A Machine Learning based Processing technique for X-Ray Emission Spectroscopy Detector Images

Report for the DESY Summer School Project

made by

Lukas Bödeker

in August 2019

Supervisor: Dr. Aleksandr Kalinko

Contents

1 Introduction to the problem and overview

This project is related to the beamline P64 for advanced X-ray emission spectroscopy of the PETRA III synchrotron at DESY, Hamburg [1]. In particular it deals with detector images from a dispersive von Hamos spectrometer that is used at the beamline [2]. The spectrometer is illustrated in figure 1.1
The physical setup of spectrometer leads to 6 or 8 emission lines according to the number of used analyser crystals. So the one should find correspondingly 6 or 8 peaks, aligned vertically for one diffracted energy. This alignment is not perfect in the practical execution as can be seen in figure 1.2
The consequence is that one must correct the x position of the peaks in the images with respect to each other for the realignment. As well one must also perform an energy calibration $x \leftrightarrow E$ (using the measured E for elastic scattering lines).
To overcome this it is natural to search for the peak positions. At the moment this is done classically by projecting the detector image (or a part of it) to one axis and find peaks in the graph corresponding to certain parameters. For this method it turned out that the peak finding parameters are not universal and must be adjusted for every new measurement.
It would be much nicer for the data taking process to have the correction of peak positions and the calibration in real time.
This circumstance leads to the idea to make the peak finding more universal and stable by using machine learning considering the fact that the peak shapes are always comparable. Due to the fact that especially (convolutional) neural networks (NNs) have proven it's great potential of dealing with data in form of images in the last decades this approach seems to be promising. This NN aproach should also be able to satisfy the condition of a fast procession.
Sequentially the task for this project is to analyse, whether NNs can contribute to the peak detection task.

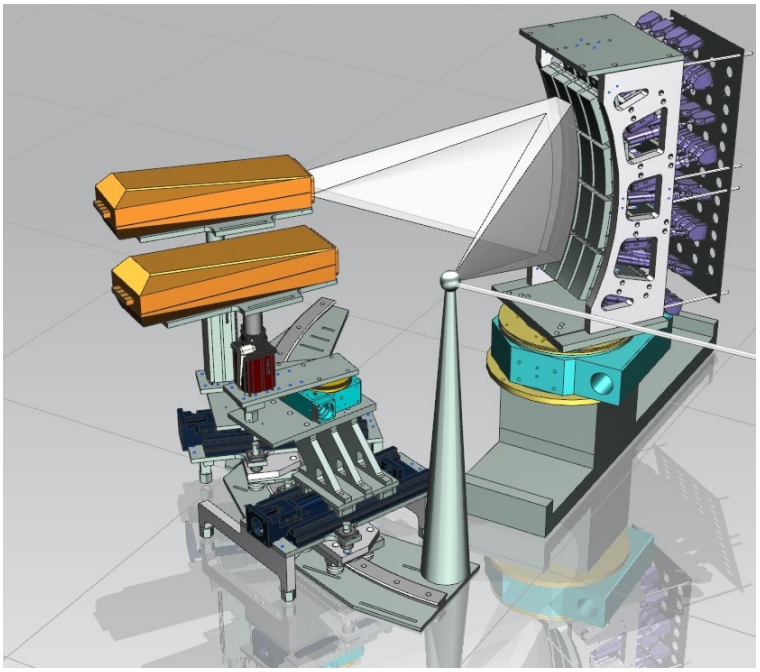


Figure 1.1: 3D sketch of the high resolution von Hamos-type x-ray emission spectrometer.[2]

Therefore this report will explain the data that is used to train the network at first. I will use simulated detector images due to the reason that the true peak positions are needed for the training. The discussion of this simulation is topic of this part.
In the second part, I will discuss the specifications of the NN that led to the smallest statical deviation of the prediction to the true values.
Further it is important to see which network approaches and configurations did not give a descend result to support continuing work at this task.

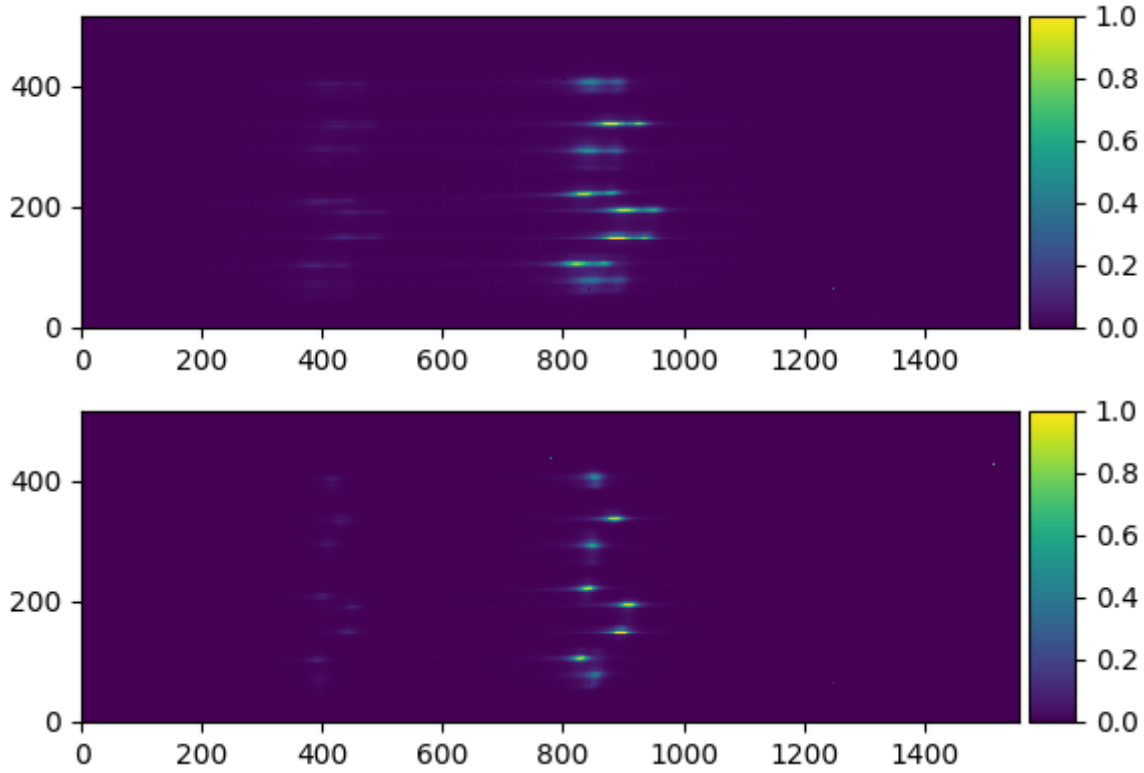


Figure 1.2: Real detector images from the von Hamos type spectrometer with 8 crystals with 1556x516 pixels.

