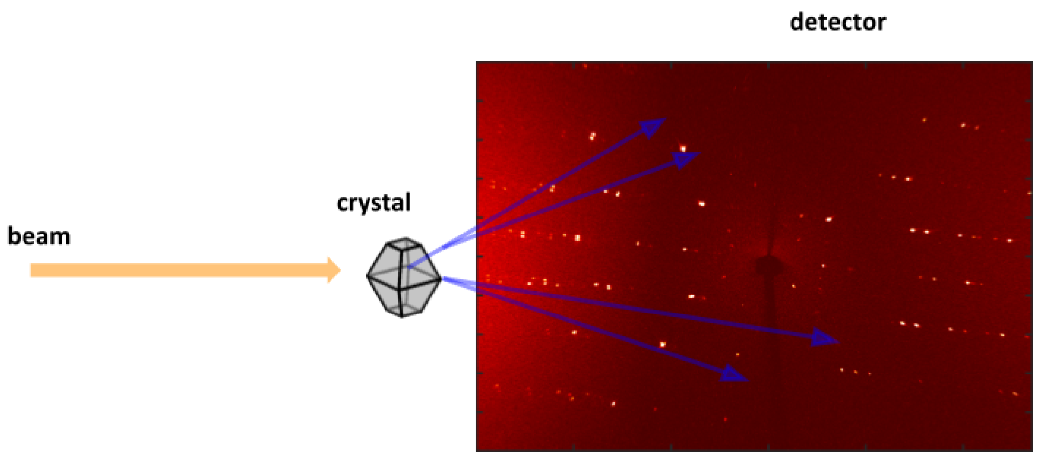


Figure 1.2: Beamline setup at BioMAX. © Zdeněk Matěj

This loop is in turn held by a goniometer [1] and the sample can rotate around 1[1] axis and translates in 3 axis. A goniometer is an instrument which allows the loop to be positioned in a precise position. Liquid nitrogen is constantly being blown on the crystal to keep it cool.

One place where this happens is the BioMAX beamline at the MAX IV laboratory in Lund, Sweden. The light source at MAX IV has much higher energy than any other synchrotron in the world at the time of writing and this is what distinguishes MAX IV for other synchrotrons.

1.1 Problem and Aim of the work

To get a high-quality diffraction from the crystal, we want to aim the beam at a good part of it. The quality of a diffraction pattern can be measured by counting the number of reflections that appear in it. We will call this the reflection count (RC). This is how we determine what is a good or bad part of the crystal. Other methods of measuring the quality exist and this is just one of them. Most synchrotrons simply do this with a 'shotgun approach', which means scanning the X-ray beam over a grid of spatial points overlapping the crystal, then simply picking the point giving the best diffraction signal.

Unfortunately, MAX IV cannot use this approach, as their beam is so intense it can inflict serious radiation damage on the crystal if the 'shotgun approach' is used. They therefore have to center the crystal more carefully to get a good result without so much exposure as to damage the crystal. This is currently done by painstakingly centering each and every crystal manually.

Scientists who do this seem to build up an intuition that might be a good part of the crystal to aim the beam at. This seemed like a good indicator that perhaps an artificial neural network [2] could also learn this skill for recognizing good points on crystals to expose to the beam.

If we can train a model to recognize how well-centered the crystal is in a given image, we can test various points to center on without having to fire the beam by simply cropping the image around the point we want to test.

In Figures 1.3 and 1.4 you can see some examples of images centered at good and bad points. As you can see, the good images have the crystal in the center, where the beam is hitting, while the really bad ones are usually centered on the edge, on the loop or do not even have a crystal.

This thesis aims to examine the possibility to use machine learning and neural network to solve this problem.

[1] 1 axis is used in most experiments at BioMAX. The actual limit is 3 axis.

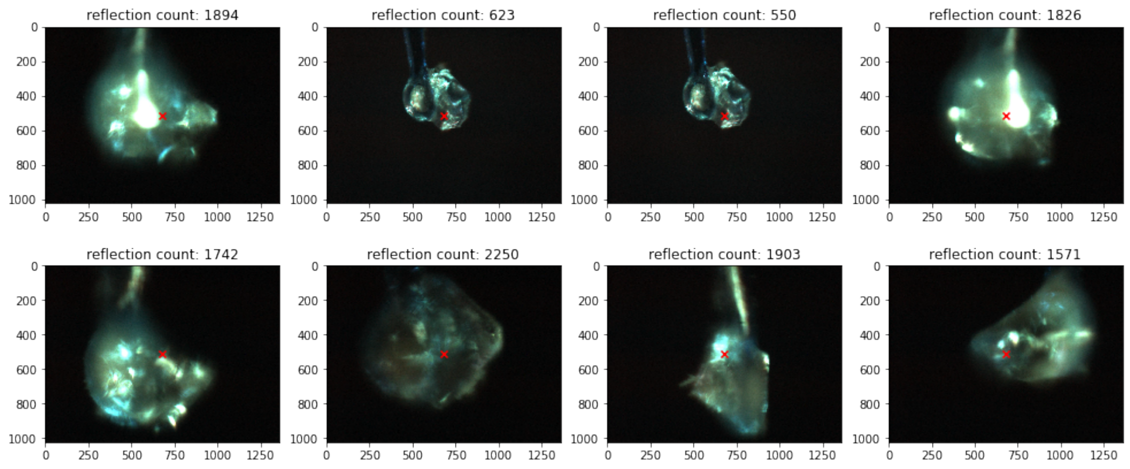


Figure 1.3: Some images with high reflection counts

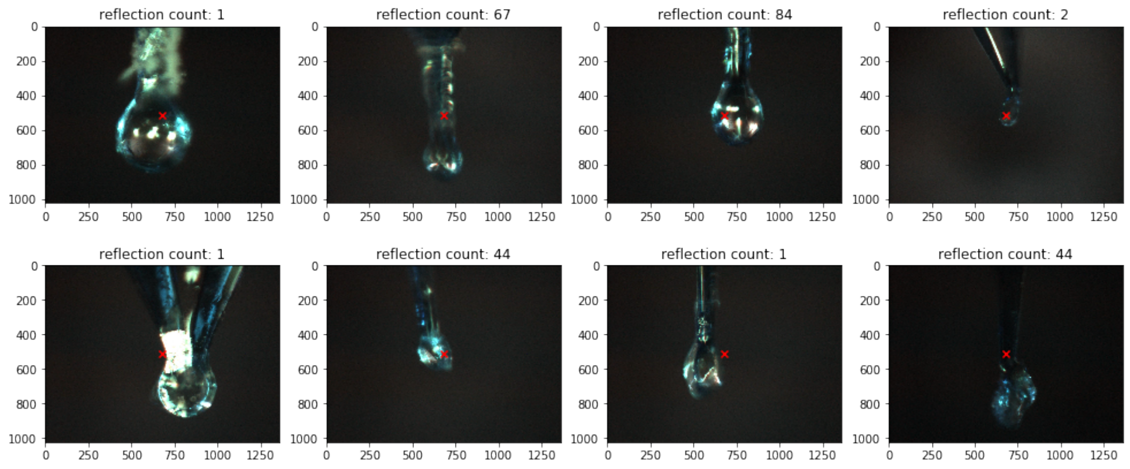


Figure 1.4: Some images with low reflection counts

1.2 Introduction to Convolutional Neural Networks

In order to have some basic understanding of Neural Networks to get the most out of this report, we recommend looking through the lecture notes of the Stanford course: `http://cs231n.github.io/`, specifically these two lessons:

- Neural Networks Part 1: Setting up the Architecture,
  `http://cs231n.github.io/neural-networks-1/`
- Convolutional Neural Networks: Architectures, Convolution / Pooling Layer,
  `http://cs231n.github.io/convolutional-networks/`.

However, we go through some of the basic concepts also here.

1.2.1 Overview

The basic structure of a Neural Network (NN) [3] are neurons arranged in layers. These layers are placed sequentially after each other. See figure 1.6 at page 14 for an example of how the layers are connected.

Each neuron has one or more inputs which is multiplied with some weight w. These weights and inputs are then summed up and added together with a bias [2]. See Figure 1.5 and equation 1.1.

$$\sum_j w_j x_j + b = \mathbf{wx} + b \tag{1.1}$$

Vector $\mathbf{x}$ represents our inputs and j is the number of inputs. Hereafter vector notation will be used.

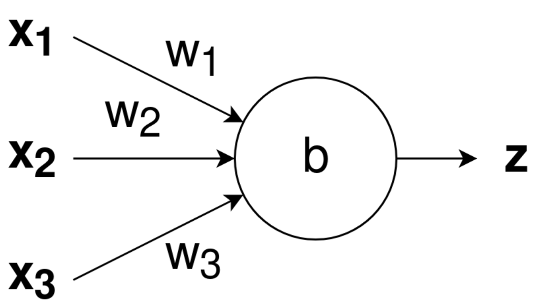


Figure 1.5: Schematic image of a neuron with inputs $\mathbf{x}$, weights $\mathbf{w}$ and bias b. The neuron will produce some output $\mathbf{z}$.

We will use a type of NN called Convolutional Neural Network (CNN). The most common input to a CNN is an image X with a dimensionality of $X \in R^3$ or $X \in R^2$ depending on the number of channels. If color images of size 28×28 are used, the number of inputs will be $28 \cdot 28 \cdot 3$. The output will differ depending on what we are trying to achieve. A single output neuron will be enough for binary classification or regression analysis. For multiclass classification, the number of outputs will be the same as the number of classes if the output is represented as base-10. [4]

Classification is a subset of supervised learning and means that the network will learn a function to assign each data point to a category. Binary classification means that we have two categories while multiclass has three or more categories. Regression analysis means the use of data to predict a continuous value with respect to our learning data.

The characteristic layer in a CNN is the convolutional layer. The intuitive meaning of a convolutional layer is to find patterns in the image. The input to this layer is arranged as a 3-dimensional matrix with dimensions $width \times\ height \times\ channels$. The layer's filters (also called kernels) slide (convolve) over the $width$ and $height$ dimension of the input to determine distinctive features. A feature is a bit of visual information in the image. It's usually something abstract or vague such as an edge. The filter is small spatially (height, width) but fills the entire $channels$ dimension of the input. A typical filter might be 3×3 or 5×5 and 3 deep for color images. Sliding the filter over the image will produce a 2D activation map with activations (real numbers) by taking the dot product between the filter weights and input pixels. Each filter will have a set of weights and therefore represent or symbolize a feature. The weights for a filter are the actual matrix values. [2]

Example. Suppose our input image has a dimension of $28 \times 28 \times 3$ and we are using filter of size 5×5. Each filter in our convolutional layer will have $5 \cdot 5 \cdot 3$ weights + 1 bias parameter = 76.

1.2.2 Arrangement of weights in a convolutional layer

How many weights are there in a convolutional layer and what is the dimension of the output? The dimension is dependent on the hyperparameters of the convolutional layer. The depth is equal to the number of filters. The width and height are dependent on four parameters:

- Input size (W) - Spatial size of the image.
- Filter size (F)- Spatial size of the filter.
- Zero-padding (P) - Padding with zeroes around the input volume.
- Stride (S) - How "far" we move with the filter when we slide it. Stride 1 means that we move the filter one pixel at the time.

The spatial size of the output from the convolution layer is given by equation 1.2.

$$\frac{W - F + 2 \cdot P}{S} + 1 \tag{1.2}$$

Example. Suppose we have an input of $28 \times 28 \times 3$. The convolutional layer has 32 number of filters with a zero-padding of 0, a stride of 1 and a filter size of 5. This means that $W = 28,\ F = 5,\ P = 0,\ S = 1$. The spatial output from the convolutional layer is then $\frac{28-5+2\cdot0}{1} + 1 = 24$.

The number of weights is dependent on the filter size F, the number of channels of the input C, and the number of filters N.

Example. The total number of weights in a convolutional layer is given by equation 1.3.

$$(F \cdot F \cdot C + 1) \cdot N \tag{1.3}$$

We add 1 to each filter to account for the bias parameter.

Given the previous example, the total number of weights is $(5 \cdot 5 \cdot 3 + 1) \cdot 32 = 2432$.

1.2.3 Fully connected layer

The layer which implements the operation specified in equation 1.1 is called fully connected layer.

1.2.4 Pooling layer

Pooling helps the model become slightly invariant to small translations of the image [5]. The function is to reduce the number of parameters by discarding activations. The activations are usually discarded by performing the MAX operation on the input by sliding a window of generally 2×2 with a stride of 2. This drops 75% of the activations and therefore helps with overfitting. Pooling is done on a single slice of the input at the time.

1.2.5 Dropout layer

A layer that reduces overfitting by randomly setting N% of the activations to zero.

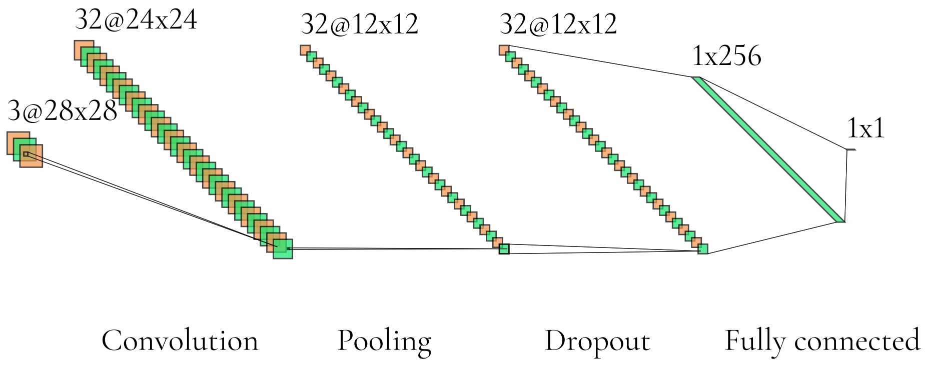


Figure 1.6: An example of a very simple architecture with the described layers.

