Primjena konvolucijske neuronske mreže u brojanju i lokalizaciji proteina fotografiranih super-rezolucijskom mikroskopijom

Meić, Ivan

University of Zagreb

Faculty of Science

Department of Biology

Ivan Meić

Application of convolutional neural network for protein counting and localization photographed by super-resolution microscopy

Primjena konvolucijske neuralne mreže u brojanju i lokalizaciji proteina fotografiranih super-rezolucijskom mikroskopijom

Graduation Thesis

Zagreb, 2020.

This graduation thesis was conducted at University of Vic – Central University of Catalonia, Vic, Spain, under supervision of dr.sc. Carlo Manzo. This thesis was submitted for evaluation to the Department of Biology, Faculty of Science at University of Zagreb with aim of obtaining title Master of Molecular Biology.

BASIC DOCUMENTATION CARD

University of Zagreb
Faculty of Science
Department of Biology

Graduation thesis

Application of convolutional neural network for protein counting and localization photographed by super-resolution microscopy

Ivan Meić

Rooseveltov trg 6, 10000 Zagreb, Croatia

In recent years novel microscopy techniques have led to the development of super-resolution imaging with increased resolution compared to conventional microscope techniques. A branch of super-resolution microscopy, based on single-molecule localization that includes techniques such as Photoactivated localization microscopy (PALM) and Stochastic Optical Reconstruction Microscopy (STORM), enables researchers to study cellular processes with increased resolution. Single-molecule localization microscopy relies on collecting a set of images where only a subset of optically resolvable fluorophores is emitting light. The analysis of these images allows for reconstruction of high-quality super-resolution images. This graduation thesis proposes a further step in super-resolution image analysis, based on the analysis of individual protein localization. It proposes to use the multiple counts produced by a single protein to predict its position. Position is predicted beyond overcounting artifact through a neural network, in order to predict the protein number and position. Counting proteins from super-resolution image is a problem because of fluorophore overcounting. Unlike previous papers trying to solve this problem, this graduation thesis does not use any modelling and relies only on neural network's ability to learn from data. This approach provides researchers with a fast performing algorithm that requires minimal knowledge about neural networks and image analysis. The main object of this graduation thesis is to create and measure how well the state of the art image processing neural network can localize and count proteins.

(24 pages, 15 figures, 2 tables, 18 references, original in English)

Thesis deposited in Central Biological Library
Keywords: super-resolution microscopy, CNN, machine learning, protein counting and localization
Supervisors: Assoc. Prof. Carlo Manzo and Assoc. Prof. Damjan Franjević
Reviewers: Assoc. Prof. Damjan Franjević, Assist. Prof. Rosa Karlić, Assist. Prof. Sunčica Bosak, Assoc. Prof. Inga Marijanović
Thesis accepted: 19. 2. 2020.

TEMELJNA DOKUMENTACIJSKA KARTICA

Sveučilište u Zagrebu
Prirodoslovno-matematički fakultet
Biološki odsjek Diplomski rad

Primjena konvolucijske neuralne mreže u brojanju i lokalizaciji proteina fotografiranih super-rezolucijskom mikroskopijom

Ivan Meić

Rooseveltov trg 6, 10000 Zagreb, Hrvatska

Razvoj novih tehnika mikroskopije tijekom proteklih nekoliko godina doveo je do razvoja super-rezolucijske mikroskopije s povećanom rezolucijom u odnosu na konvencionalne tehnike mikroskopije. Grana super-rezolucijske mikroskopije, bazirana je na lokalizaciji jedne molekule i uključuje tehnike "Photoactivated localization microscopy" (PALM) i "Stochastic Optical Reconstruction Microscopy" (STORM), koje omogućuju znanstvenicima proučavanje staničnih procesa s povećanom rezolucijom. Mikroskopija lokalizacije jedne molekule oslanja se na prikupljanje seta slika gdje samo podskup optički razlučivih fluorofora emitira svjetlost. Analiza tih slika omogućuje rekonstrukciju visoko kvalitetnih super-rezolucijskih slika. Ovaj diplomski rad predlaže sljedeći korak u analizi super-rezolucijskih slika, baziran na analizi lokalizacije pojedinog proteina. Rad upotrebljava višestruko brojanje signala jednog proteina za predviđanje pozicije proteina. Predviđena lokalizacija nadilazi artefakte višestrukog brojanja kroz upotrebu neuralne mreže, s ciljem brojanja i lokalizacije protein. Brojanje proteina sa slika super-rezolucijske mikroskopije predstavlja problem zbog višestrukog brojanja fluorofora. Za razliku od ostalih radova koji pokušavaju riješiti ovaj problem, ovaj diplomski rad ne koristi modeliranje nego se oslanja na sposobnost neuralne mreže da uči iz podataka. Ovaj pristup pruža znanstvenicima brzo izvršavajući algoritam koji zahtjeva minimalno znanje o neuralnim mrežama i analizi slika. Glavni cilj ovog diplomskog rada je izraditi i izmjeriti koliko dobro suvremena neuralna mreža za obradu slika može lokalizirati i prebrojati proteine.