2 Training Data

It is convenient to use simulated data to train NN in many application. In this case it is reasonable to assume that the simulated images could imitate the real detector images to an arbitrary extend.
At first I tried to model the main features of the images. In this case, the problems appear in the step of generalisation from simulated to real images the solution would be to modify the simulation by adding finer features and retrain the model.

For the investigation of a good NN I simulate images corresponding to 8 silicon crystals with a size of 1500 x 700 pixels. If this can be done successfully I should be able to scale the NN easily to train corresponding models. This work will be restricted to this example.

2.1 Specifications of the simulation

To simulate the detector images the image generator sets two dimensional Gaussians to an empty array of pixels in a sensible way compared with figure 1.2. The result should be to have 8 strong peaks and 8 weaker peaks (referred to as right and left) which are respectively poorly aligned in the vertical direction.
To describe the random distribution I use the notation $X \sim \mathcal{N}(\mu, \sigma)$ and $X \sim \mathcal{U}(a, b)$ for the normal, respectively uniform distribution of the variable X.

To simplify the up scaling of the NN correspondingly to an image I use relative values to the image dimensions which will be called A x B (=1500 x 700). Further N_s (=8) denominates the number of silicon crystals, which is also a variable magnitude.

Generally, the generator produces the simulated images in tranches of N_{im}=1000 images per file what leads to a file size of about 4GB. The calculation time per image is about 2.2s.

2.1.1 Specification of the locations of the Gaussians

The aim of the setting of the coordinates (x, y) of the peaks in first estimation, is to follow a certain global patterns as the poor alignment in y or the typical distance of the right to the left peak column etc. (compare fig. 1.2).
It is desired to distribute the peaks roughly equal in y direction and to have full diversity of the location of the peak columns in x direction among all images. The concrete distribution of the y coordinate is the flowing:

$$y_{i+1} \sim y_i + min\Big[\frac{95\%}{N_s} * B; \mathcal{N}\Big(\frac{85\%}{N_s} * B, \frac{20\%}{N_s} * B\Big)\Big] \; i \in \{1, 2.., N_s\}$$

For the x direction the image generator has to consider the right peaks and the left peaks. These x coordinates are different for each of the 1000 images in one file in a way that all possible locations of the peak columns are represented in one file.

$$\begin{aligned} x_j^l &\sim \mathcal{U}\Big(\Big(15\% + \frac{50\% * j}{N_{im}}\Big) * A, \Big(16\% + \frac{50\% * j}{N_{im}}\Big) * A\Big) \\ x_j^r &\sim \mathcal{U}\Big(\Big(40\% + \frac{50\% * j}{N_{im}}\Big) * A, \Big(41\% + \frac{50\% * j}{N_{im}}\Big) * A\Big) \\ j &\in \{1, 2.., N_{im}\} \end{aligned}$$

2.1.2 Specification of a single Gaussian

The mean $\vec{\mu}$ and the standard deviation $\vec{\sigma}$ of each 2D Gaussian is chosen to be normal distributed around the previous generated coordinates.
The following distributions are to be given:

$$\begin{aligned} \mu_x^{r/l} &\sim \mathcal{N}(x^{r/l}, 2\% * A) \\ \mu_y &\sim \mathcal{N}(y, 0.5\% * B) \end{aligned}$$

$$\begin{aligned} \sigma_x^r &\sim \mathcal{N}(2\% * A, 6\% * A) \\ \sigma_y^r &\sim \mathcal{N}(1.5\% * B, 2\% * B) \\ \sigma_x^l &\sim \mathcal{N}(1.5\% * A, 4\% * A) \\ \sigma_y^l &\sim \mathcal{N}(0.7\% * B, 1\% * B) \end{aligned}$$

The heights $h^{r/l}$ for the left and right peaks are chosen in a way that for each right peak, the left peak has a decreased intensity by the order of 10%. This shall give a direct correlation.

$$h^r \sim |\mathcal{N}(2,1)|$$
$$h^l \sim h^r * \mathcal{U}(8\%, 15\%)$$

Furthermore the generator considers a possible correlation ρ close to 0.

$$\rho \sim \mathcal{U}(-0.1, 0.1)$$

Therewith the resulting 2D Gaussian is given by the expression:

$$f(\vec{x}) = \frac{h}{\sqrt{(2\pi)^2\, det(\Sigma)}}\, exp\Big(-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu})\Big)$$

$$\Sigma := \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$$

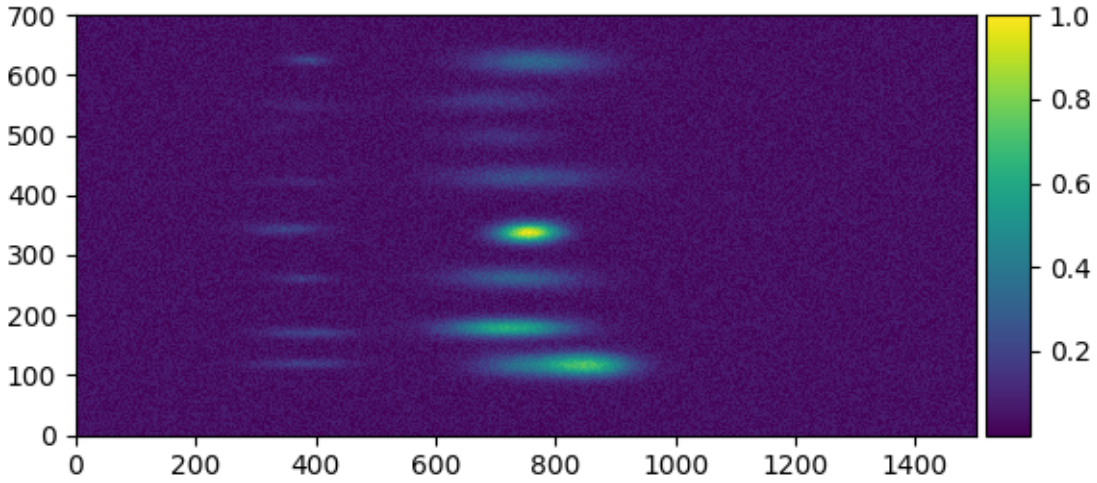
2.1.3 Adding noise

A random array of noise that is of the order of the left peaks is also added to the image:

$$Noise \sim \mathcal{U}\Big(0, 10\% * Max(Image)\Big)$$

Furthermore in the experimental images one can find secondary foci that are displaced from the main peak. Because of this feature could lead to misclassification, it will be simulated as well. For each peak the probability to have a secondary focus is set to 25%. The second focus is centred in the x direction around $x + \mathcal{U}(0.5\% * A, 0.7\% * A)$. The rest of the properties of the new Gaussian are as described before, beside of the hight of the sub peaks $\tilde{h}$:

$$\tilde{h} \sim h * \mathcal{U}(20\%, 50\%)$$



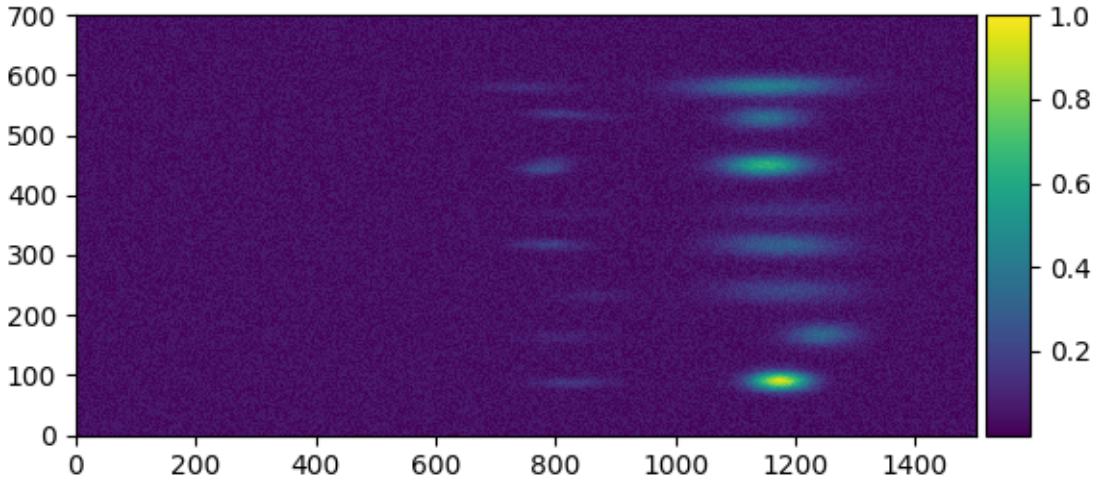


Figure 2.1: Example for generated images.

2.2 Result and conclusion

Considering the presented model one can draw some conclusion for the NN and by comparing the product of the simulation in figure 2.1 visually with the detector image in figure 1.2 one can discuss possible problems and how to solve them.
In general the training images, produced in the described way are considered to match with the real images in terms of the degree of difficulty for the NN to predict the peaks. Accordingly a prove of principle on this data should hold for variations of the image simulation as well.
Furthermore I have to state that I tried to make a very convoluted construct of random variable generations to make it hard for the NN to analyse and simply adept the resulting probability distribution for the images. This distribution of the whole image should not simply be learned by the network, because that would make the generalisation to real data very hard. Instead the network shall ideally learn certain local shapes in the image that lead then to the peak prediction. This fact will be one of the reasons for the use of convolutional layers as explained in the follow.

2.2.1 Generalisation problems

The biggest difference by eye for the shape of the real peaks in comparison with the simulated ones is that real peaks can be asymmetrical with respect to a normal Gaussian. One observes that for example in y direction the distribution to one direction (up/down) can be very diffuse and to the other direction quiet sharp. Another noticeable feature of the real image is that the area between the right and left peaks is stronger illuminated then the rest. There is often a horizontal line between the peaks.
It can not be excluded that this or other differences in the peak shape disturb the NN and cause worse performance on real data than it was seen before.
For this case it is recommended to use for example logarithmic Gaussians instead, which have a finite skewness that can be tuned.
This was not done initially to keep the computation time low.
Additionally it still holds that the approach of the use of NNs can be tested non regarding changes in the simulation as stated before.

2.2.2 Expectation for the neural network

By analysing the simulated images I can derive certain thresholds for the prediction precision which the NN has to overcome.

A criterion that is crucial for the prediction is the stable separation of two neighbour peaks in y direction. To quantify that, I demand that: $3\sigma_{prediction} <$ distance of two neighbour peaks in y direction. That corresponds to a probability of misidentification in one image is $(1 - 99,73\%) * 16 \approx 4\%$.
Now I estimate a rather save relative minimal distance D of the peaks. Therefore we have the knowledge about the distribution of the peaks of section 2.1.1 and 2.1.2:

$$D \approx \frac{85\%}{8} - 2 * \frac{20\%}{8} - 2 * 0.5\% \approx 4.6\%$$

For this estimation it was taken a shift of 2σ from the mean value.
Finally, I find as the condition for the mean prediction error (relative to the dimension):

$$\sigma_{prediction} \overset{!}{\lesssim} 1.5\% \underset{1500\text{x}700}{\approx} 11\,\text{pixel} \tag{2.1}$$

3 Results

This part will present the results of the search process for a good structure and hyper parameters of the NN. The basic proposition will be the excluded parameter space that is found (and the corresponding conclusions).
The systematic search is often done in grid searches. In such a search the NN is trained several times with differing parameter configurations that are taken from a previously defined pool. One will not describe every grid search in detail. Instead the following part will be structured by the elements and aspects of the NN, for which certain statements could be concluded by the analysis of the searches.

3.1 Global parameters

There are parameters which are either set by the problem or conveniently selected in a global way. Other global parameters are found by testing different configurations at the very beginning. Ther are assumed to be rather independent form other conditions.
That means that this parameters are not part of the further grid searches and variations when they are set once.
One has to mention as a very global parameter, that the NN is set up in Keras with tensorflow (1.1.13) backend.

3.1.1 Input and output

At first one would assume that the most fixed parameter are the size and type of the input to the NN and the output. Elementary, the input for this problem is a 1500x700x1 array of pixel values and the output are the 16 2D coordinates of the peaks ($\rightarrow$ 32 values).

Intensity regression: Additionally one could also take the denoised intensity of each peak as a target to train the NN on. That would increase the number of regression targets from 32 to 48.
In principle this would be a useful feature, but it was seen at the beginning that the

condition 2.1 is not to be reached easily. Therefore the intensity regression was quitted to make the problem easier. One can consider as well to fit a Gaussian with the regressed coordinates as initial guess to find a good intensity value.

Preprocessing: One has found during the searching process that the regression performs better when the magnitude of the gradient of the image is provided to the regression as well. That this information can be useful is somehow clear because the gradient vanishes at each maximum non regarding the intensity of the peak, which is initially provided only.

Normalisations: Initially the regression target are 32 values in units of pixels and so in the range for $x, (y) \in [0, 1500\,(700)]$. It is convenient for NN approaches not to deals with such big numbers due to the problem of exploding gradients and the need of much regularisation (exploding weights). Following this general rule one will regress the relative coordinates: $x, y \in [0, 1]$.
It is to mention that the regression is tested with absolute values as well. Following this approach the NN is much harder to train as expected.
The circumstance that NNs are not good in dealing with big numbers leads as well to the normalisation of the input images by dividing all pixels by the pixel value with the highest intensity. On this way each input pixel is as well in the interval $[0, 1]$.