1.2.6 Loss function

Loss functions are needed to measure compatibility between a prediction and ground truth data. Two loss functions for continuous data are Mean Square Error (MSE) and Mean Absolute Error (MAE). n is the number of predicted data points, $\hat{y}$ is ground-truth value and y is predicted value.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2 \tag{1.4}$$

MSE is useful the error weight should increase exponentially. Being off by 4 should weight more than twice as much than being off by 2. If that's not the case, MAE is better to use.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y-\hat{y}| \tag{1.5}$$

Binary cross-entropy loss may be used for binary classification. [4]

$$H = -(y * log(\hat{y}) + (1-y) * log(1-\hat{y})) \tag{1.6}$$

1.3 Related Work

Convolutional neural networks are often used for classification tasks and in 2012, *Krizhevsky et al. (2012)* [2] used an architecture called AlexNet to classify 1.2 million images into 1000 different classes. AlexNet has ReLU, Max-pooling and dropout layers in the architecture.

AlexNet also uses data augmentation techniques such as horizontally flipping the images to increase the data set. All of these are techniques are used in this thesis.

Using CNN for regression analysis seems to be less common, however. *Shimobaba et al. (2018)* [6] uses CNN for a regression problem. They try to do depth prediction in digital holography to get a precise prediction down to a few millimeters of precision. They use an architecture similar to us and a single neuron as the output layer.

In the crystallography field there exists work trying to classify the outcome of the crystallization process [7]. *Miura et al. (2018)* [8] takes this one step further by first segmenting the crystallization area and then trying to classify crystals in that area only.

There is little to no research in solving the exact problem we have. Some work exists in trying to detect the sample holder and/or the crystal. There are two versions that we are aware of: *lucid2* [9] and *lucid3* [10]. Background and motivation for lucid can be found in [11]. Both of these are using traditional image analysis with OpenCV. They are developed for usage at the European Synchrotron Radiation Facility (ESRF) which has a slightly different setup than MAX IV and does not guarantee us a good result.

Both lucid versions use a laplacian edge detecting algorithm [12]. These work well in constant lightning conditions but they are very fragile to changes in the lighting environment. Modifications are therefore needed before one can expect the same result in BioMAX as in ESRF with lucid. One would also need to be aware of changes to the lightning at the experimental setup and modify lucid to continue using it. Our method will not be as affected by the lightning since diverse training data will help neutralize this.

1.4 Work Distribution

Most of the work has been carried out in close collaboration between the authors. The only exception is that when Approach 1 looked like it might be a lost cause, Jonathan Schurmann started working on Approach 2 while Isak Lindhé kept tweaking Approach 1 until we got results we could use.

The writing of this report was divided similarly, with Isak Lindhé being mainly responsible for Approach I, and Jonathan Schurmann being mainly responsible for Approach II, as well as the Introduction and conclusion chapters.

Chapter 2

Experiments

We will now describe two different approaches for solving the problem under consideration in this project. The reason for multiple approaches came naturally during the project timeline when the first approach was chosen was not giving reliable results in the initial stage of the project. The report on the first approach is also describing the heads on-solution to solving the problem. When that approach didn't turn out as expected, we started with the second approach as a way to troubleshoot what could be wrong. The approaches are not necessarily in chronological order see Figure 2.1.

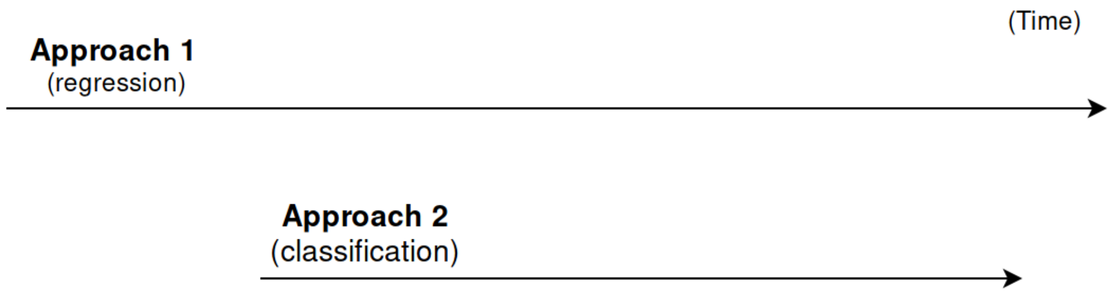


Figure 2.1: Timeline over the execution of approaches.

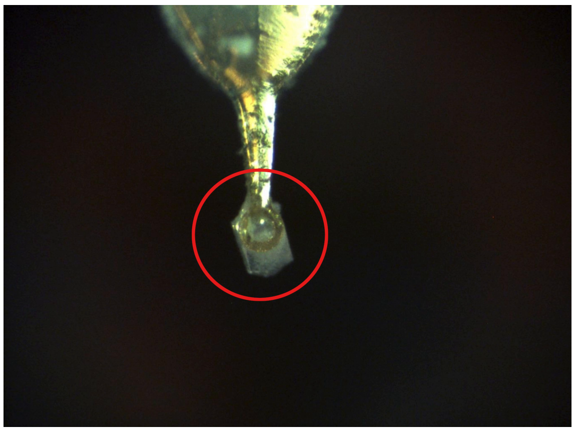


Figure 2.2: A loop holding a crystal, seen from the perspective of the beam source.

2.1 Approach I: Regression analysis

2.1.1 Overview

Here we attempt to teach a CNN to estimate the reflection count associated with a particular image. Reflection count, i.e. the number of diffraction spots in the X-ray detector image, is one of the possible measures of quality of the diffraction pattern obtained by shooting the crystal. The center of the image is always the place where the beam is aimed at.

This was done by using a CNN as a regressor. As input data, it was given photos of crystals taken while the beam was on. These images were taken from the point of view of the beam source and were centered on the point where the beam hit. An example of these images is shown in Figure 2.2. As output data, the CNN was given a numeric metric of the quality of the diffraction pattern resulting from the shot in the associated image, specifically the number of reflections in the diffraction.

Prediction heatmaps

Once we can guess the reflection count of a particular image, we can use this to find the crystal by creating what we call a *prediction heatmap*. Since the beam always hits the center of the image, we can simulate aiming the beam anywhere in an existing image by cropping around some chosen point. If we then give our cropped image to the neural network, it will predict what reflection count we could expect if we were to fire the beam at the point we cropped around. Now we can do this for each point in a dense grid all over the image, and get a heatmap of where the network predicts a high RC and where it predicts a low RC. We can now aim the beam where the highest RC is predicted, or in some other way decide where to aim from this information.

2.1.2 Technical details

We are using Python 3.6 with the neural-network library Keras [13] to build, train, and test our various models. Description from Keras website:

> Keras is a high-level neural network API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

Keras was at the right level of abstraction for us to make it easy to prototype our models without too much work. We use TensorFlow in the backend since it is a well-known library and popular in the community and was also available to use at the High-Performance Computing cluster (HPC) at MAX IV and Lunarc. Fast.ai [14] and PyTorch [15] were also considered. Fast.ai is similar to Keras and is using PyTorch as backend but didn't have as big community support as Keras.

2.1.3 Data set

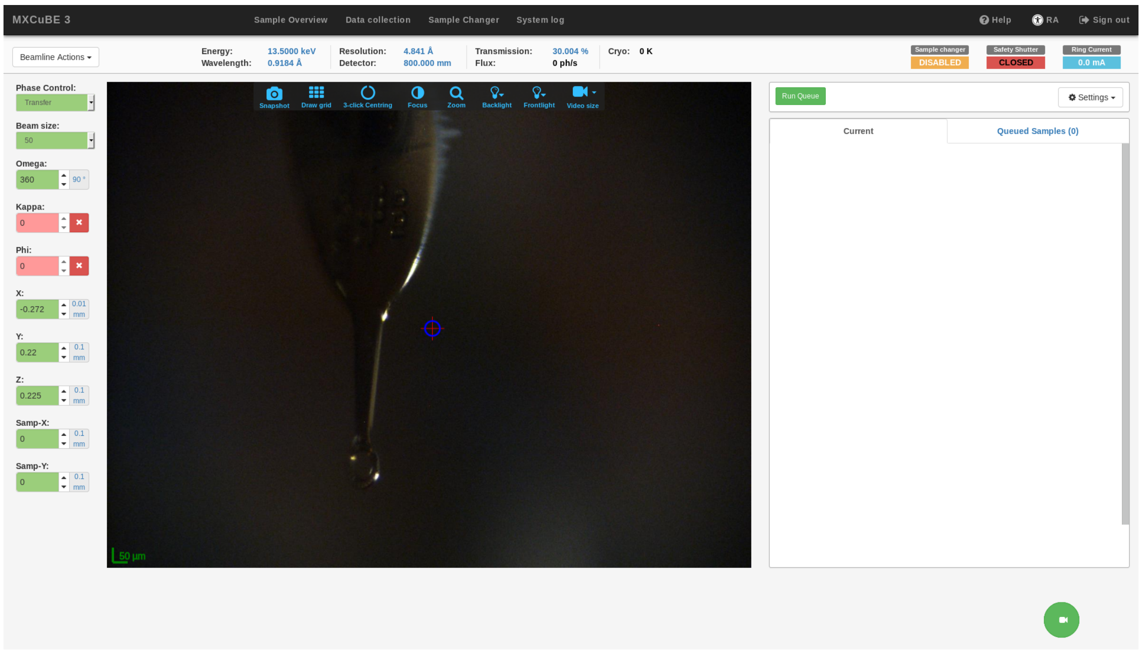
The data set encompasses a collection of images with the observed reflection count (RC) for each image. To collect this we need to capture those images as the beam is firing, then determine which image belongs to which reflection count.

Collecting data

The data was obtained by modifying the source code of MXCuBE [16] which is used to control the data acquisition in BioMAX. Figure 2.3 shows the interface for MXCuBE. There are two "camera" systems relevant to the acquisition of data for the training set. There is an X-ray detector that is hardware-synchronized with the data acquisition. Next is the visible-light camera which is controlled indirectly over another vendor-specific subsystem and it is not hw-synchronized to the supervising control system. In order to acquire images that could be associated with a particular reflection count, modifications were made to save timestamped copies of the image from the visible-light camera at approximately 8 images/second (as fast as performance would allow) while the beam was on.

While collecting data, we moved the crystal in the following two ways.

Helical scan is where the crystal is moved in a line through the beam. This gives us a high variation of reflection counts since the beam moves through both good and bad parts of the crystal, as well as the background. However, the variation in images is quite small, since they are all from the same angle in a given scan.



Energy: 13.5000 keV
Wavelength: 0.9184 Å
Resolution: 4.841 Å
Detector: 800.000 mm
Transmission: 30.004 %
Flux: 0 ph/s
Cryo: 0 K
Sample changer: DISABLED
Safety Shutter: CLOSED
Ring Current: 0.0 mA

Phase Control: Transfer
Beam size: 50
Omega: 360 (90 °)
Kappa: 0
Phi: 0
X: -0.272 (0.01 mm)
Y: 0.22 (0.1 mm)
Z: 0.225 (0.1 mm)
Samp-X: 0 (0.1 mm)
Samp-Y: 0 (0.1 mm)

Figure 2.3: Graphical interface of MXCuBE running in browser.

Rotational scan is where the crystal is manually centered and then rotated while firing the beam. This yields a high variation in images since the angle is always changing, but all the reflection counts will likely be quite high since we are always hitting the crystal.

We used both these types of scans in order to gain a high variation in both images and reflection counts.

Matching image and RC

Our strategy for determining which image belonged to which reflection count was to timestamp each image as it was being captured, and then match those timestamps with those of the diffractions being generated by the detector.

BioMAX utilizes EIGER 16M X-ray detector [17] which outputs data in the HDF5 file format [18]. This file contains timestamps of each frame (the smallest unit in which the detector's input is divided by time, analogous to a frame in a normal video) but these were unfortunately not accurate, so another way to match each image with each frame was necessary. The solution was found in the log file from EIGER. These logs have timestamps of when each frame is written into the HDF5 file. The solution is a compromise available at the time of the project. Our method relies on logs that disappear after a few days and are not guaranteed to match from a time-perspective so it is not much robust. Time might not be synchronized on the different servers.

RC was found by using DIALS [19] to count the number of reflections in the diffraction image in the HDF5 file from EIGER. Instructions for doing so are presented in appendix A.