(24 stranice tekste, 15 slike, 2 tablica, 18 literaturnih izvora, jezik izvornika: engleski)

Rad je pohranjen u Središnjoj biološkoj knjižnici
Ključne riječi: super-rezolucijska mikroskopija, CNN, strojno učenje, brojanje i lokalizacija proteina
Voditelji: izv. prof. dr. sc. Carlo Manzo i izv. prof. dr. sc. Damjan Franjević
Ocjenitelji: izv. prof. dr. sc. Damjan Franjević, doc. dr. sc. Rosa Karlić, doc. dr. sc. Sunčica Bosak, izv. prof. dr. sc. Inga Marijanović
Rad prihvaćen: 19. 2. 2020.

List of abbreviations:

NN – neural network

CNN – convolutional neural network

RMSE- root mean square error

ML – machine learning

Contents

1. Introduction

1.1. Microscopy and machine learning

Microscopy has been a major tool in cell biology for centuries. It has been used for a wide range of application from visualizing cellular processes to use in diagnostics. In the past couple of years a set of unique challenges has arisen in microscopy. These problems arise from the huge amount of data generated by microscopy aided by high-throughput imaging techniques and need to analyze that data effectively. These kinds of problems can be efficiently solved with machine learning (ML). Image pattern recognition has been a challenge for ML until early 2010s when deep learning based on neural networks (NN) became promising tool for image pattern recognition. Examples of problems that can be solved with this novel technique include image labeling for big data sets, image segmentation which includes finding cells and sub-cellular structures on images and image super-resolution microscopy. Synergy between machine learning research especially CNN and microscopy is novel, with most of papers on this topic appearing in the last couple of years (Von Chamier et al. 2019).

1.2. Super-resolution microscopy

There is a fundamental limit to resolution for all conventional microscopy methods called Abbe's diffraction limit. It states that spatial resolution is approximately half of optical wavelength. All cellular processes and structures that occur on a smaller scale than the ones defined by visible light and Abbe's law should be studied and visualized with other methods. To work around this problem novel methods have been developed, including stimulated emission depletion, structured illumination microscopy and localization microscopy (Nehme 2018). Localization microscopy includes techniques such as Photoactivated localization microscopy (PALM) (Betzig et al. 2006) and Stochastic Optical Reconstruction Microscopy (STORM) (Betzig et al. 2006). Localization microscopy relies on collecting images containing light emissions from a set of nanometer size fluorophores. The first step in performing an experiment is to attach nanoscale fluorophores to sample. These fluorophores can transition to active state after light excitation. In active state fluorophores can either go to dark state while emitting light and afterwards be reactivated again multiple times or it can get photo-bleached meaning that fluorophore will not emit light again. Fluorophore's ability to transition from activated to dark state multiple times results in a process called fluorophores blinking. Second step takes a series of microscopy images after exposure to light, sample can be exposed to light multiple times. After every light exposure fluorophores will start emitting light stochastically, meaning not all fluorophores emit at the same time but only a subset of them. Fluorophores in images cannot have overlapping emission signals because those signals cannot be resolved. To avoid overlapping signals fluorophores need to be spatially separated and if possible separated in time, this separation is achieved through blinking (Rollins et al. 2014). Afterwards each frame is analyzed to determine fluorophores localizations and collection of analyzed frames is then used

to construct a single image with higher resolution than one defined by Abbe's law (Boyd et al. 2018).