3.1.2 Loss function

It is very convenient to use the squared absolute distance of the prediction and the true value or modifications of it (e.g. Huber) for regression tasks. That is due to the fact that the parameter update then is proportional to the absolute distance.
In view of the data simulation one must not fear very exotic data (outliers), so there is no need to use the Huber loss for example.
Later one will see that a sigmoid function is used for the output layer. Normally they harmonise with logarithmic loss functions to prevent a learning freeze out. That happens when the sigmoid approaches 0 or 1 because in this regime its derivative vanishes. Because here the sigmoid will only be used at the output the values can not be as extreme due to the fact that the peaks never lie on the edge of the image. In conclusion the squared absolute distance loss combined with the sigmoid in the last layer should not cause learning problems. Instead should the sigmoid function help to get a faster training result because the output values lay automatically in the the range of possible coordinates.

One has to notify that this loss will lead to the same relative error for the x and y prediction. I could also try to equalise the absolute errors by weightening the contribution of x and y to the loss. I do not do that do that because it is not bad to predict the vertical position better than the horizontal one because of the arrangement and shape of the peaks.

The condition 2.1 can now simply be expressed by the loss as well: $loss \lesssim 0.00023$.

3.2 Convolutional NN

The use of convolutional layers are the key to the success of machine learning in image analysis (mostly classification). That can simply be reasoned by the fact that not every pixel of an image is important to determine the output prediction and a full connection of the image pixels to a dense layer would so lead to a big amount of parameters that are not needed.
The idea of a convolutional neural network (CNN) is to let the network learn to abstract features/shapes of the image by using filers. This leads to a translation the information coded in the spatial image dimension to a continually growing feature dimension. Those features are then given to a fully connected network that processes the information further.
It is clear that well abstracted information about the existence of various shapes in the image easily leads in a classification task to good results (non regarding the location). For a spatial regression not only the existence but the location of a special shape must be extracted. That is a much more complex task.

3.2.1 Convenient CNN approach → layer structure

As a starting point, I analyse the NNs, used in [3] and [4] that describe similar problems. They are solved successful respectively, in spite of the fact that e.g. for [3] a relative error of 8% could be achieved. This error level could not be considered as good given the criterion 2.1. Generally this examples encourage that descend results in image regression tasks can be achieved by CNNs.
Nevertheless, the described networks as in figure 3.1 are tested in an upscaled version (to fit 1500x700).
The NN configurations, given on this way are generally convenient structures for a CNN if one compares them with famous image processing CNNs, just in a downscaled variant.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d 1 (Conv2D)            (None, 126, 126, 64)      2368