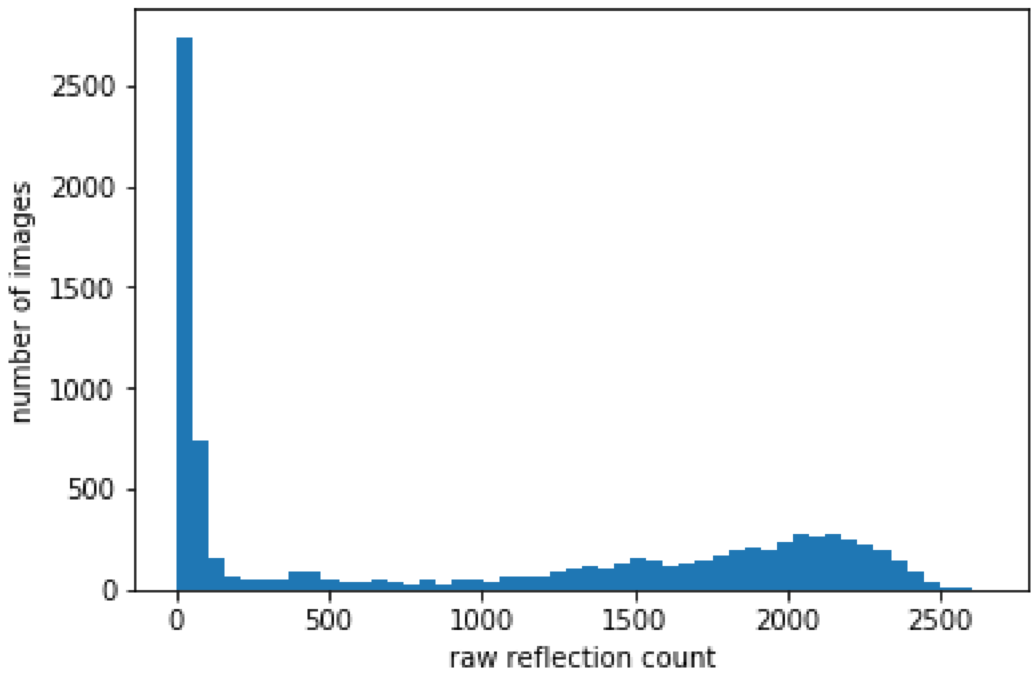


Figure 2.4: Histogram of the reflection counts in the whole data set.

The complete data set was then assembled by going through log timestamps and find a matching image if the image was not taken too far away in time. You can see the distribution of the reflection counts for all the images in Figure 2.4. As you can see, there is a big spike towards the lower end. These are all the bad shots, often not even pointing at the crystal, resulting in roughly 0 reflections. The high values are more spread out, ranging up to over 2500 reflections.

In the end we had a dataset of roughly 12000 pairs of images and reflection counts.

The data set is split into three parts:

- Training
- Validation
- Testing

This split was 60/20/20 (training/validation/testing), a rather typical ratio [20]. The training set is used during the training phase to fit parameters such as weights, kernels, and bias. The validation set is also utilized for the training but it is used to optimize hyperparameters such as learning rate and momentum. The testing set is used for an unbiased evaluation of the final model.

2.1.4 Transformation and augmentation

The following sequence of transformations is made to the images before they are fed into the neural network.

Cropping and resizing are applied to normalize and to put the focus on the object itself. First, the images are cropped into squares. The size of the cropping linearly depends on the zoom level of the camera when the image was taken (which was saved as metadata at the time of data collection). The output size after resizing the cropped image is 128×128, a loose compromise between fidelity and computation speed.

We then normalize the pixel values of the images to have average value 0 and standard deviation 1. Not only does this help neural nets learn [21], it also deals with the problem that some images are much brighter than others, removing a factor that could confuse the network.

The reflection counts are also normalized in the same fashion as the images, but along the whole data set, so the mean value is 0 and the standard deviation is 1. We will call an RC (reflection counts) that has been fed through the transformation pipeline TRC (transformed reflection count).

Data augmentation

A common practice in machine learning is to artificially increase the size of your dataset by adding modified versions of the data you already have. This is to stop the network from overfitting to features we know to be irrelevant.

We can assume that the horizontal "flippedness" of the image should affect what reflection count we can expect from it. Therefore we add the horizontally-flipped version of every image with the same reflection count.

On top of this, for each image we add a version that has been rotated $90°$, $180°$ and $270°$. We could have added more rotations but it would be more cumbersome since we would also need to zoom or add black pixels around. We have already octupled the size of our dataset. An example of this data augmentation can be seen in Figure 2.5.

By using this data augmentation our dataset increases in size from circa 12 000 pairs to circa 96 000 pairs.

2.1.5 Variations on data transformation

In an attempt to increase the accuracy of our model, Three different variations of this transformation sequence were tried over the course of this project. These are presented here.

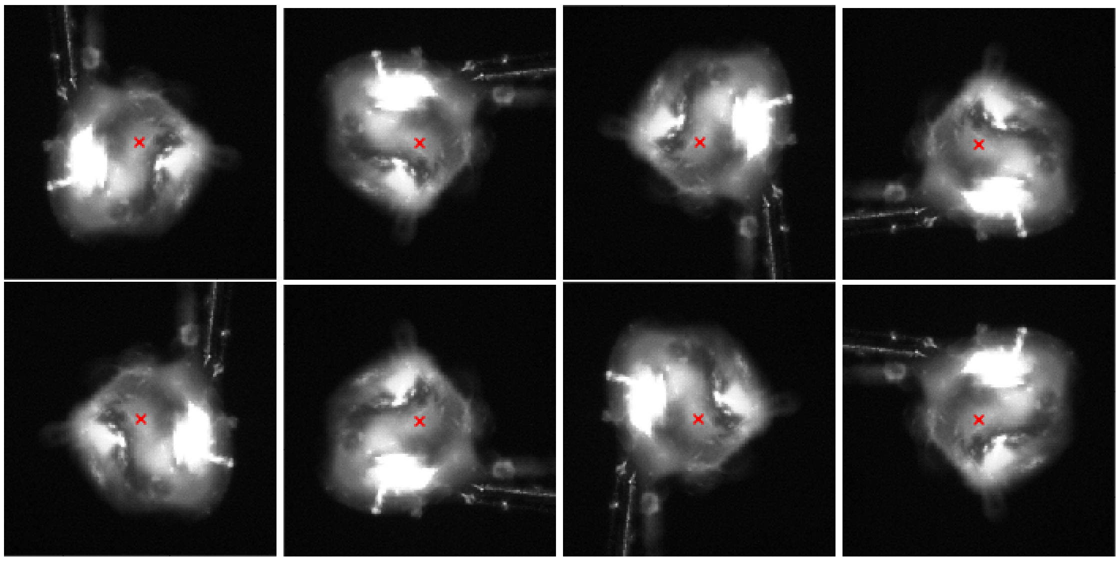


Figure 2.5: The resulting images from data augmenting a single image. All these would have the same reflection count in the dataset.

Vanilla transformed

This is the transformation sequence described above in Section 2.1.4. Histograms after splitting the dataset into train/test/validate are visible in Figure 2.6. Using this transformation pipeline, we can see that there is a high variance in the reflection counts where the beam hit the crystal (those above 100). This is to be expected since the crystals are of varying quality. By this, we mean that some crystals have a higher capacity in the number of reflections that can be produced in a diffraction. However, it might add unnecessary difficulty to the neural network. For example, if the network predicts that a particular image will give a reflection count of 500, and it was 2000 reflections, it would look like a large error; but for our purposes, 500 is a perfectly acceptable guess, since it means we are aiming somewhere on the crystal, which is what we care about. This is why we created the two variations below.

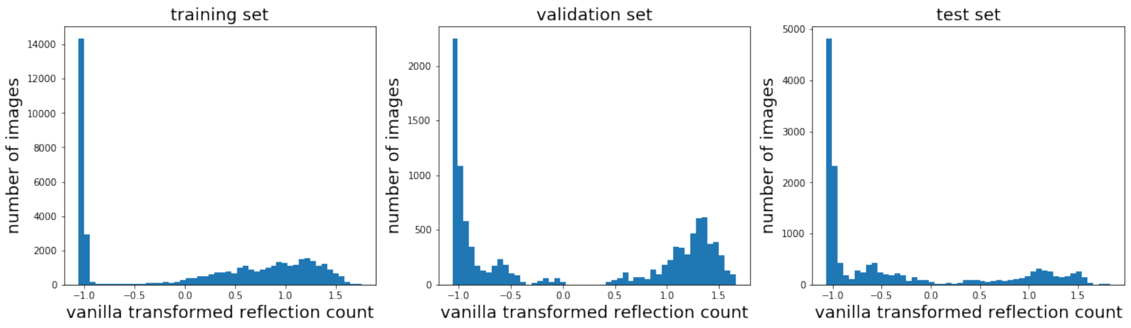


Figure 2.6: Histograms of the vanilla transformed reflection counts in the three parts of the data set. Note that the range has shrunk due to the the normalization transformation, as mentioned in Section 2.1.4.

Log transformed

Here the reflection counts are logarithmized (i.e. being modified by the function log(x)) after being normalized. The log function is used because it groups large values. The point of this is to reduce the difference between 'good enough' reflection counts of crystals of different quality. One crystal might have a peak reflection count of 1500 while a lower quality crystal can only produce 500 reflections at best, but there is no point in the model trying to distinguish between these cases. All we care about is hitting the crystal. The histograms of the transformed reflection counts are visible in Figure 2.7.

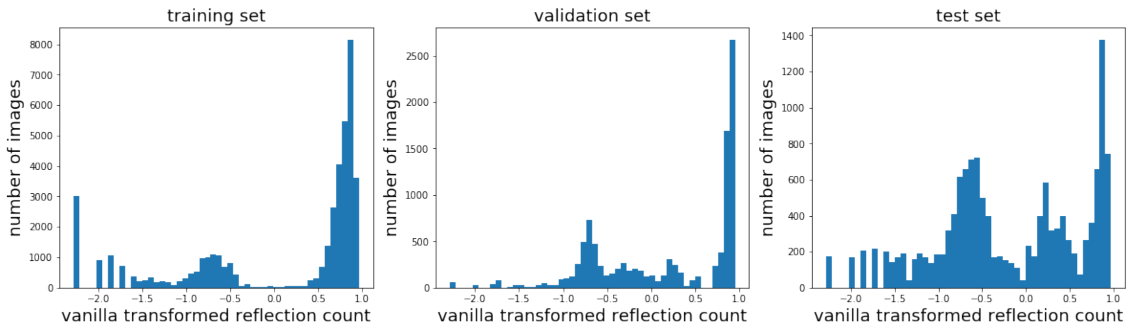


Figure 2.7: Histograms of the log transformed reflection counts in the three parts of the data set.

Tanh transformed

Although logarithmizing reflection counts clusters high values together, the low values are spread out. But we do not care about the difference of an image with 3 reflections and one with 10. They are both worthless, so the data set should reflect that. We thought that the hyperbolic tangent (tanh) function (plotted in Figure 2.8) could be a good way to cluster together both low and high values. The histograms of the modified reflection counts are visible in Figure 2.9

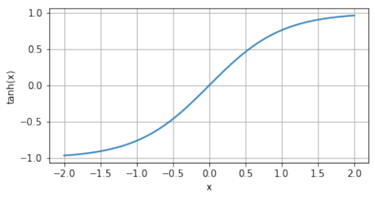


Figure 2.8: Plot of tanh(x)

2.1.6 Model

The CNN architectures used for this experiment are regressors, meaning they are for guessing a particular value (in this case the transformed reflection count) rather than choosing

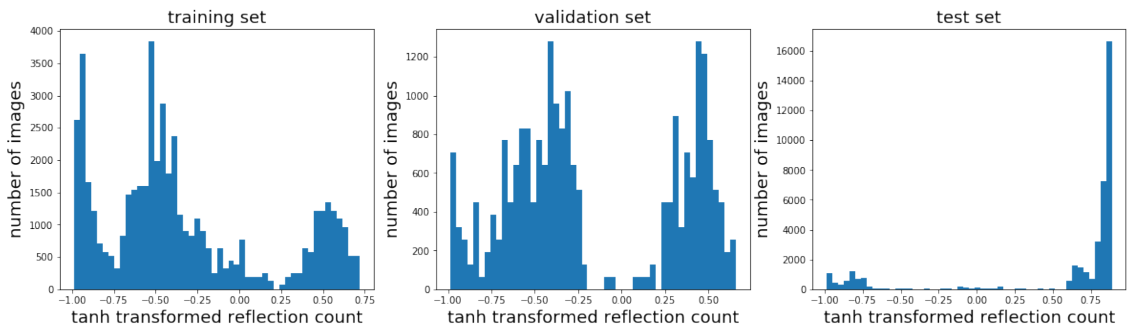


Figure 2.9: Histograms of the tanh transformed reflection counts in the three parts of the data set.

between a set of classes. [22] The input is some image of a crystal taken from the beam-source while the beam is firing. The point that is hit by the beam is in the center of the image. The output is the reflection count yielded from that particular firing of the beam. Both input and output are first transformed as described in section 2.1.4.

The neural network begins with a series of convolutional layers, followed by flattening and a series of fully connected layers (as described in section 1.2.3). The final output layer only has one neuron which represents the guessed transformed reflection count. All architectures are shown in Table 2.2.

Keep in mind that this project is not about optimizing a neural net for the given task. Its main aim is answering a principal question if we can get a neural net to solve these kinds of problems. And so the 10 architectures are somewhat arbitrarily chosen by us to get a good variety of models. Some architectures are deeper. Some have more convolutional layers. Some have more fully connected layers. There are likely other architectures that would perform much better. The reason only one architecture includes a max-pooling layer is that they reduce the spatial information in the image. The pooling layers are useful if one wants to know if there is an object in the image, but do not care about the object position. Since we care very much about where the object (in particular crystal in this case) is, pooling layers did not seem like they would be much help. [5]

All tested architectures were using the Adam optimizer [24] with Mean Squared Error as the loss function. All layers use the activation function ReLU [23] except the last one with no activation function since this is a regressor. Adam, MSE, and ReLU were chosen because they are all rather typical choices for neural networks. Each model is trained until its MSE on the validation set stops decreasing. This is called early stopping and is conveniently available as a callback function in Keras (documentation: `https://keras.io/callbacks/#earlystopping`) [13].

Header	Description
C*N*	Number of convolutional layers with N filters of size 3×3 followed by 25% dropout. [23].
MPOOL	Number of max pooling layers [2]
F*N*	Number of fully connected layers with N neurons followed by 50% dropout [23]

Table 2.1: Keys for the Table 2.2. These values (including the values for N) were chosen somewhat arbitrarily by trial and error. We had to be selective in which hyperparameters to vary due to time constraints.