The problem of constructing a single image has been solved with a variety of algorithms. Algorithms based on sequential fitting of emitters followed by subtraction of the model PSF, blinking statistic, sparsity, multi-emitter maximum likelihood estimation, single-image super-resolution by directory learning and CNN named Deep-STORM (Nehme 2018). Aim of all these algorithms is to reduce fluorophore signal area on microscopy images. The field of super resolution image analysis has so far been mostly concentrated on constructing a super-resolution image with previously mentioned algorithms. Besides constructing a super-resolution image it is also important to find fluorophore localizations and to count fluorophores from set of frames or from super-resolution image. The task of counting and localization is an important problem in the field of quantitative bioimaging, which is trying to quantify and measure biological properties from bioimages. Due to fluorophore blinking and signal overlaps it is not possible to simply localize and count fluorophores and equate signals to labeled cell structure or protein. As is described above, fluorophores do not emit just one signal instead they can appear multiple times. This makes for good separation in space and time, but if every signal is counted as one fluorophore it will lead to overcounting. The number of times one fluorophore blinks depends on the experimental condition and used fluorophores. The number of blinks per fluorophore can be modeled with geometric distribution (Lee et al. 2012). Depending on the experimental condition and used fluorophore overcounting problem can be severe. This problem is so far mostly resolved with modeling fluorophore blinking (Lee et al. 2012, Rollins et al. 2014, Nieuwenhuizen et al. 2015, Fricke et al. 2015, Nicovich et al. 2017). To resolve the localization and counting problem this graduation thesis proposes to use a neural network able to take in super-resolution images and predict protein localizations. The next section will introduce basic concepts of neural networks needed to understand this thesis.

1.3. Machine learning and deep learning

ML describes a group of algorithms able to learn from examples unlike other algorithms which are programmed by hand to do specific tasks (Goodfellow et al. 2016). ML algorithm starts with a mathematical model whose parameters are free to vary. To estimate the correct values of the parameters data is presented to a ML model, usually pairs of input and output data. ML parameters are then changed to best describe input output dependence. For example, if the task is to predict if the cat is present in the image ML approach is to present a ML model with labeled images with and without cat and let the ML algorithm set parameters. To solve this problem with other algorithms the programmer would need to code rules in the algorithm to make that prediction. ML encapsulates a vast range of algorithms, from simple ones like linear regression to more complex ones like deep neural networks. For ML to be applicable to a problem two conditions need to be met: first it is not clear how to write a program to perform a task and second there is available data for a program to learn from. With the rising of available data and improvements in processing speed machine learning research and application has been on a rise for past few decades. These two phenomena have especially helped deep learning to reach

practical application. Deep learning is a subset of neural networks with multiple hidden layers. This graduation thesis uses deep convolutional neural network and the next sections will describe the main ideas and problems relevant to the application of this method.

1.4. Generalization

Generalization is an ability of ML algorithms to predict correct answers for data points it has never seen. ML models have two pitfalls. First is a simple model that is unable to find meaningful aspect of data for correct prediction. In this case the ML model will underperform because important aspects of data are ignored. To fix this problem a more complex ML model should be chosen. Second pitfall is an opposite, the model can be too complex and it falls short on unseen data while excelling on data it was trained on. In this case ML model treats noise in data as an important aspect in forming a prediction. This is a well know problem named overfitting. Generalization guides programmer while developing ML model to avoid pitfalls and find a ML model with enough complexity to handle the task at hand, but at the same time avoid overfitting. This makes generalization an important ML model performance metrics (Bishop. 2006). To measure the ML model generalization ability available data is usually split into two categories: train data that is used while the ML model is being trained and validation data that is used to evaluate trained ML model performance.

1.5. Neural networks

NN represents a type of machine learning algorithm. There is a variety of different types of NN specific for different application based on data structure, but the bases for all of them are the same. In this section basic NN concepts will be explained after which convolutional neural networks (CNN) specifics will be explained.

1.5.1. Basics of Neural Networks

NNs are ML models loosely based on a way human brain works. The entire field was started from a desire to understand and model human brain. At the beginning most ideas in NN research were based on neurophysiology ever since the field is moving away from neurophysiology and is adopting ideas from mathematics and statistics. Currently the field has completely moved from neurophysiology to data modeling tool (Rojas 1996).

NN building blocks and fundamental units are neurons, which are nothing like human neurons and actually represent mathematical operations shown in Figure 1. The first neuron operation is the weighted sum of neuron input points. In some cases, it is beneficial to add bias to resulting weighted sum. After summation the result is passed through an activation function (Bishop 2006).

$$f(b + \sum_{i=1}^{n} x_i * w_i)$$

Figure 1. Operation performed by one neuron. *f* is an activation function, B represents bias, x_i are input values and w_i are corresponding weights.