conv2d 2 (Conv2D)            (None, 124, 124, 64)      36928

conv2d 3 (Conv2D)            (None, 122, 122, 64)      36928

max pooling2d 1 (MaxPooling2 (None, 61, 61, 64)        0

batch normalization 1 (Batch (None, 61, 61, 64)        256

dropout 1 (Dropout)          (None, 61, 61, 64)        0

conv2d 4 (Conv2D)            (None, 59, 59, 128)       73856

conv2d 5 (Conv2D)            (None, 57, 57, 128)       147584

conv2d 6 (Conv2D)            (None, 55, 55, 128)       147584

max pooling2d 2 (MaxPooling2 (None, 27, 27, 128)       0

batch normalization 2 (Batch (None, 27, 27, 128)       512

dropout 2 (Dropout)          (None, 27, 27, 128)       0

flatten 1 (Flatten)          (None, 93312)             0

dense 1 (Dense)              (None, 256)               23888128

batch normalization 3 (Batch (None, 256)               1024

dropout 3 (Dropout)          (None, 256)               0

dense 2 (Dense)              (None, 256)               65792

batch normalization 4 (Batch (None, 256)               1024

dropout 4 (Dropout)          (None, 256)               0

dense 3 (Dense)              (None, 4)                 1028
=================================================================
Total params: 24,403,012
Trainable params: 24,401,604
Non-trainable params: 1,408
_________________________________________________________________
```

Figure 3.1: Layer structure for a spatial regression of two points in a 2D image as discribed in [3].

parameter	values
#convolutional blocks	1,2,3
#convolutional layers in one block	1,2
#convolutional depth	40,60,80,120
#dense blocks	1,2,3
dense size	128,256,512,1024

Table 3.1: Structural parameters of the grid search based on the convenient convolutional approach. Convolutional depth means the depth a for example of an image 1500x700xa, after passing all convolutions by not decreasing the spatial size. The depth increases gradually after each convolution step.

The NN consist of blocks of convolutional layers combined with pooling stages afterwards. After the convolutional part, the rest image is flattened and passed to fully connected layers.
One could not expect that the just upscaled network works directly for this problem. Subsequently I perform a vast grid searches in architectural terms like number of convolutional blocks/kernel size/number of covolutional layers/ number and size of dense layers and also on hyper parameters like the learning rate and dropout. More detailed conclusions of this search concerning special architectural elements will be given in the following paragraphs. For now it is important to see that search was done with the concrete parameters in table 3.1. In total this search lead just to a $loss \gtrsim 0.03$.
The search also leads to the result that smaller networks seem to perform better.

Conclusion: Considering the rest of hyper parameter space covered I conclude that the tested model structure is overloaded. That can be explained by the fact that the images which are treated normally by a CNN have much more diversity in terms of shapes and edges that can be decoded by the CNN. In this case the lack of complexity in the images leads probably to noise and therewith an error level that is far from acceptable.
In conclusion it will not be expedient to focus further on the convolutional part of the network. That means that it is not promising to tailor complex convolutional structures due to the given arguments.
With this insight on must see as well that this means the loss of a tool that has shown to be very useful for image procession using NN in the past.

3.2.2 Pooling and desizing

A detail that was investigated in other searches was the influence of the use of (max-) pooling layers after the convolution to decrease the size of the images and gain a better ability of the NN to be trained. In usual image related problems this has additionally the effect the CNN can through out features that are not useful.
A competing method to pooling is the use of stride in the convolutional layers. This acts less harsh on the images and might theoretically prevent the loss of spacial information.
It has been tested several combinations of applying max pooling and/or stride in the convolution to achieve different orders of image sizes that then are flattened to be passed to the dense layers.
The outcome is that desizing never increases the goodness of the prediction.
By quitting on the downsizing of the image, the NN will contain at least 1500x700x2x32$\approx 67 \cdot 10^6$ parameters.
In principle this result is understandable because of the link to a decrease of spatial resolution and as well because the networks task is it at the end to translate a specific position in the array to a float number. This process cannot be supported by changing the size of the image constantly.

3.2.3 Kernel size

Size: To study the influence of the kernel size of the respective convolutional part in particular I perform a search of different sizes for a simple network of just one convolutional layer with a depth of 2 and no stride and a dense layer with the width of 64 before the output layer. Networks of this simplistic architecture will turn out to be a quiet good choice. NN of this type will be referred to generically as simple networks in the follow.
I tire out different filter sizes with different aspect ratios as well. In particular the tested Kernel sizes reach from $(5, 5)$ to $(300, 100)$ with aspect ratios from $1:1$ to $6:1$ (due to the image aspect ratio of $2.1:1$).
The outcome is that filters in the size range of 30^2 till 80^2 perform the best and can not be found any difference in performance in this interval. The aspect ratio seems to not have any infuence to the outcome as well. Bigger and smaller filters worsen the error on the other hand.

Sequence: Considering the use of more than one convolutional layers, the question occurs whether the kernel of the following layer should be bigger or smaller than the

kernel of the previous layer. In the normal CNNs the size increases gradually, reasoned by the idea to find first smaller features, which than can be combined by bigger filters. This must not hold for this problem due to the fact that detector images do not have as many shapes as usual images. One could imagine as well that big filters at the beginning help the CNN to decode the more global topology to go into more detail afterwards.