Architecture	C8	MPOOL	C16	F256	F128	F64
a1	2	0	0	1	1	0
a2	2	0	1	1	1	0
a3	2	0	2	1	1	1
a4	2	1	2	1	1	1
a5	2	0	2	2	2	2
a6	2	0	0	0	1	1
a7	1	0	1	1	1	0
a8	1	0	0	1	1	0
a9	1	0	0	1	0	0
a10	1	0	0	1	1	1

Table 2.2: The different neural network architectures used in this experiment. Key for the headers is in table 2.1. The layers are in the same order as in the table.

2.1.7 Evaluation

Here we describe the metrics and methods we used to evaluate our models. Some of these methods are more quantifiable, while others are more based on perception.

Mean Square Error

We use Mean Square Error as our loss function, as described in section 1.2.6. We chose it over Mean Absolute Error (MAE) because if the model guesses almost the right reflection count, that is good enough. We only care if the guess is way off, and MSE reflects that by inflating big errors more than small errors, therefore MSE seemed to us like a more relevant metric than MAE.

Correlation Graph

For each model, we create scatter plots with real reflection count on the X-axis and predicted reflection count on the Y-axis. This is to help us visualize how well the model is doing and what it's getting wrong.

High and low predicted images

It could be useful to see what kind of images it predicts to have the highest/lowest reflection count, as well as what images it gets wrong. We chose a random sample of images that the model predicted to have a very high RC or very low RC. This way we can see if the predictions of RC look reasonable for those images.

Prediction heatmap

The final, actual use case of this is to find the best point (or at least *a point*) on the crystal to shoot. To simulate this we take an image of a crystal and crop around different points on the image, arrayed in a grid, thus creating input data for the neural net as if we were aiming at that point. We can then use the neural net to try to predict the RC on each of those cropped images. Hopefully, the ones that are more centered on the crystal will have a higher predicted RC. To make this more useful, we will put an X on the center point of the highest N points in the point matrix, where N is some low positive integer. This will serve as our guess for where to aim the beam.

2.1.8 Results

Mean Square Error

Table 2.3 shows the resulting scores of our various experiments with using the regressors described in section 2.1.6 on page 24.

	Layers						MSE		
Architecture	C8	MPOOL	C16	F256	F128	F64	vanilla	log	tanh
a1	2	0	0	1	1	0	0.5789	0.3207	0.3638
a2	2	0	1	1	1	0	0.5241	0.5729	0.4886
a3	2	0	2	1	1	1	0.923	0.5539	0.4789
a4	2	1	2	1	1	1	0.7803	0.3789	0.3572
a5	2	0	2	2	2	2	0.8812	0.4141	0.4467
a6	2	0	0	0	1	1	0.4279	0.3539	0.3585
a7	1	0	1	1	1	0	0.6095	0.3794	0.4088
a8	1	0	0	1	1	0	0.3775	0.2931	0.2291
a9	1	0	0	1	0	0	0.442	0.3036	0.2188
a10	1	0	0	1	1	1	0.4292	0.2076	0.2869

Table 2.3: Result of training on various architectures with various data set. Keep in mind that the MSE showed is the error in the *transformed* reflection count, therefore MSE scores are not as comparable between different transformation pipelines as they are between different models. The key for this table is found in Table 2.1 on page 26.

Correlations

More interesting is the correlation between the real reflection counts and the predicted. Figure 2.10, 2.11, and 2.12 show the correlations using log-, tanh-, and vanilla-transformed reflection counts respectively (TRC).

High and low predicted images

To get a more intuitive understanding of the model, we present some images that were predicted to have very high or very low RC. We will focus on one of the more promising of our 10 models.

Figure 2.13 shows some images that received very low predictions (under 0). Figure 2.14 shows some images that received very high predictions (over 0.7). Note that in all these images, the red cross in the center is added afterwards to make it easier for you to see where the physical beam was aimed when this picture was taken and is not part of the data.

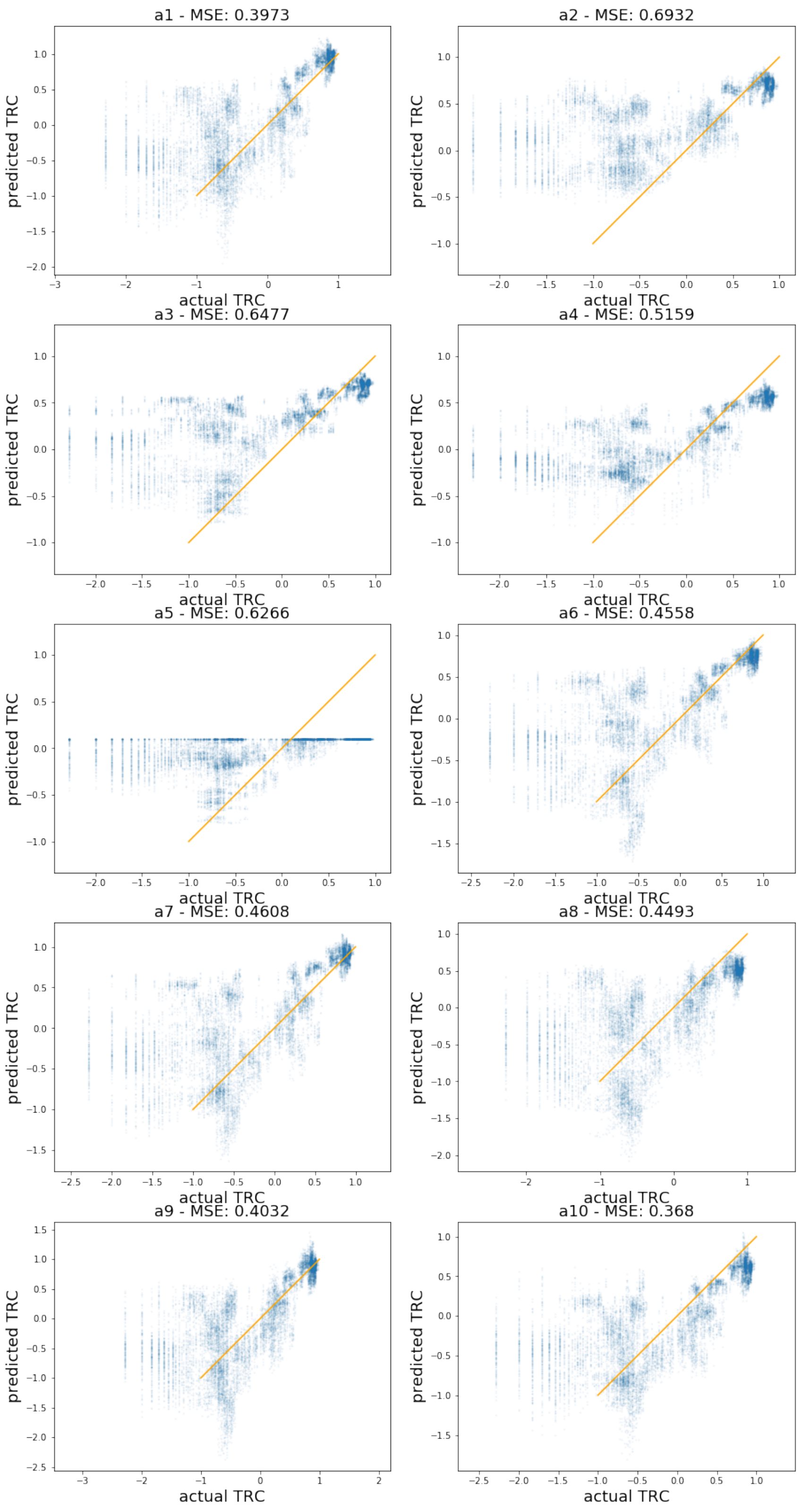


Figure 2.10: Correlation of the real values and the predicted values (logarithmized). Each dot corresponds to an image, the X-axis is the actual value and the Y-axis is the predicted value. The orange line is x=y, which is the ideal target.

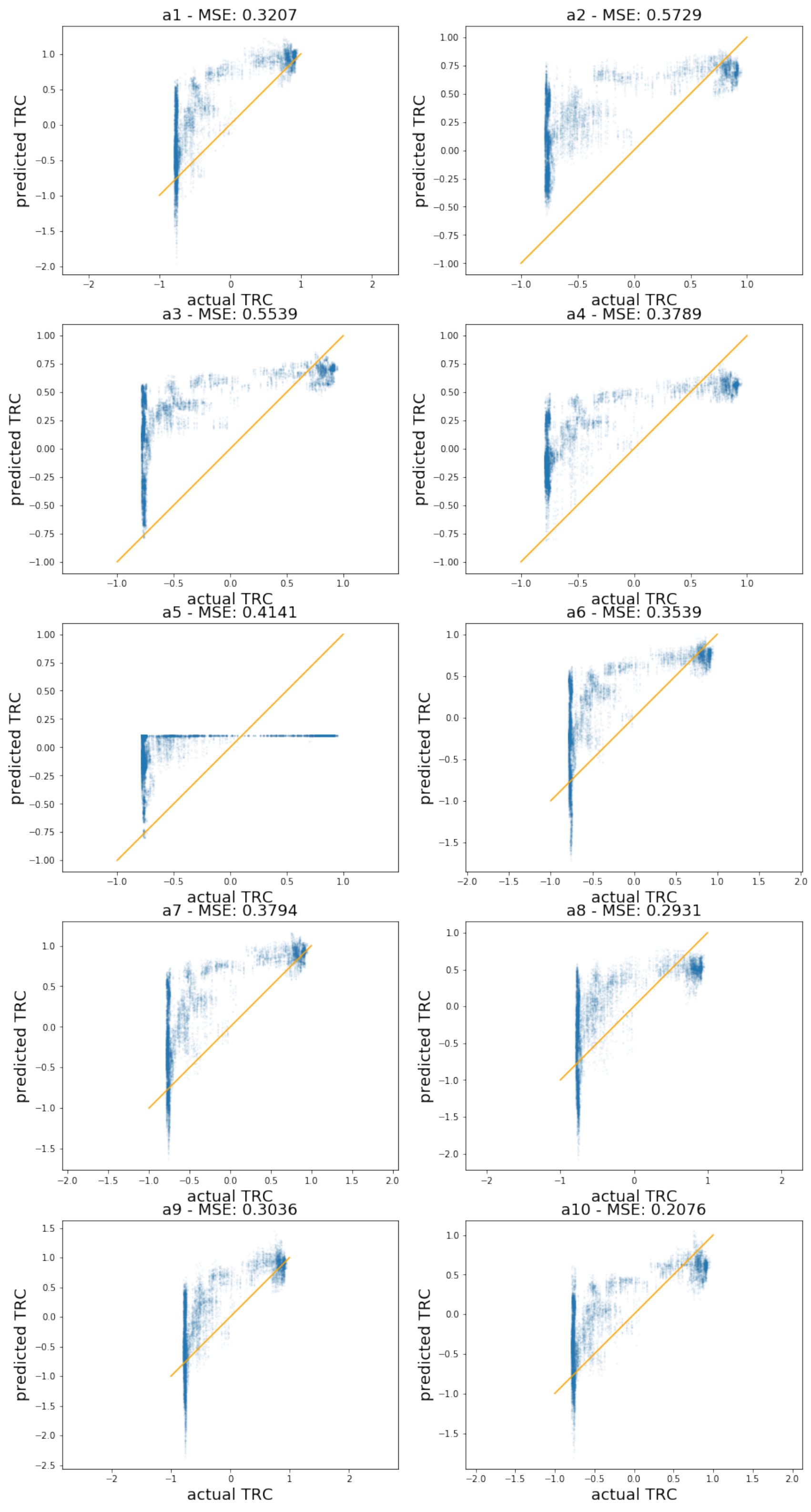


Figure 2.11: Correlation of the real values (modified by tanh(x)) and the predicted values. Each dot corresponds to an image, the X-axis is the actual value and the Y-axis is the predicted value. The orange line is x=y, which is the ideal target.