Activation functions are different functions whose main role is to introduce nonlinearity to NNs. Nonlinearity is one of the most important properties of NNs that enables them to become universal approximators (Hornik 1991). Without activation functions the NN is reduced to linear regression. The second important reason to use an activation function is to limit neuron output values in a range. Having a range of values enables neural outputs to be treated as probability if the range is between 0 and 1. Having a range of values for a neuron's output also reduces problems during NN training with exploding and vanishings gradients (Goodfellow et al. 2016). Examples of activation functions used most commonly are ReLu and sigmoid function shown in Figure 2.

$$ReLu(x) = \max(0, x)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Figure 2. Shows ReLu and sigmoid functions ReLu activation function returns the input value if it is bigger than zero, otherwise it is zero or less function returns zero. Sigmoid function symbolized with $\boldsymbol{\sigma}$, has important property of returning values from 0 to 1 for any input.

Neurons are grouped into layers. NNs have 3 basic types: input layer, hidden layer and output layer (Bishop 2006), the NN scheme is shown in Figure 3. Input layer is the first layer responsible for taking on input values. Neurons in an input layer are unlike other neurons because they represent input to a network and do not perform any mathematical operation. E.g. if NN is built to predict the price of a house based on: number of rooms, number of bathrooms and house area. Number of neurons in input layer is three. Hidden layers are the heart of a NN they perform an operation described above. Number of neurons in a hidden layer and number of hidden layers in a NN can vary significantly depending on a problem at hand. Output layer gives a final NN prediction. It is constructed depending on how many predictions NN needs to output. Activation functions in output layer usually differ from one used in the hidden layer, they dependent on NN prediction type. E.g. from the previous house example number of neurons in output layer is one, it is house price. The activation function would be linear because it needs to output a price not a probability of an event.

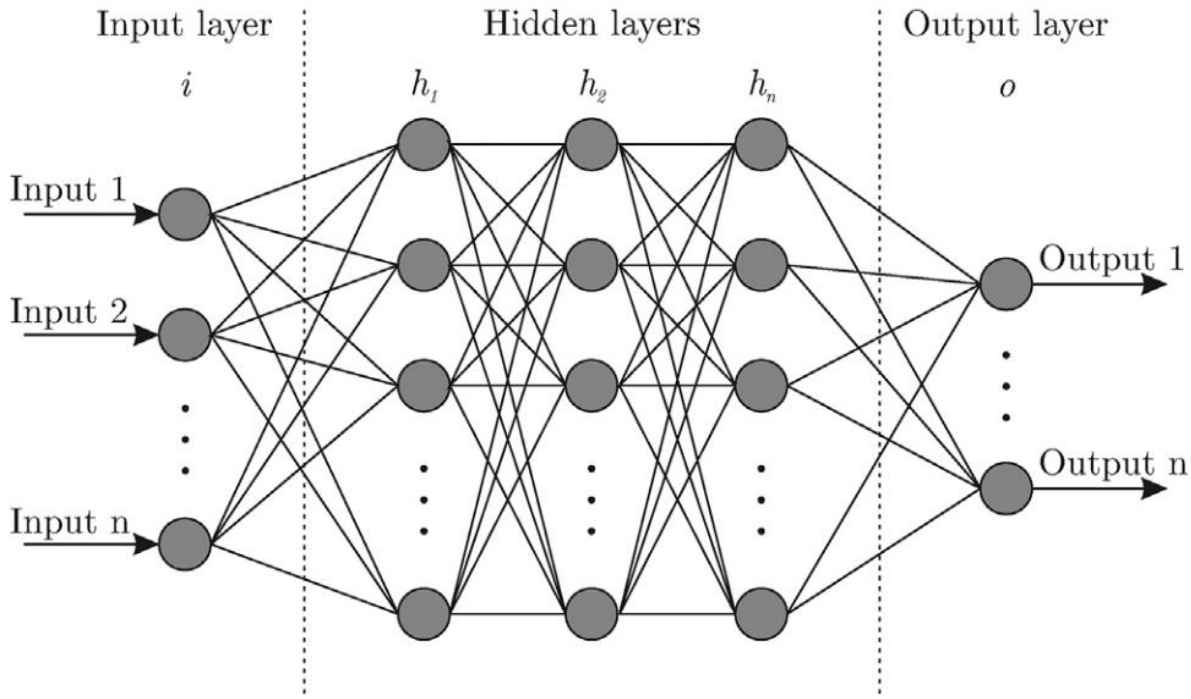

Figure 3. Example of NN's graphical representation taken form Bre et al. (2018).