The study of this was done as before in a simple network with now two convolutional layers with increasing or decreasing kernel size relative to each other.
I find that the consequences of an increasing or decreasing sequence are quiet severe. For a well adjusted network in terms discussed in section 3.4 and 3.4.2, one finds a loss of 0.003 for the increasing variant and a loss of 0.05 for the decreasing one.
When it is considered to have more than one convolutional layer, they should have an increasing kernel size.

3.3 Recurrent NN

Recurrent neural networks (RNN) can perform very good on sequences of data that contain certain pattern.
In this problem one has a sequence of peak positions as output and the input image can also be seen as 1D signal sequence. Because this sequences are correlated intrinsically, the use of recurrent elements in the NN at some point seems to be a interesting idea.
For all analysis concerning RNNs (CuDNN -) Long short term memory (LSTM) cells are used in a bidirectional way.

3.3.1 Full RNN

In principal I just could flatten the whole image an give it as a whole sequence to a LSTM cell with the aim to classify the pixels as peak or non peak. The problem with this approach is that modern RNN elements are not able to memorise longer sequences than with ≈ 5000 elements [5]. On the other side one can estimate that a cell would need to be able to consider the correlation between $1500\text{x}\frac{700}{8} \approx 130\,000$ pixels to gain predictive power. Due to this lack a totally RNN based solution is not possible.

3.3.2 Additional RNN part

Another approach is to integrate LSTM cells at two possible parts of the NN and tests whether this recurrent element can contribute to a better performance.

CNN→ RNN → Dense: This configuration only can work when a form of desizing of the image by the CNN is applied, as discussed in the previous paragraph.
I tie to include a bidirectional LSTM with the depths$\in \{1, 2, 4, 8\}$. The rest of the network consist of two convolutional blocks and three fully connected blocks. The detailed setup is not relevant to observe the effect of the LSTM.
The outcome is that the effect of LSTM is negligible but sightly negative for the loss outcome.

CNN → Dense → RNN: The LSTM directly before the output layer is tested for the bigger NN that was also used for the other test as well as for a small NN with only one convolutional and one dense layer.
The outcome is quite different. For the big NN one finds a improvement of the loss: $0.05 \rightarrow 0.01$. That is because the LSTM causes a better alignment of the peaks of one cloumn, but it does not help by hitting the peaks precisely. That can also be seen by looking at the smaller network that starts already with a loss of ≈ 0.003. Here I do not find any improvement due to the inclusion of LSTM cells.

Conclusion: It is not found that LSTM cells could enhance the precision of this regression to the degree that is needed. I observe that rough correlations between the peaks could be recognised by the RNN part, but that is not useful for a network that can already map this.

3.4 Deep learning techniques

There are several techniques that shall help the NN to learn. Correspondingly there are related hyper parameters that can be optimised to better the training.

3.4.1 Optimiser

The basic of all learning is the update of the parameters of the NN $\vec{P}$ by stochastic gradient decent (SGD):

$$\vec{P}' = \vec{P} - \alpha \frac{\partial Loss(batch_i)}{\partial \vec{P}}$$

decay rate/(α/epochs)	momentum
0	0
0.1	0.1
0.5	0.3
1	0.6
10	0.8

Table 3.2: Grid search on the decay rate and momentum of the SGD optimiser. Each configuration is tested for various learning rates

Hereby α is the learning rate which must not be constant. It can be tuned by an optimiser to enhance the training process. What is called learning rate later on is the parameter that the optimiser takes (initial learning rate). The stochastic in SGD refers to the use of varying data batches instead of using the whole data for each update step.
In the searches the optimisers: SGD, ADAM and Adadelta are considered. The SGD optimiser is the standard optimiser of Tensorflow. ADAM considers as well the momentum of previously calculated gradients to determine the current learning rate and has shown to be able to outperform SGD in various expamle problems [6]. Adadelta uses similarly a window of past gradients. For Adadelta it is recommended to leave the hyper parameters (initial leraning rate) to their default values. For the other two optimisers the learning rates $\{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ are used for training, which is a convenient range to investigate.

As the result comes out, that surprisingly the SGD optimiser with a learing rate of 0.01 performs the best. Indeed it can outperform ADAM in bigger and smaller NN architectures. Adadelta performs slightly worse than SGD. Both are reaching easily to losses < 0.01, which is already very difficult for ADAM.

Investigating SGD: I analyse the influence of an additional momentum and a decay of the learning rate as hyper parameters parameters of SGD. The learning rate is varied as well as well, because it must be considered that this parameters cross influence the outcome. Additionally to the learning rates as given above I use the parameter pool in table 3.2 , which contain convenient values.

The outcome that the final error is mostly depending on α. For $\alpha \neq 0.01 \Rightarrow loss > 0.01$ in the search.

The additional parameters could only better the error through stronger fluctuations in the saturation area of the training. That is true for small values of the parameters (0.1 and 0.1) and could lead combined with the save best only method to a silently better outcome.

3.4.2 Normalisation and activation function

As mentioned in section 3.1.1, learning can come to a halt due to rapidly growing values inside the NN. There are some methods to prevent this from happing and so to ensure a good and stable training. In this analysis batch normalisation and self normalising methods are tested.

Batch normalisation: In short the application of batch normalisation after a layer means to subtract the activation values of each nod in the layer by the mean of a activation values of the respective nod inside one batch. Also the activatuion values are divided by the standard deviation of the distribution of the activation values of the respective nod. This leads to the result that all activation values belong to distributions with mean 0 and variance 1.