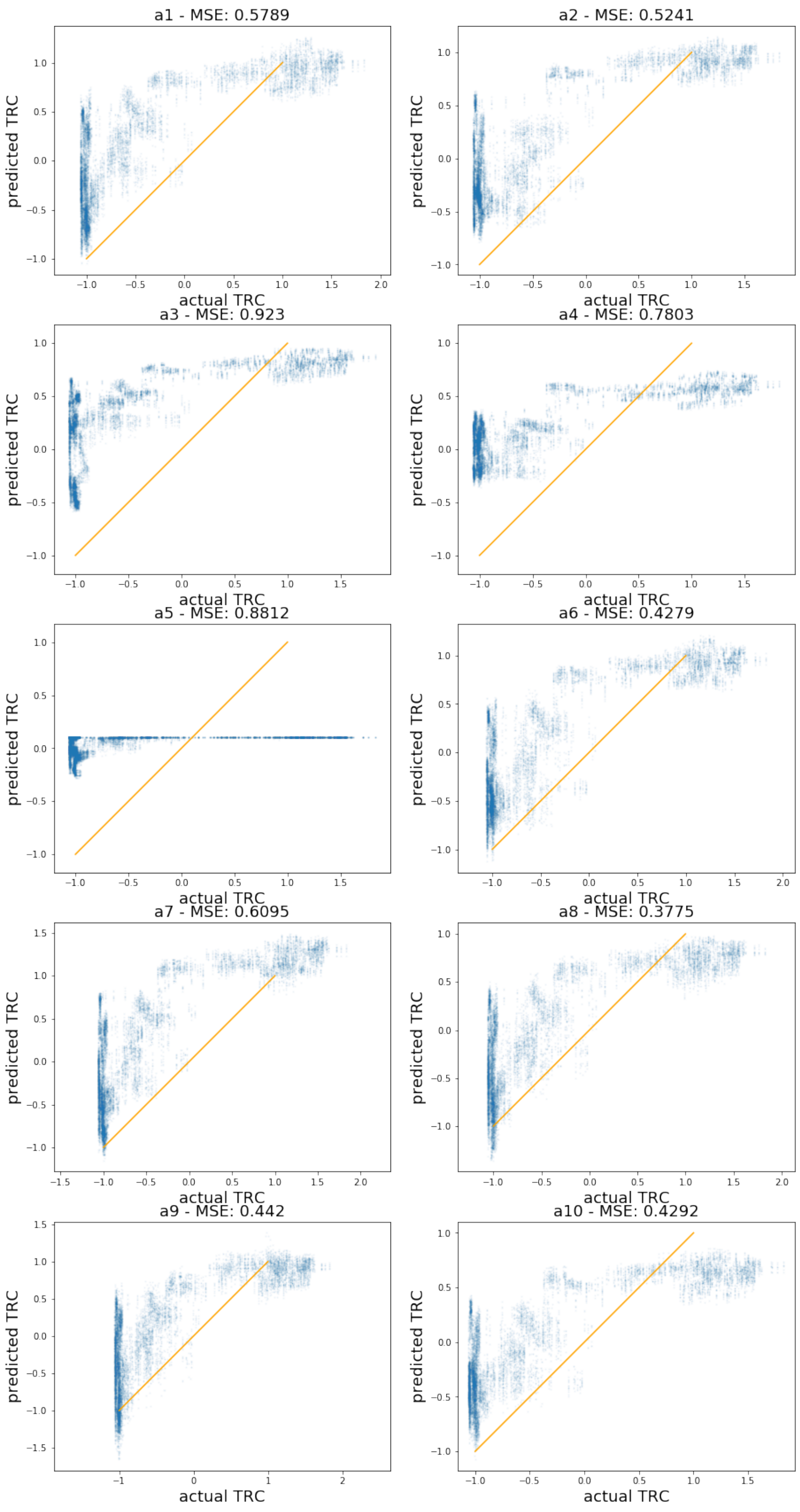


Figure 2.12: Correlation of the real values (unmodified). Each dot corresponds to an image, the X-axis is the actual value and the Y-axis is the predicted value. The orange line is x=y, which is the ideal target.

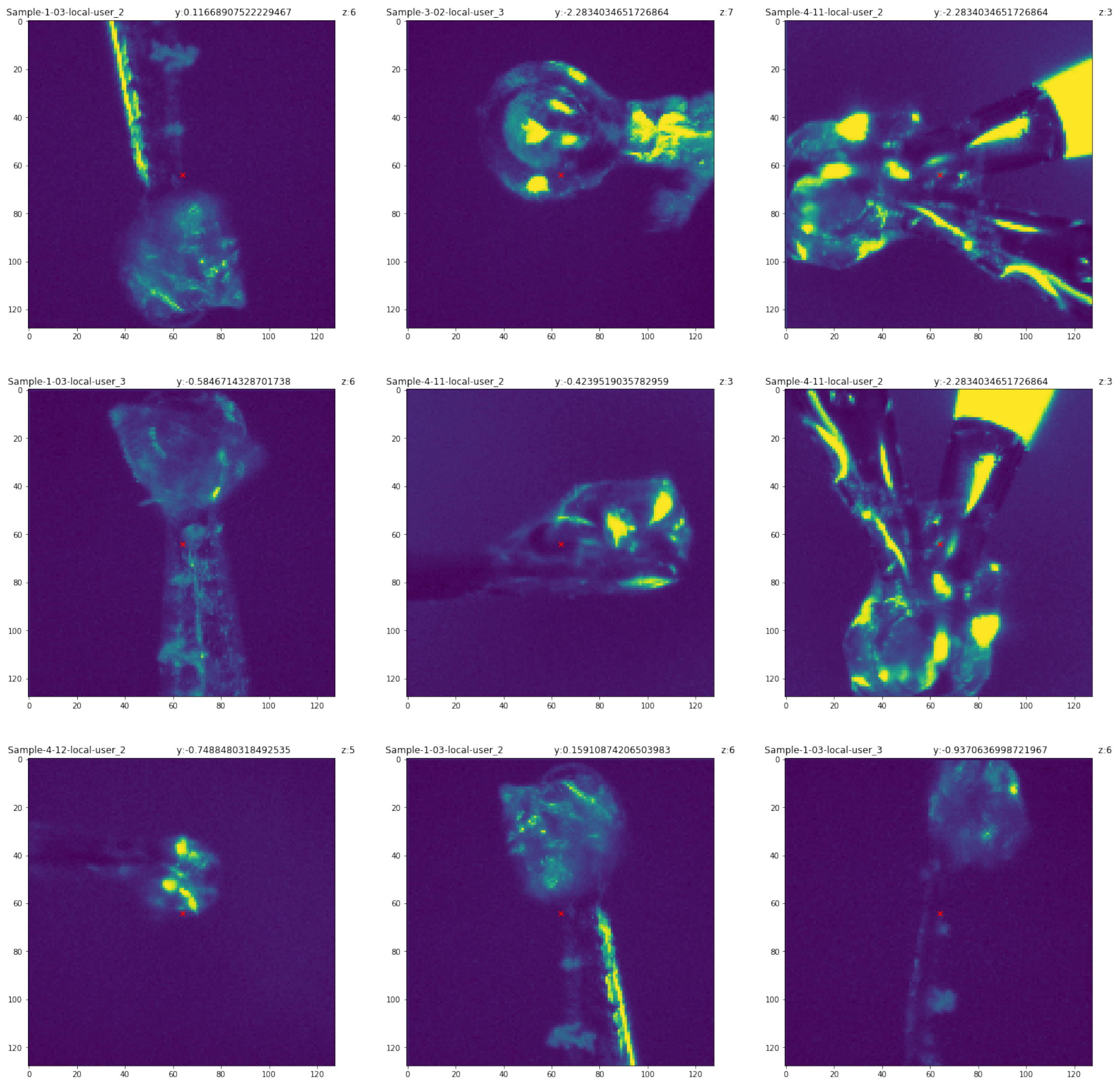

Sample-1-03-local-user_2 y:0.11668907522229467 z:6
Sample-3-02-local-user_3 y:-2.2834034651726864 z:7
Sample-4-11-local-user_2 y:-2.2834034651726864 z:3
Sample-1-03-local-user_3 y:-0.5846714328701738 z:6
Sample-4-11-local-user_2 y:-0.4239519035782959 z:3
Sample-4-11-local-user_2 y:-2.2834034651726864 z:3
Sample-4-12-local-user_2 y:-0.7488480318492535 z:5
Sample-1-03-local-user_2 y:0.15910874206503983 z:6
Sample-1-03-local-user_3 y:-0.9370636998721967 z:6

Figure 2.13: Images that the model predicted to have a low RC.

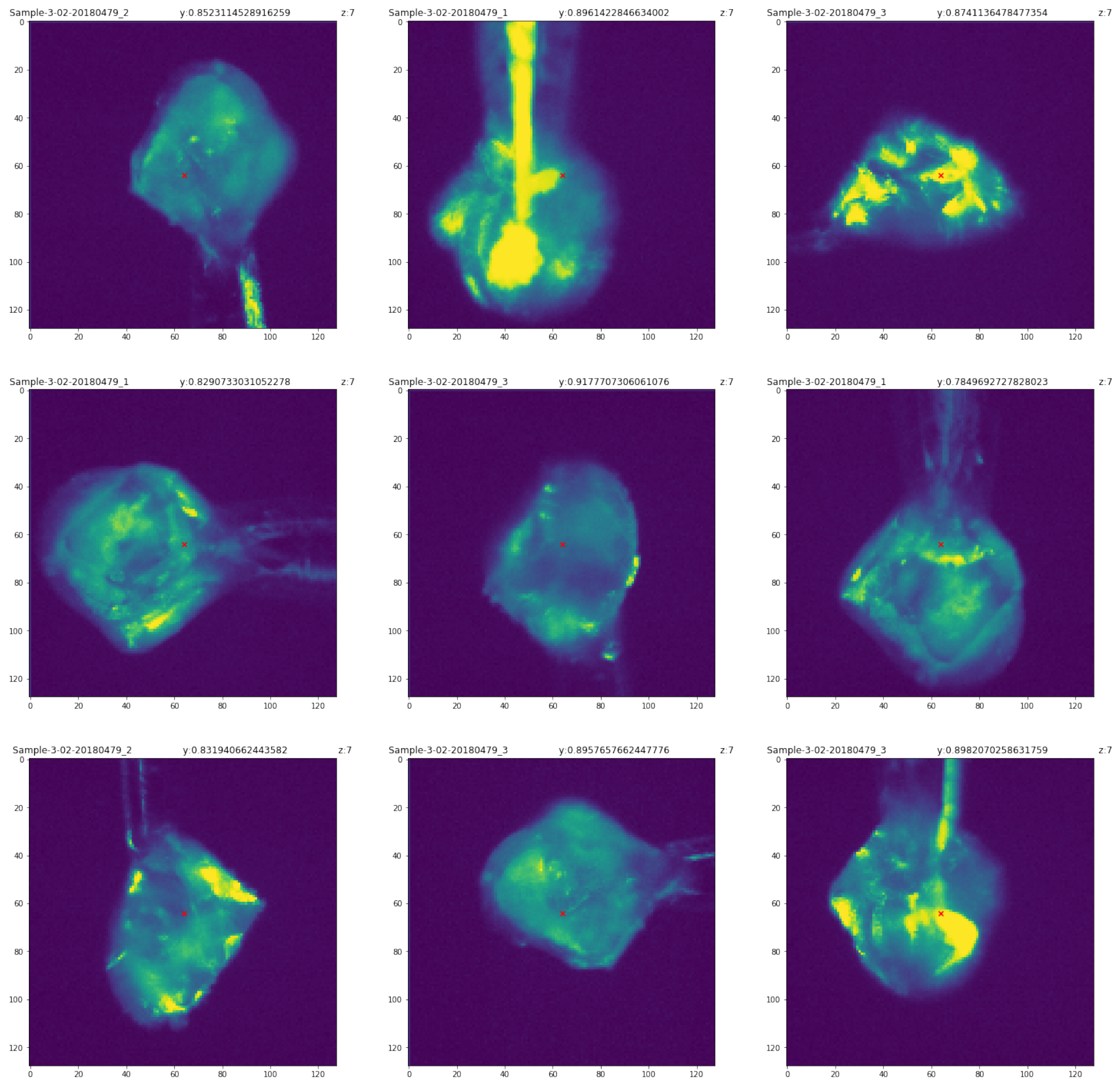

Sample-3-02-20180479_2 y:0.8523114528916259 z:7
Sample-3-02-20180479_1 y:0.8961422846634002 z:7
Sample-3-02-20180479_3 y:0.8741136478477354 z:7
Sample-3-02-20180479_1 y:0.8290733031052278 z:7
Sample-3-02-20180479_3 y:0.9177707306061076 z:7
Sample-3-02-20180479_1 y:0.7849692727828023 z:7
Sample-3-02-20180479_2 y:0.831940662443582 z:7
Sample-3-02-20180479_3 y:0.8957657662447776 z:7
Sample-3-02-20180479_3 y:0.8982070258631759 z:7

Figure 2.14: Images that the model predicted to have a very high RC.

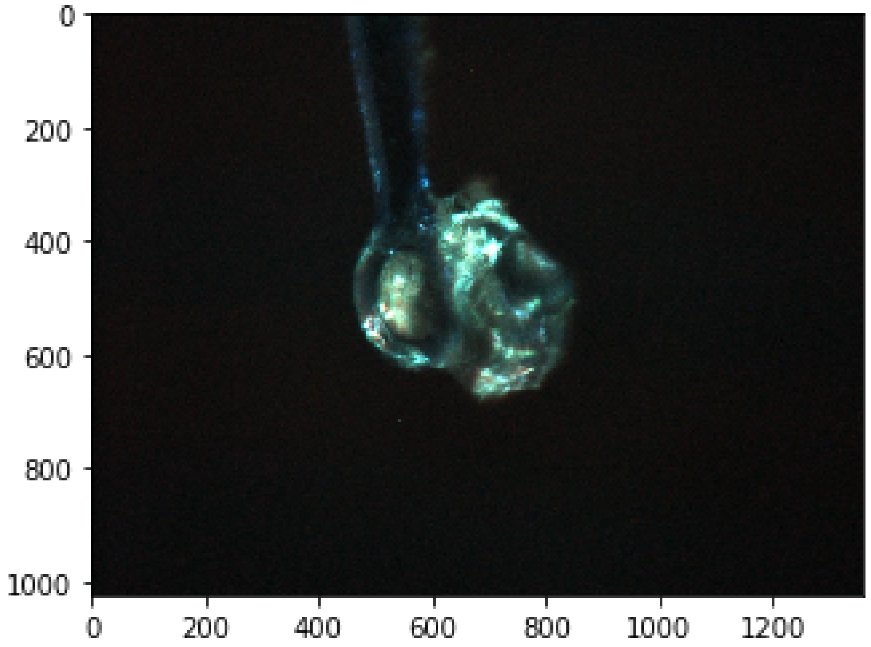


Figure 2.15: The raw image used as the only input to the trained model to create the heatmaps below. The models have never seen this crystal before.

Prediction heatmaps

Here we present the prediction heatmaps that were described in Section 2.1.1 on page 18. This shows how our models would perform in the real use case on a crystal they've never seen (shown in Figure 2.15). See Figure 2.16, 2.17, and 2.18.