It is important to notify that batch normalisation is just sensible if the batches are big enough so that the initial distributions before normalising are similar. Otherwise one would risk to damage the causal connection between input and output.
I use always the maximal batch size that is compatible with the hardware resources for the training. That means mostly about 100 images per batch, which is assumed to be sufficient to apply batch normalisation.
Concretely, I test the application of batch normalisation after each dense layer and as well after a convolutional block for the bigger NNs.
I find that batch normalisation always worsens the final error. Maybe that is because the diversity in the images is much bigger than it can be covered by a batch of 100 images. It is further possible that the normalisation just is not needed and so brings no positive effect. Then as a consequence the batch normalisation would just add noise to the system.

Selfnormalisation: There is another approach to keep the normalisation of data when it is reached once. This technique does this by using scaled exponential linear units (see figure 3.2) instead of the common leaky rectified linear units and a special dropout that keeps the normalisation [7]. The initial normalisation is granted by a specific initialisation (lecun normalisation).

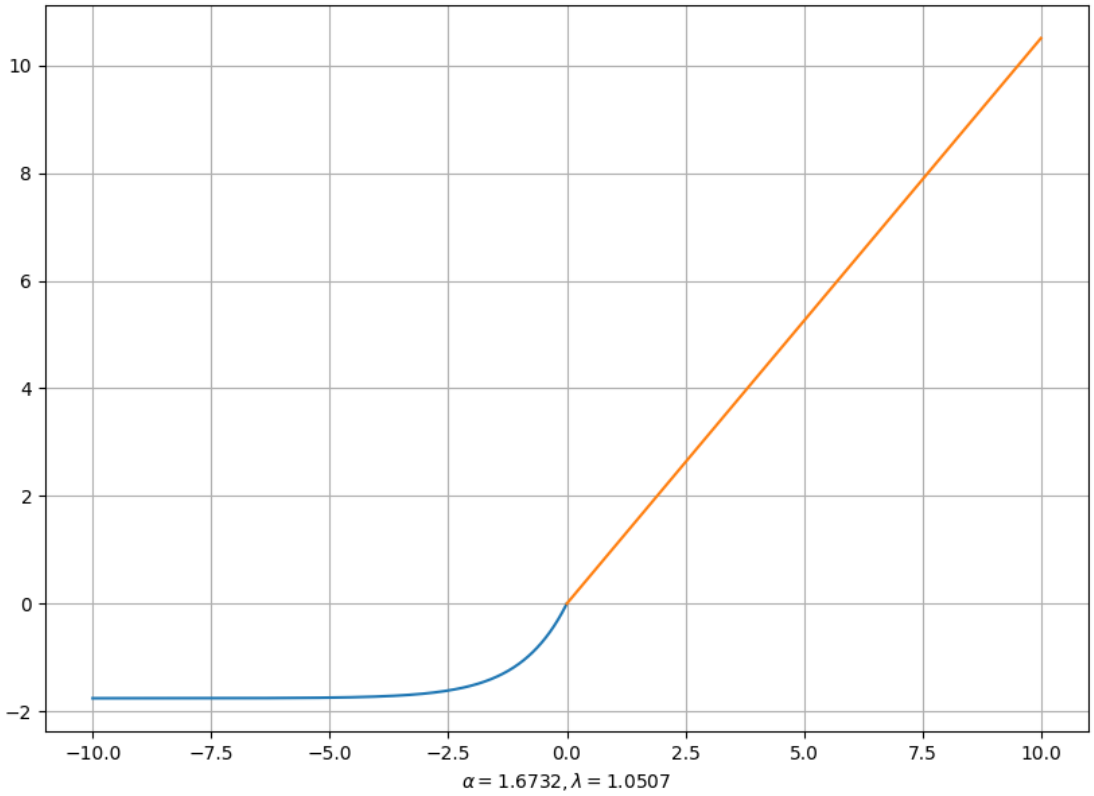


Figure 3.2: Selu function.

The result of the application of selfnormalisation is a slight decrease of the error for bigger NNs and does not change the outcome for smaller networks (<4 layers). There is not observed any bad effect at all.

Activation function: This previous result determines the activation function that is used for the hidden layers of the NN.
As already discussed in 3.1.2, the activation function of ultimate layer is the sigmoid function and not a semi linear function.
The selu function is defined as follows:

$$selu(x) = \lambda \begin{cases} x & x > 0 \\ \alpha e^x - \alpha & x \leqslant 0 \end{cases}$$

$$\lambda \approx 1.00507 \quad \alpha \approx 1.6732$$

unit	subunit
1 training	50 epochs
1 epoch	10 updates on one batch
1 batch	100 images
1 training	50 000 images

Table 3.3: Definition of the standard training.

3.4.3 Dropout

To apply dropout to a layer in the NN means to not use a certain fraction d of nods for the pass forward. This shall usually help to prevent overfittig (see sec. 3.5) and help the NN to escape global minima in the training process.
In various grid searches on different architectures d was a standard searching parameter. It was tried out $d \in \{0, 10\%, 20\%, 30\%, 40\%\}$.
I find the best results for the low regime of dropout. That is confirmed by a specific search, where just one dense layer was used after the convolution and number of nods of this layer was varied with the dropout to keep the number of active nods constant.
Especially for very small networks it turns out that no dropout is the best option. That is somehow understandable because in smaller networks it is less possible to compensate the dropout and still perform the same.
Combined this with the insight that small NN perform best and the conclusions about overfitting in section 3.5 it gets very clear that dropout is not really needed.

3.5 Training process and overfitting

3.5.1 Amount of training

One can define a standard training to comparability in the testing and searching process. The concrete values can be found in table 3.3. This standard training accesses about 200 GB of traing data and takes (inclusive validation steps) about 4.5 h.

3.5.2 Overfitting

Usually every NN training has to fight with the issue of overfittig, because the used data is repeated often for the training. This phenomena means that the NN is memorizing the training data, but do not generalise to the whole class of data.

Overfitting is not an issue for this problem, because one can generate in principle an infinite amount of data. Even practical the amount of data that can easily be stored is sufficient to bring every test NN to saturation in the training. Hereby saturation means that the loss is constant over many epochs. I could easily train 5 times longer than the standard training without repeating a single image but that would not lead to much more improvement.
To prove that overfittig does not occur I look at the loss calculated from an in independent data set that is not used for training. By doing so the expectations can be confirmed.

Consequence: Due to the fact that overfitting cannot be a problem I must not care about regularisation methods of the NN and dropout is rather optional as well.
One can see it in a way that the obstacle of generalisation is transferred to simulation task: the NN just generalises to the real data if the simulated images contain the mayor features of the real images.

4 Best performing model

In this part the final model with all its specifications is presented, that was found to perform the best. The way on which the model was found is previously explained as well. Finally I will draw conclusions from the error level that is reached at least.

4.1 Searches

I have done more specific searches in the regime of the final model to enhance the prediction.

4.1.1 Architecture

Starting point: I find that a NN with just one convolutional layer and one dense layer results in a loss of about 0.0015 $\rightarrow \sigma_{prediction} \approx 3.8\%$, using the SGD optimiser with a learning rate of 0.001.
To increase the accuracy I tire now to add additional layers.

Convolutional layers: The NN performs worse when I adds a second convolutional layer non regarding the size.

Dense layers: One additional dense layer shows a positive effect but more already worsens the result. Therefore I apply a short cut over two additional layers. This shows a very slight improvement of about 0.0002 in the loss.
I do a new grid on the width of the layers as well. The pool is the following: $\{20, 32, 50, 60, 80, 100, 120, 150, 200, 250, 300\}$. The best results were found for a constant width of 250 nods per layer.

4.1.2 Learning rate

Because the learning rate has shown to be a very sensitive parameter, I perform a tighter search for the final architecture of the NN.

Concretely, I take the learning rate $\alpha = 10^{-a}$ with $a \in \{1.1, 1.2, ...3.0\}$.