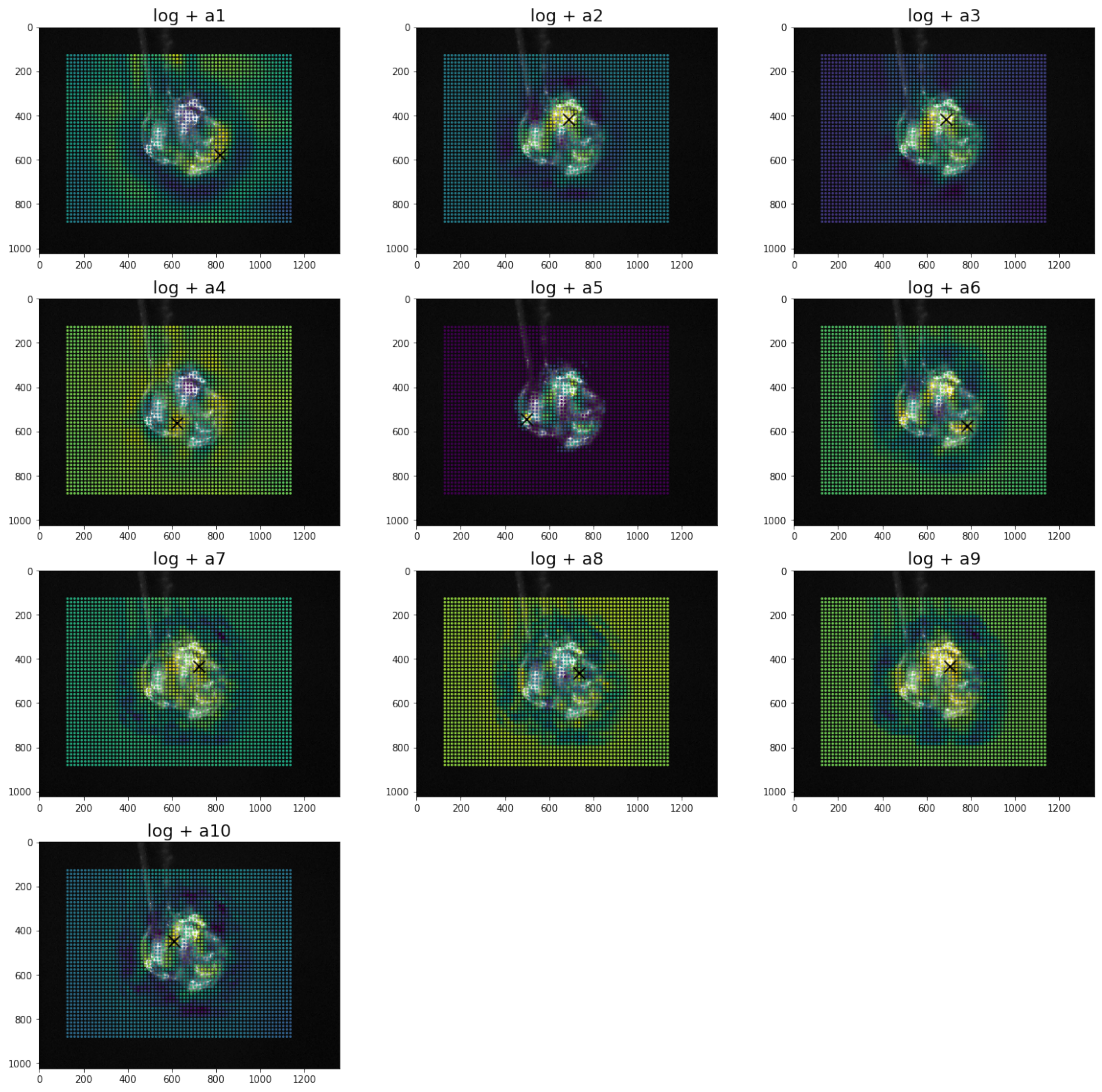


Figure 2.16: prediction heatmaps (as described in section with architectures trained and tested on log-transformed reflection counts. Warmer color means higher predicted reflection count. The black X is the global maximum.

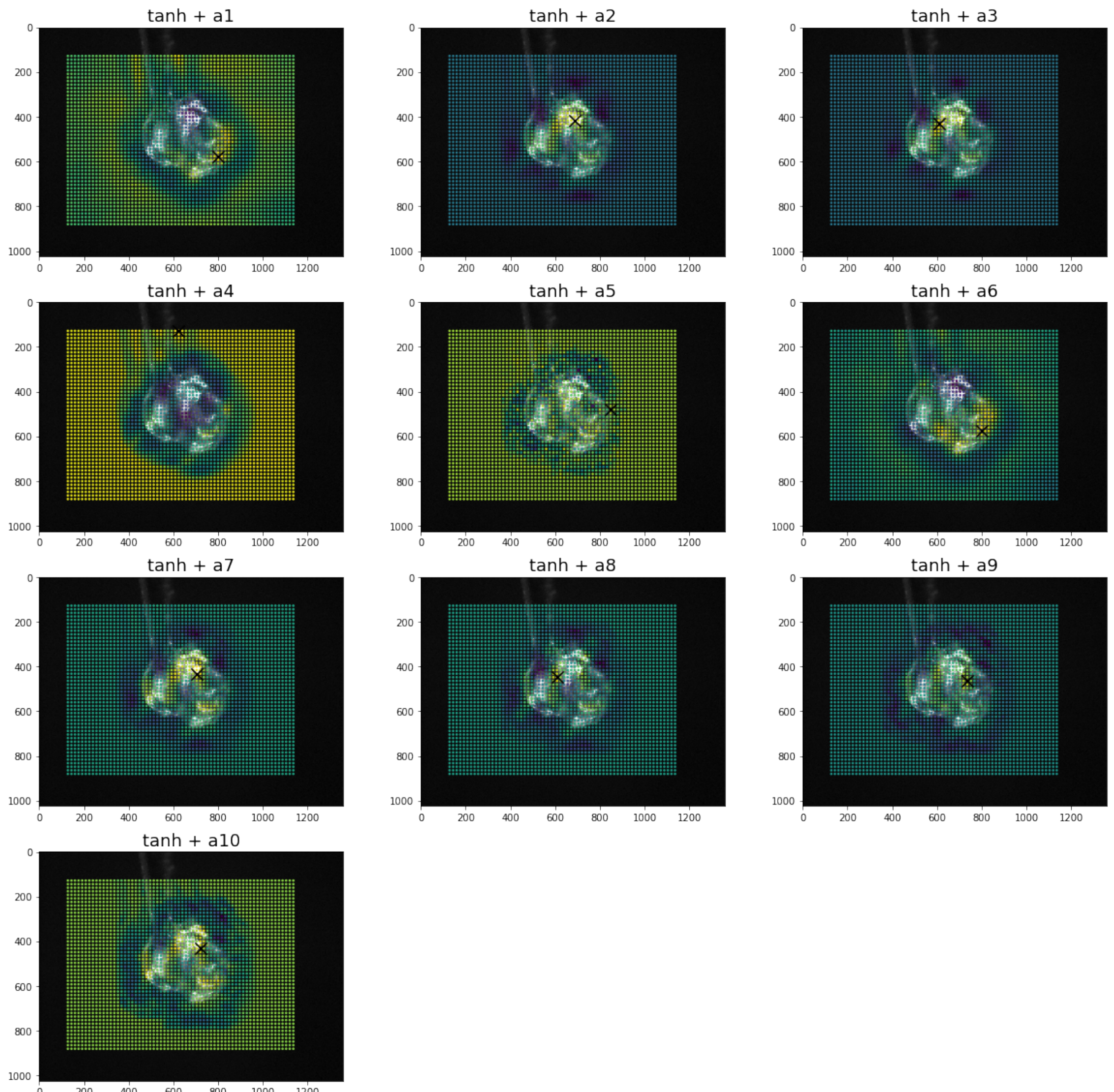


Figure 2.17: prediction heatmaps with architectures trained and tested on tanh transformed reflection counts. Warmer color means higher predicted reflection count. The black X is the global maximum.

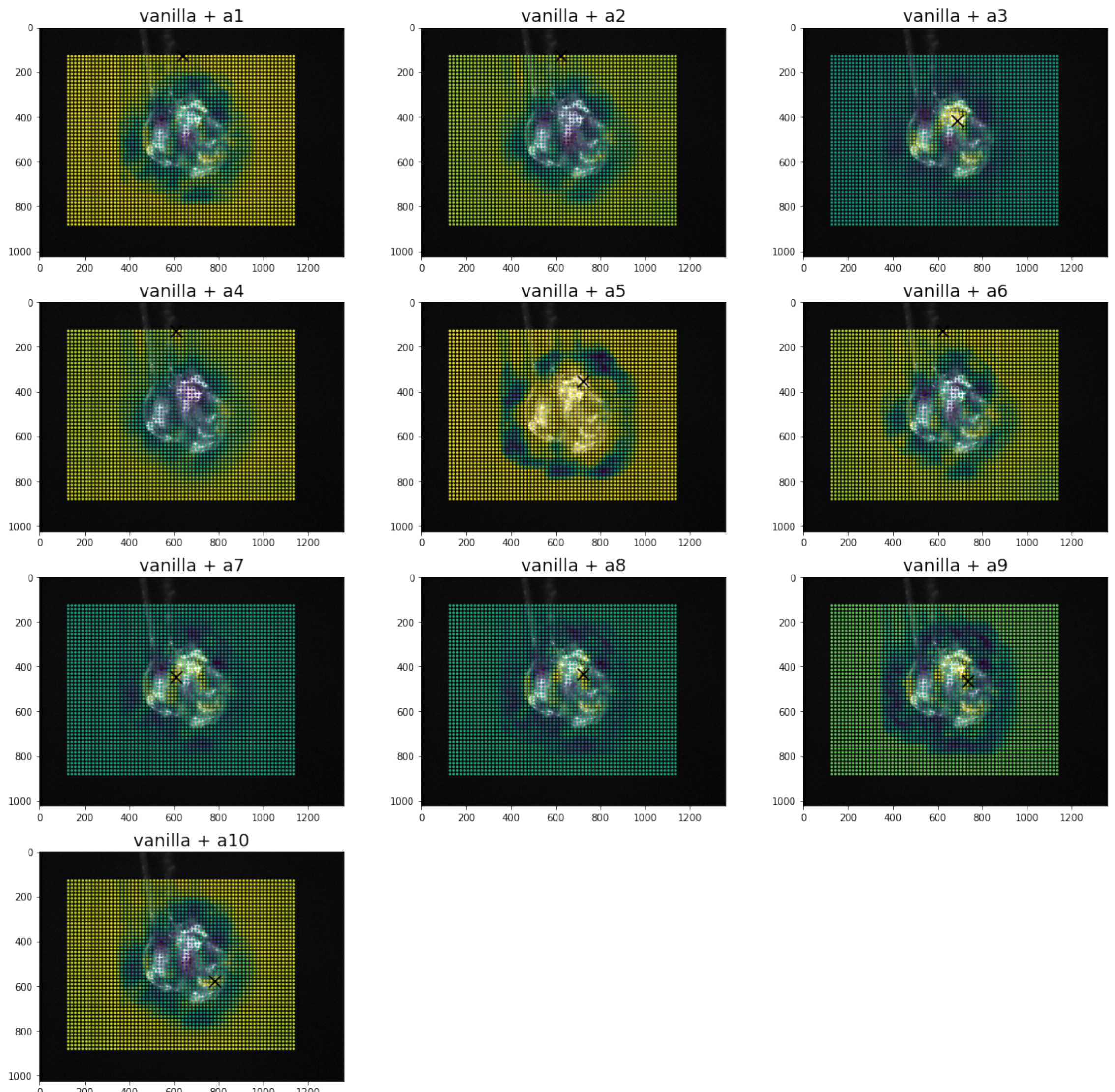


Figure 2.18: prediction heatmaps with architectures trained and tested on vanilla transformed reflection counts. Warmer color means higher predicted reflection count. The black X is the global maximum.

2.1.9 Discussion

From the tables, it appears the simpler models are usually the ones that produce better results. MSE might not be very trustworthy though. If we look at Figure 2.6, 2.7, and 2.9 in subsection 2.1.4, we see that the distributions of reflection counts are quite different between the training, validation, and testing sets. This might result in unfair MSE scores. For example, the tanh variant has very few low values in its test set, so if low reflection counts are predicted poorly, that will not affect the MSE as much as it should. This problem stems from the quite small size of our data set and the fact that we need to separate our data by crystals so that no crystal is in two sets. We think this problem is likely to even out if we collected data from more crystals since there could then be a greater variety of crystals with different ranges of RC in each subset.

Regarding the correlations, a theme common to all models is that they have trouble with the very low-RC images. However, we found this to be quite excusable when we look at the images it predicts too high but is low (see Figure 2.19). We think they are visibly very similar to images that do have high reflection counts, and therefore the model can not be faulted for predicting it as high. This is more likely to be a problem with our dataset. After showing this to some beamline scientists working in BioMAX, it turned out that this is a common problem that they run into even when centering manually.

Other than that, the correlations for the log-based models (Figure 2.10) are surprisingly highly correlated for images with high reflection counts! It is important to remember that these are not the actual reflection counts, but the log-transformed reflection counts. If we were to transform the data back into real reflection counts, we would have much bigger errors for high reflection counts. However, the point of this is not to exactly predict the reflection count, just to make sure if we are hitting the crystal in a decent place.

When we look at the heatmaps in Figure 2.16 and 2.17, a recurring problem seems to be that some of the networks get confused when the crystal is almost or completely out of view. When the image is cropped so that the crystal is barely nor not at all visible some models get confused, which results in a bright background in the heatmap. This is not that strange since we barely have any images in the data set when the beam is aimed nowhere near the crystal. This could quite easily be solved with data augmentation or with further data collection. Despite this problem, some of these models look rather useful for the task, given the heatmaps. Except for tanh + a4 and arguably both a5, all models chose a point somewhere on the crystal. Another interesting phenomenon is that it seems that we can visibly see the extreme overfitting of some models in the noisiness of their heatmaps, for example, tanh + a5. Points that are very close to each other, and thus have very similar cropped images still get predicted as having very different RC, resulting in a grainy appearance in the heatmap.

The most relevant method to compare our results to are Lucid 2 and 3 [9][10]. The results of using them to find the crystal in the image used for the heatmaps are visible in Figures 2.20 and 2.21. It's important to point out that lucid and our method have different goals and will not achieve the same result. To quote lucid 2 technical manual [9]:

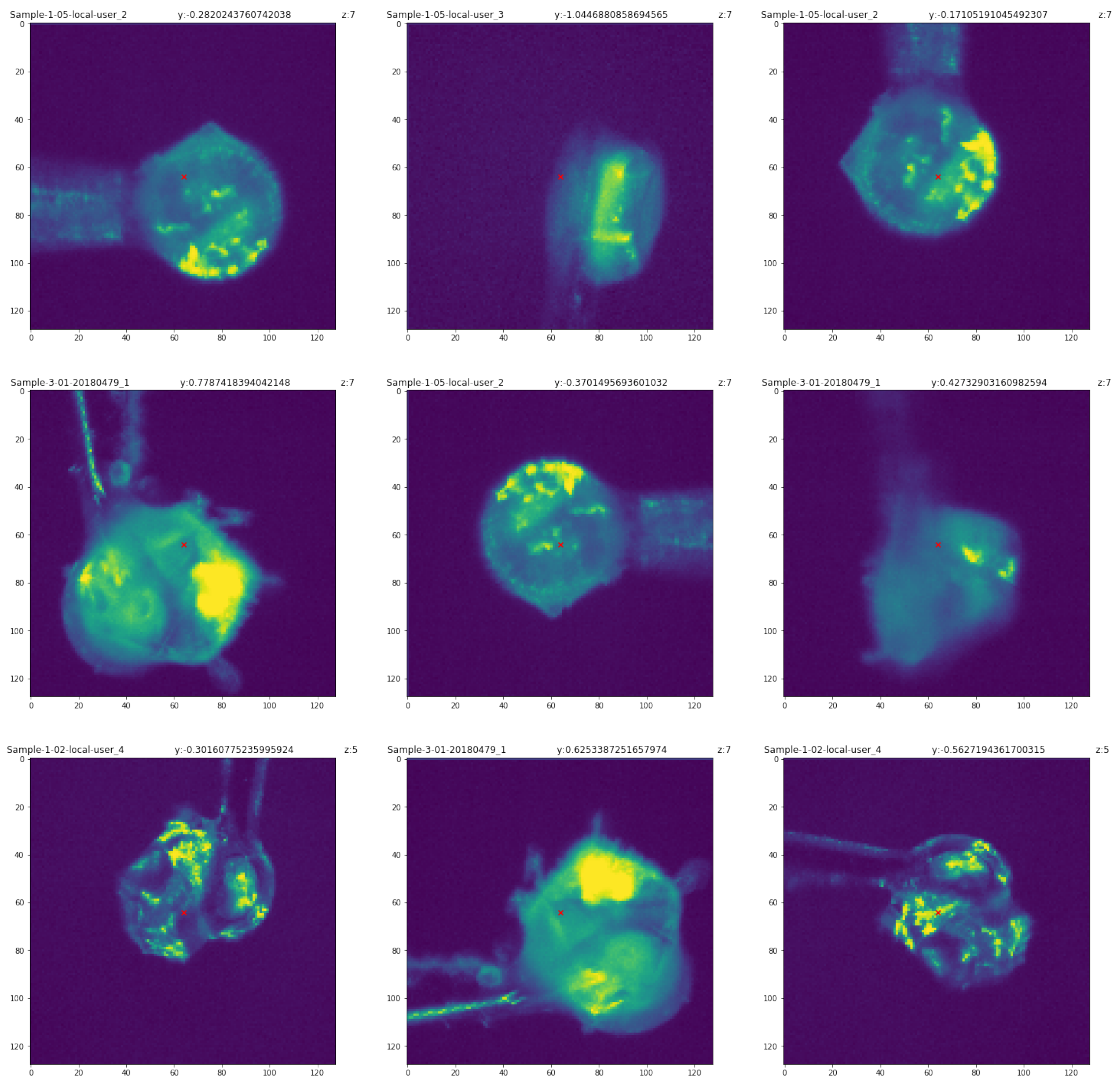

Sample-1-05-local-user_2 y:-0.2820243760742038 z:7
Sample-1-05-local-user_3 y:-1.0446880858694565 z:7
Sample-1-05-local-user_2 y:-0.17105191045492307 z:7
Sample-3-01-20180479_1 y:0.7787418394042148 z:7
Sample-1-05-local-user_2 y:-0.3701495693601032 z:7
Sample-3-01-20180479_1 y:0.42732903160982594 z:7
Sample-1-02-local-user_4 y:-0.30160775235995924 z:5
Sample-3-01-20180479_1 y:0.6253387251657974 z:7
Sample-1-02-local-user_4 y:-0.5627194361700315 z:5
0
20
40
60
80
100
120

Figure 2.19: Images that have very low real RC, but were predicted to have quite high. These predictions were made using vanilla transformed reflection counts.

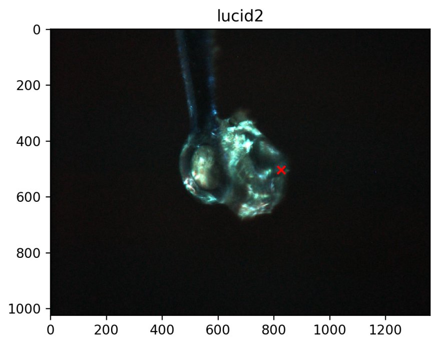


Figure 2.20: The red x shows the result from lucid2 trying to find the loop in the image.

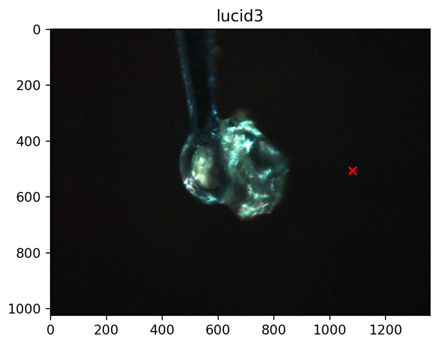


Figure 2.21: The red x shows result from lucid3 trying to find the loop in the image.

> Lucid 2.0 is an image processing software that detects the target of Synchrotron ESRF and returns its coordinates in an input image.

The target being referred to is the loop that the crystals are attached to. Lucid is developed for usage in ESRF (European Synchrotron Radiation Facility, located in Grenoble, France) and therefore we might not be as reliable at finding the loop at MAX IV since it was developed assuming the setup at ESRF. We can see this in Figure 2.21.

Potential error sources

Since all of our data needed to be collected manually, and the time to do so was quite limited, our data set is not as big as we would have liked. This coupled with the fact that we needed to divide the data set along crystal boundaries (so no crystal would be in both the training set and the testing set), meant that only a handful of crystals were in the test set and validation set. So a model that happens to be well suited for those particular crystals would receive an unfairly low MSE. This problem is made worse by the fact that images from a helical scan (described in Section 2.1.3) will be very similar to each other, so the test set is not very diverse.

If we look at the heatmaps of, for example log+a2 and log+a10 in Figure 2.16, and tanh+a7 in Figure 2.17, there seems to be an indirect correlation between the bright parts of the image and a high predicted value. It is possible that for some of our models, all it's doing is looking for bright blobs in the image since these are likely to be glare from the crystal. If this is the case, the model could fail when presented with images with glare on the holder.

2.2 Approach II: Classification

To narrow down the problem and see if the model can learn the most simple features we wanted to try binary classification. If the classification is a success we can assume that the data set have learnable features and there might be some other problem for regression. The labels were *empty* (empty loop) and *crystal*. That is, determine if we can see a crystal or not. The input to the CNN is an image and the output is the probability for the classifying the image as label *empty* in the range $[0, 1]$. We use the same tools here as in Approach I. See subsection 2.1.2 at page 19

2.2.1 Data set

The data set encompasses crystals from the previous data set. Some crystals were removed due to difficulties in recognizing if it actually was a crystal or just an empty loop. The reason for this was to get a good data set as possible to minimize the possibility that the data set was a source of error. One more data collection was performed to collect images with empty samples.

Due to not having any crystal there was no need to do a helical/rotational-scan, and therefore the images collected did not have any movement, i.e. many images with the same loop looked the same. To increase the data set and cope with an overabundance of crystal images, data augmentation was performed on selected images with empty loops. The images were selected so that each image was easy to distinguish from the others, either by zoom level or the characteristic of the loop. Crops were generated from each image in 2.22. This was done by sliding a window over the image and saving each crop as a new image. Splitting the data set was done in the same fashion as for regression. The final size of the data set was 4468 images.

2.2.2 Transformation and augmentation

The images were transformed in the same way as the images in regression, see section 2.1.4. Labels were encoded to $0/1$ in alphabetical order i.e. *crystal*=0, *empty*=1.

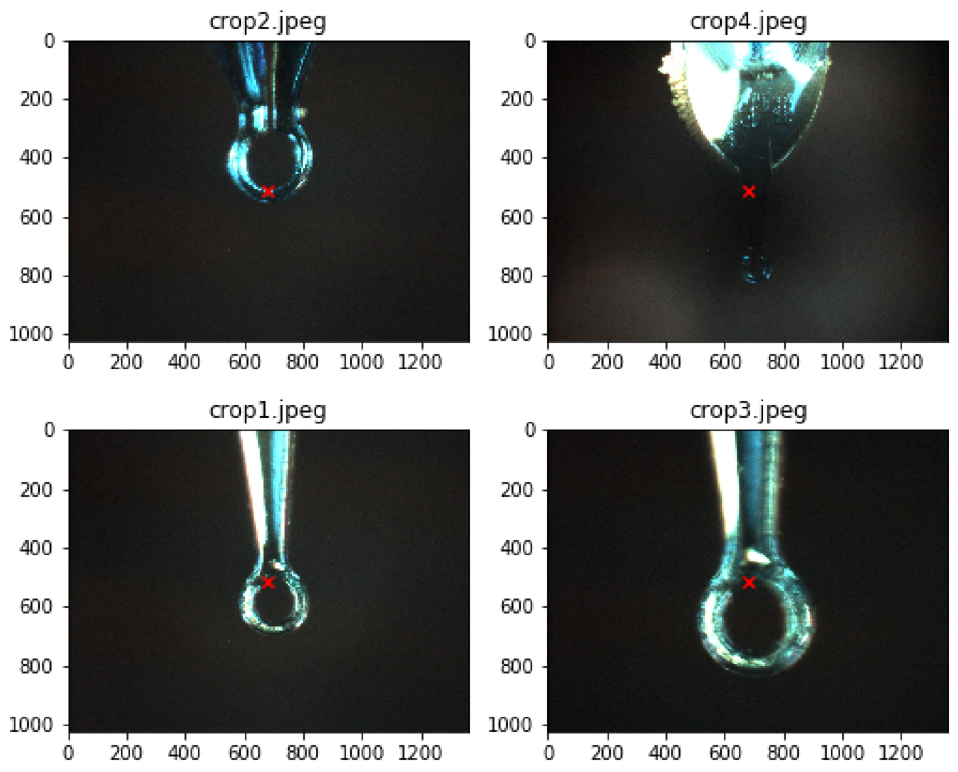


Figure 2.22: Loops without any crystal that are used for classification.

2.2.3 Model

Only one architecture was used since this approach was used to determine if simple features could be detected. The architecture was inspired by models used for other kind of binary classification such as one at Keras [25] and one at Towards Data Science [26]. Loss function was binary cross-entropy see previously defined equation 1.6 at page 14.

C*N*	Number of convolutional layers with *N* filters of size 3×3
MPOOL	Number of max pooling layers [2]
F*N*	Number of fully connected layers with *N* neurons
DROPOUT*N*	Number of dropout layers with *N*% dropout
LOSS	Loss function
ACC	Prediction accuracy

Table 2.4: Keys for the Table 2.5

C32	MPOOL	C32	MPOOL	C64	MPOOL	F64	DROPOUT50	LOSS	ACC
1	1	1	1	1	1	1	1	0.2604	0.891

Table 2.5: Result of training on binary labeled data set

2.2.4 Result

The results are determined from evaluating from predicting on the test set. Figure 2.23 is a confusion matrix visualizing the number of correct/incorrect predictions for each class. Header *ACC* in table 2.5 shows the percentage correctly predicted.