The best performance is found for $a = 1.8$ with slight improvement with respect to $a = 2$. This configuration leads than to a loss of about 0.0008.

I make the interesting observation that the loss does not increase much for a going up to 3. On the other hand the loss explodes by decreasing the learning rate to a value lower than $a = 1.7$. For $a = 1.7$ I find a loss of about 0.001 but for $a = 1.6$ the loss is of the magnitude of 0.2 after the training. One can state that this is a critical point of convergence for the NN.

4.2 Specifications of the final model

All results and advises of chapter 3 are now taken into account for the final model. Correspondingly, the NN has just a small convolutional part compared with the fully connected stage.

4.2.1 Architecture and hyper parameters

Architecture: The architecture of the model can be seen in figure 4.3. It contains one convolutional layer by size of 61x21x2 which is not decreasing the size of the image.
Than there is one dense layer with the size of 250 that takes the flattened and by the convolutional layer preprocessed image. This stage causes the biggest amount of free parameters (525.0 million).
As the next procession step there are two further dense layers that are short cutted to the output layer. That is a technique to ensure a good learning despite an increased depth of the network.
Both branches are the input for the sigmoidal output layer of 32 values.

Hyperparamters: Following the results of chapter 3, I use the SGD optimiser with a small decay rate of the learning of $\alpha/(2 * \#epochs)$. I also do not use any dropout.
The learning rate α is set to $10^{-1.9}$.

4.2.2 Outcome and conclusion

The best loss that could be achieved for the validation sample is $loss \approx 0.0008$. That corresponds to an mean error of 2.9% which means a deviation of 44 pixels in x direction

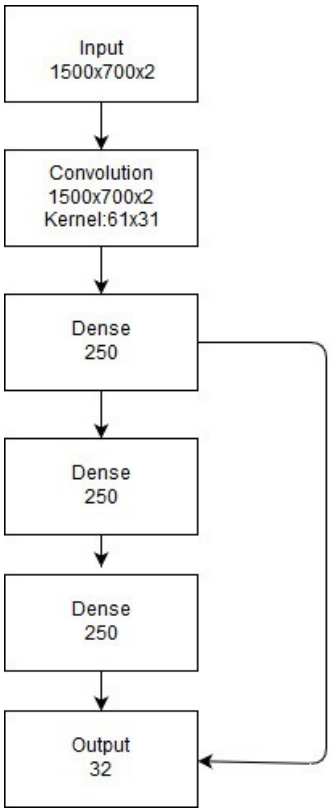


Figure 4.1: Schematic representation of the final NN.

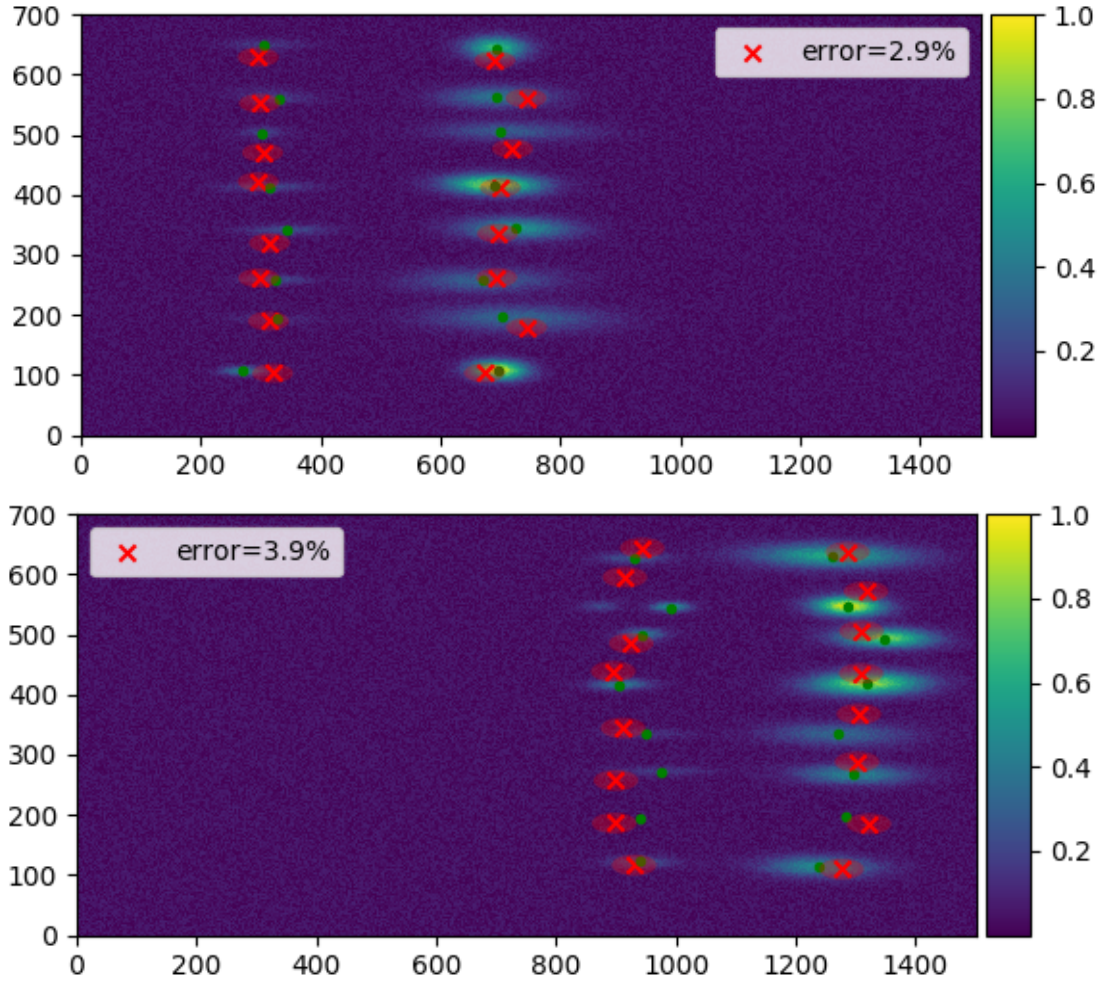


Figure 4.2: The prediction of the NN. True coordinates are marked with a green dot. The red x mark the predicted coordinates. The red ellipsis represent the mean error of the NN. The specific error for this image is given as well.

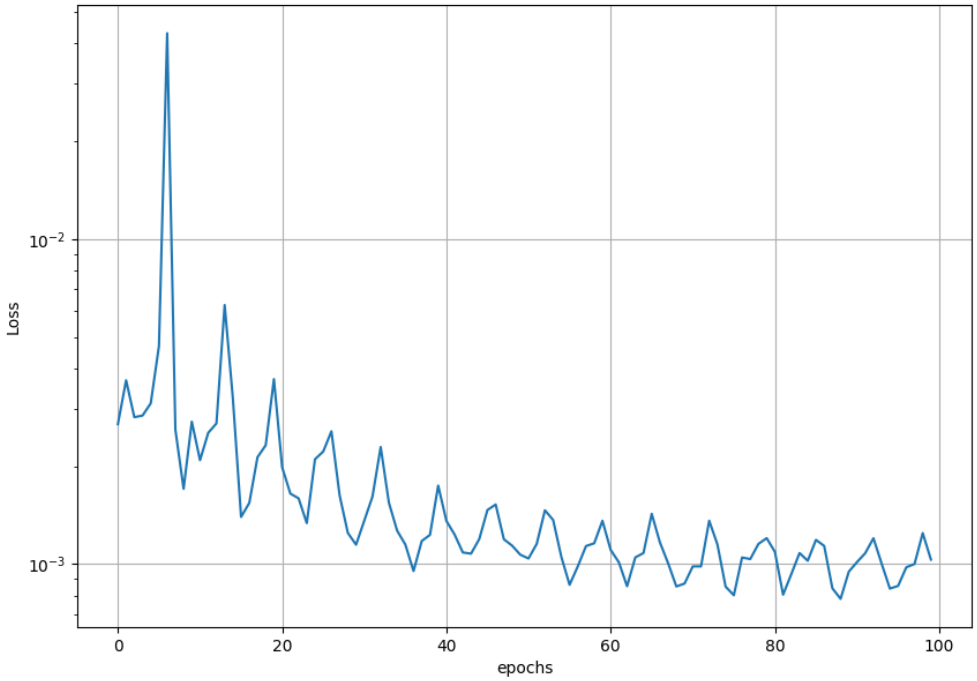


Figure 4.3: Learning curve of the final training.

and 20 pixels in y direction. The learning curve for the final network training over 100 epochs can be observed in figure
The correspondent implication for the prediction power of the NN can be observed in figure 4.2 .
One can see that the error is not constant for the images. It was found that the error has a standard deviation of 0.3%. That value was calculated from 1000 image samples, so the uncertainty of the standard deviation is by a factor of about 30 decreased and therefore negligible.
The procession time for the prediction with the trained model is 82.5 s for 1000 images. So one can the procession is very fast for one image and in the order of 0.1 s what is reasonable for the procession of images during data taking.

Conclusion: The error level is about twice as high as it is desired. That can be confirmed by looking at the prediction: there is basically no image were very peak is found by a reasonable accuracy.

5 Summary and final conclusion

5.1 Data generation

By analysing the generation procedure one can estimate a threshold for the mean prediction error from which on the prediction can be considered as successful. This threshold is 1.5%.

Problems concerning the generalisation of the NN to the real data must be solved in the image simulation process, for example by using logarithmic Gaussians instead of non skew ones.
Generally the used images should have the same degree of difficulty for the NN compared with possibly modified generated images.

5.2 Results

In table 5.1 all results and conclusions concerning the properties of the NN of chapter 3 and 4 are given.

5.3 Best model

The best achievable outcome with an average error of 2.9% is not satisfactory comparing with the estimated threshold of 1.5%.
One can see in figure 4.2 by eye that the prediction does not have a good accuracy.
Despite this it is possible for of the peaks to use the predicted coordinates nevertheless.
One can think of the fitting of 2D Gaussian with the prediction values as start parameters.
On this way the precision of the detection could be enhanced.
It must be considered as well that this fitting might not work for every peak in very image as can be observed as well in figure 4.2.

Parameter	Optimal value / Conclusion for use
Input	Normalised, image+ its gradient
Output	Normalised, sigmoid activation
#conv. layers	1
#conv. blocks	1
Depth of conv. layers	$\leqq$4
Stride/pooling	not useful
Kernel size	(30,30)-(80,80)
Aspect ratio of kernel	1:1-6:1
LSTMs /RNN elemets	not useful
Optimiser	SGD
Learning rate	≈ 0.01, very sensitive
Learning rate decay	$\approx 1.6 \cdot 10^{-4}$, small
Dropout	0-10%
Normalisation	Selformalisation, no batch norm
Amount of training	<50.000 images
Dense width	≈ 250, <300
Dense depth	$\leqq$2
Shortcuts	useful

Table 5.1: Summary of all investigated features of NNs and the advice how use them for this problem.

List of Tables

List of Figures

Bibliography

[1] Wolfgang A. Caliebe, Aleksandr Kalinko Marcel G. Vadim Murzin M. Vadim Murzin: High-flux XAFS-beamline P64 at PETRA III. In: *AIP Conference Proceedings 2054, 060031 (2019)* (2019). http://dx.doi.org/[10.1063/1.5084662]. – DOI [10.1063/1.5084662]

[2] *High resolution von Hamos-type x-ray emission spectrometer.* : *High resolution von Hamos-type x-ray emission spectrometer*, 2019. http://photon-science.desy.de/facilities/petra_iii/beamlines/p64_advanced_xafs/infrastructure_and_experiment/spectrometer/index_eng.html

[3] Key point detection in flower images using deep learning. (2018). https://hackernoon.com/key-point-detection-in-flower-images-using-deep-learning-66a06aac

[4] Detecting facial features using deep learning. https://towardsdatascience.com/detecting-facial-features-using-deep-learning-2e23c8660a7a

[5] Shuai Li, Chris Cook Ce Zhu Yanbo G. Wanqing Li L. Wanqing Li: Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN. (2018). – arXiv:1803.04831

[6] Diederik P. Kingma, Jimmy Lei B.: ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. In: *ICLR* (2015). – arXiv:1412.6980v9

[7] Günter Klambauer, Andreas Mayr Sepp H. Thomas Unterthiner U. Thomas Unterthiner: Self-Normalizing Neural Networks. (2017). – arXiv:1706.02515