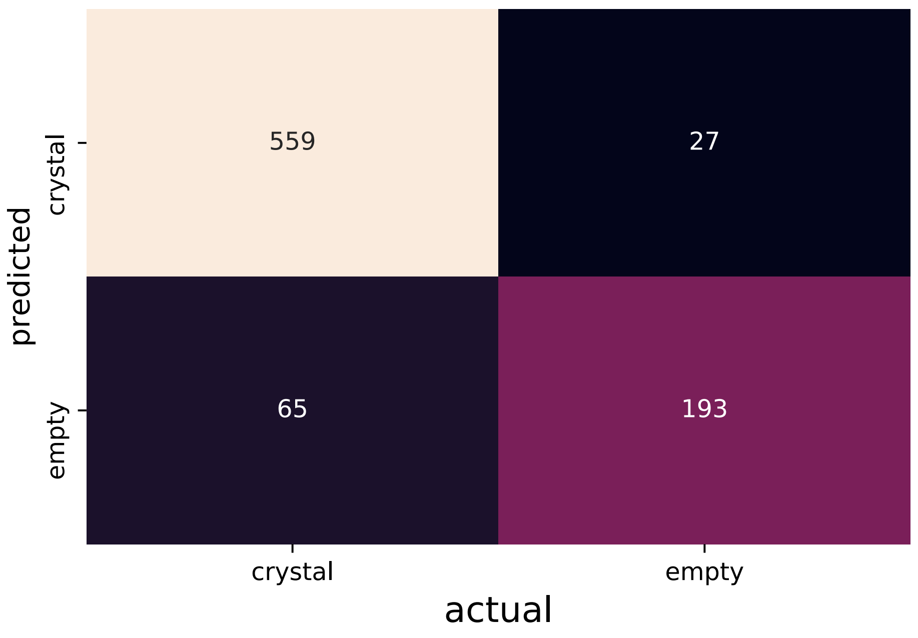


Figure 2.23: Confusion matrix showing the result of predicting on the test set.

2.2.5 Discussion

In this approach we have tried to classify images to one of two classes: *crystal* and *empty*. We did this by training a CNN with an architecture inspired by other architectures used successfully. We show that the CNN learns to classify the images with an accuracy of $\sim 89\%$. This shows that a CNN can learn features from our data set and deep learning can be a viable method for regression.

The data set is handpicked to have the most distinctive images which is why we expect the network to achieve a high level of prediction accuracy. The low amount of training data will probably cause overfitting, and the network might not predict with the same accuracy on real-life data. This is however not important in our case and is nothing we investigated further, as this was just a way to see if we were on the right track with the network architecture and our data set.

Due to the way data set splitting is done, the number of empty samples is drastically less than crystals. The reason for this is that the number of crystal images varies from sample to sample.

Chapter 3

Conclusion

In this thesis, we have used neural networks to evaluate centerings of crystals before running the experiments. In the correlation figures 2.10, 2.11, and 2.12 from approach 1 we can in many cases see that the predictions are following the ideal target, although not that precise. This shows that using the raw crystal images as input, we can get an idea of how good the centering is. This mostly applies to crystals with a high reflection count since we found it difficult to predict the outcome from low-quality crystals due to the similarity in appearance.

To try to increase the accuracy of our models, we have used three different transformations of the reflection count. In the correlation figures we can see how they compare with each other and the effect they have. There are some limitations to our current state of the models. Predicting pixel-precise reflection counts or even correct reflection count is not possible with our training data and models.

There is still major work to implement this in production at BioMAX. There will be technical challenges in the integration and improvements to the model is necessary. We are hopeful that our work will be of use, if not directly but indirectly.

Future work

As mentioned before, this is probably not the most efficient or optimal way of automatically centering crystals. The architectures used here are semi-arbitrarily chosen. It would be interesting to try to use some well known, high performing architectures in place of our a1-10 architectures. Lots of these are conveniently available in Keras (`https://keras.io/`

`applications/`) [13].

The fact that different crystals can produce different amounts of reflection count is a problem. It means that RC is not a direct measurement of how well the beam was aimed, one crystal could produce the same number of spots when hit perfectly as another crystal produces when hit on the edge. It might be beneficial to look for some quality measures other than the reflection count. Perhaps combining the reflection count with one or more measures of the crystals quality as a whole, i.e. its capacity for producing reflections.

Another thing that could be improved is how we match camera images with reflection counts. The current method of using timestamps seems very ad hoc and unreliable.

We might also be able to produce better results by combining this with other techniques for locating an object in an image, such as segmenting the image into the crystal and not crystal before doing regression analysis as Y. Muira *et al* [8] might improve results since the entire focus will be on finding the best spot in the crystal, and not having to find the crystal itself.

A different way of acquiring data might be worth looking into. Our data were collected by doing a helical scan or a rotational scan. Mesh scan is a type of scan where the user draws a 2-dimensional grid over the area of interest. This was not available during our time but might be implemented later. This way we could more directly compare our heatmaps to real measurements.

Increasing the size of the data set is another way that is straightforward to improve the result. Specifically collecting data from a larger number of crystals. This could help with our problem of being unable to evenly divide our data set into train/test/validate subsets.

To use this in practice, one could first use lucid to guess the target and then create a heatmap around that area. Depending on their size and density, heatmaps could take a few seconds to be generated. If we wanted to do this in real-time while a user is examining a crystal, with a heatmap as an overlay, this could speed up that process, since the heatmap would not have to cover the whole camera image.

Bibliography

[1] "Md3 high precision x-ray microdiffractometer." `https://www.arinax.com/md3-high-precision-x-ray-microdiffractometer`. Accessed: 2019-3-27.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[3] Fei-Fei Li et al, "Module 1: Neural networks, neural networks part 1: Setting up the architecture," in *CS231n Convolutional Neural Networks for Visual Recognition*, 2019.

[4] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[5] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*, pp. 92–101, Springer, 2010.

[6] T. Shimobaba, T. Kakue, and T. Ito, "Convolutional neural network-based regression for depth prediction in digital holography," *CoRR*, vol. abs/1802.00664, 2018.

[7] M. Yann and Y. Tang, "Learning deep convolutional neural networks for x-ray protein crystallization image analysis," 2016.

[8] Y. Miura, T. Sakurai, C. Aranha, T. Senda, R. Kato, and Y. Yamada, "Classification of x-ray protein crystallization using deep convolutional neural networks with a finder module," *CoRR*, vol. abs/1812.10087, 2018.

[9] "Esrf - lucid 2, technical note." `https://github.com/mxcube/lucid2/blob/master/doc/Lucid2TechnicalNote.pdf`. Accessed: 2019-4-02.

[10] "lucid3: Loop and ucrystals identification version 3." `https://github.com/mxcube/lucid3`. Accessed: 2019-3-27.

[11] O. Svensson, S. Malbet-Monaco, A. Popov, D. Nurizzo, and M. W. Bowler, "Fully automatic characterization and data collection from crystals of biological macromolecules," *Acta Crystallographica Section D*, vol. 71, pp. 1757–1767, Aug 2015.

[12] "Opencv: Laplace operator." `https://docs.opencv.org/4.1.0/d5/db5/tutorial_laplace_operator.html`. Accessed: 2019-7-09.

[13] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[14] J. Howard *et al.*, "fastai." `https://github.com/fastai/fastai`, 2018.

[15] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[16] U. Mueller, M. Thunnissen, J. Nan, M. Eguiraun, F. Bolmsten, A. Milàn-Otero, M. Guijarro, M. Oscarsson, D. de Sanctis, and G. Leonard, "Mxcube3: A new era of mx-beamline control begins," *Synchrotron Radiation News*, vol. 30, no. 1, pp. 22–27, 2017.

[17] "Eiger x detectors for synchrotron." `https://www.dectris.com/products/eiger/eiger-x-for-synchrotron`. Accessed: 2019-3-27.

[18] The HDF Group, "Hierarchical Data Format, version 5," 1997-2019. http://www.hdfgroup.org/HDF5/.

[19] G. Winter, D. G. Waterman, J. M. Parkhurst, A. S. Brewster, R. J. Gildea, M. Gerstel, L. Fuentes-Montero, M. Vollmar, T. Michels-Clark, I. D. Young, N. K. Sauter, and G. Evans, "DIALS: implementation and evaluation of a new integration package," *ACTA CRYSTALLOGRAPHICA SECTION D-STRUCTURAL BIOLOGY*, vol. 74, pp. 85–97, FEB 2018.

[20] A. Ng, "Model selection and train/validation/test sets - advice for applying machine learning | coursera." `https://www.coursera.org/lecture/machine-learning/model-selection-and-train-validation-test-sets-QGKbr`. Accessed: 2019-7-08.

[21] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[22] D. F. Specht, "A general regression neural network," *IEEE transactions on neural networks*, vol. 2, no. 6, pp. 568–576, 1991.

[23] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8609–8613, May 2013.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[25] F. Chollet, "Building powerful image classification models using very little data." `https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html`, 2016. Accessed: 2019-6-25.

[26] G. Surma, "Image classifier - cats vs dogs." https://towardsdatascience.com/image-classifier-cats-vs-dogs-with-convolutional-neural-networks-cnns-and-google-colabs-4e9af21ae7a8, 2018. Accessed: 2019-6-25.

[27] "eiger2cbf." https://github.com/biochem-fan/eiger2cbf. Accessed: 2019-4-09.

Appendices

Appendix A

Instructions for counting reflections with DIALS

1. Run *eiger2cbf* [27] on the HDF5 master file from EIGER detector.

   ```
   ./eiger2cbf filename.h5 N:M out.cbf
   ```

   N is start frame, ***M*** is last frame.

2. Run *distl.signal_strength* on each cbf file to get characterization of candidate Bragg spots. See appendix B for an example output.

   ```
   distl.signal_strength filename.cbf
   ```

   In the output we can see *Spot Total* which is our RC. The binary *distl.signal_strength* is found in the DIALS project. [19]

Appendix B

DIALS output

```
distl.signal_strength: characterization of candidate
   Bragg spots
#Parameters used:
#phil __ON__

distl {
  image = "/data/staff/common/ML-crystals/real_cbf
     /20181214/Sample-3-07/20180479_1_master/out000137.
     cbf"
  res {
    outer = None
    inner = None
  }
  verbose = False
  dxtbx = False
  bins {
    verbose = False
    N = 20
    corner = True
  }
  range = None
}
autoindex_override_beam = None
autoindex_override_distance = None
autoindex_override_wavelength = None
```

```
autoindex_override_twotheta = None
autoindex_override_deltaphi = None
image_specific_osc_start = None
codecamp {
}
pdf_output {
}
distl {
  minimum_spot_area = None
  minimum_signal_height = None
  minimum_spot_height = None
  spot_area_maximum_factor = None
  peak_intensity_maximum_factor = None
  method2_cutoff_percentage = 20
  compactness_filter = False
  scanbox_windows = 101 51 51
  peripheral_margin = 20
  pdf_output = None
  port = 8125
  processors = 1
  nproc = 1
}

#phil __OFF__

MAX-IV Eiger 16M

                     File : out000137.cbf
               Spot Total :   1707
               Remove Ice :   1610
      In-Resolution Total :   1543
    Good Bragg Candidates :   1313
                Ice Rings :      6
      Method 1 Resolution :   1.60
      Method 2 Resolution :   1.52
        Maximum unit cell :  153.7
<Spot model eccentricity> :   0.790
%Saturation, Top 50 Peaks :   1.46
In-Resolution Ovrld Spots :      0

Bin population cutoff for method 2 resolution: 20%

Number of focus spots on image #137 within the input
   resolution range: 1543
Total integrated signal, pixel-ADC units above local
   background (just the good Bragg candidates) 927143
```

```
Signals range from 14.3 to 18020.4 with mean integrated
   signal 755.7
Saturations range from 0.0% to 3.0% with mean saturation
   0.1%
```

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2019-06-11

EXAMENSARBETE Crystal Centering Using Deep Learning
STUDENTER Jonathan Schurmann, Isak Lindhé
HANDLEDARE Jörn Janneck (LTH)
EXAMINATOR Flavius Gruian (LTH)

Can we teach a neural network to aim an X-ray at a crystal without completely frying it?

POPULÄRVETENSKAPLIG SAMMANFATTNING **Jonathan Schurmann, Isak Lindhé**

One of the most tedious tasks of a crystallographer at the MAX IV laboratory is to manually center crystals in the beam. How can we harness machine learning to replicate the crystallographers' intuition for this boring task?

At the BioMAX research station in the MAX IV laboratory, very bright x-ray beams are fired at tiny tiny crystals to see how the beam diffracts in them. To get good results you need to hit specific spots. Most radiation facilities are shooting the crystal holder all over and are taking the best result.

MAX IV, having the brightest X-ray source in the world at the time of writing, can't really do that, because exposing the crystal to the beam for that long would inflict severe radiation damage and might even make the crystal explode. So researchers at BioMAX need to spend their valuable time painstakingly centering the crystals manually. **So can we improve this?**

Yes! What if we can use the same "shotgun approach" as other synchrotrons without having to fire? We taught a convolutional neural network to recognize what an image of a well-centered crystal looks like in comparison to a poorly centered one. Using the number of reflections in the resulting diffraction patterns (quality of the diffraction) as labels for the dataset we can learn our network to "understand" the crystals. The label corresponds to the result of shooting in the center of the image. With our trained model, all we need to do is crop the image around different parts and for each ask the model *"Is this centered? This one? What about this one?"*

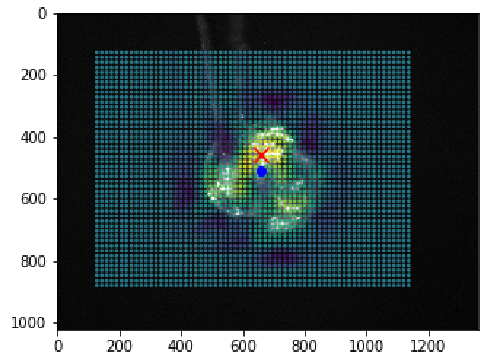


Figure 1: A crystal help onto a loop by sheer surface tension

Here is the resulting heatmap, using one of our better models. The color of each point is determined by how much the model thinks it the crystal is centered at that point. The red X is the point where that the model is the most certain is centered, and at least for this crystal it appears to be quite right.