To find an optimal weight for each neuron is a complicated task, especially for NN with multiple hidden layers. Backpropagation algorithm is used to solve this problem. The first requirement is to define a loss function. Loss function is a NN performance estimator, it uses a distance measure between correct answers and NN prediction (Bishop 2006). E.g. for our house price example loss function could be a root mean square error. Backpropagation goal is to minimize loss function, which makes NN prediction more correct. To minimize loss function derivative of loss function is calculated. Partial derivation for every parameter in a layer is calculated using chain rule of derivation. Starting from a last hidden layer, layer-by-layer partial derivations are calculated going to the first hidden layer. Backpropagation got its name due to this process of going backwards through a network. After derivation for every parameter in a NN is calculated it is time to change those parameters. There is a range of different optimization parameters with combined information from current variable value and its derivation to calculate a new value for a parameter. Besides derivation and current value every optimization algorithm takes in a learning rate as an input. Learning rate regulates parameter change in every step. A bigger learning rate value will change NN parameter faster, a smaller learning rate on the other hand will change NN parameter values slowly. Combining these three inputs optimization algorithm calculates new parameter values. Currently there are multiple optimization algorithms to choose from, but at its core they are all based on a simple rule current value is subtracted from partial derivation for parameter multiplied with learning rate (Bishop 2006), the equation is shown in Figure 4.

$$\theta_p := \theta_p - \alpha \frac{\partial}{\partial \theta_p} J(\theta_p)$$

Figure 4. The basic equation for optimization algorithm. $\boldsymbol{\theta_p}$ is NN parameter, $\boldsymbol{\alpha}$ is learning rate, $\frac{\partial}{\partial \boldsymbol{\theta_p}} \boldsymbol{J}(\boldsymbol{\theta_p})$ represent loss function partial derivative with respect to $\boldsymbol{\theta_p}$.

Backpropagation and optimization can be executed on all examples at one or on a subset of examples, these methods are called batch gradient descent and stochastic gradient descent, respectively. One cycle of performing optimization on all examples is called an epoch. Number of epochs determines how many times process of weights adjustment will be done across all training examples (Wilson and Martinez 2003).

After the training process is completed NN performance is tested on examples NN has not seen before. This step tests NN generalization ability. Overall quality of NN is based on generalization ability and is used to guide next step of the NN design process.

1.5.2. Convolutional neural network (CNN)

To analyze images with classical NN every grayscale pixel is represented by one number and for a typical colored image every pixel is represented by three numbers. If the size of an image is 620x480 pixels^2 it means it has 297 600 pixels. If that image is grayscale its NN would have 297 600 neurons in the input layer and if it is a typical colored image its NN would have 892 800 neurons in the input layer. Classical NN would require an enormous number of parameters just for first hidden layer. Such an enormous amount of parameters just for first layer would make classical NN hard to train and its performance would be slow after training. Probably the biggest problem would be NN susceptibility to overfitting.

Another specific requirement for images is translational pattern recognition. Pattern position in an image is not important, the only important thing is the pattern's presence or absence. For example, if NN task is to locate abnormal cell among healthy ones, abnormal cell position and special rotation is not an important factor. Classical NN are unequipped for this problem because they treat every pixel separately from one a nether. One pixel should be treated in the context of surrounding pixels.

To solve these problems CNN uses convolution operation between a filter and a part of an image. Filter is a n x n matrix, convolution operation is a dot product between the filter and part of an image plus bias. Convolution is applied to one part of the image afterwards filter is moved to a next position and applied again process is repeated across all of an image (Goodfellow et al. 2016). This process replaces a classical neuron that was explained above. Because a filter is usually significantly smaller than the image number of parameters is drastically reduced. Most commonly used filter sizes are 3x3, 5x5, 7x7 and 9x9. To visualize a scale of parameter reduction let's look at previously mentioned 620x480 pixels^2 image that would have 297 600 neurons in the input layer for classical NN, if that network had ten filters of size 9 for inpute layer it would only require 640 parameters instead of 297 600. The second benefit of filter usage is filter application across different image sections. This enables filter to specialize in

recognition of a single pattern. His application across all image positions means it can recognize a pattern at any image position. Third benefit is in filter construction itself, because filter is a matrix it takes image subset equal to filter size not a single pixel. This means filter interprets part of an image in context, not a single pixel out of context.

Stride is used to specify filters shift. For example, if the stride is (2,2) it means the filter will be moved 2 pixels in horizontal direction and when the filter is moved in vertical direction it will be moved by 2 pixels as well. Applying a filter this way will result in lower usage of pixels at image edges. To solve this problem image size can be extended by adding more values at the image edge by a process called padding. Added values are usually just zeros. Padding is usually used to preserve the image size from one layer to the next and to use pixels at an image edge same number of times as rest of pixels.

Part of most of modern CNNs is pooling layer. Pooling layer separates image form previous layer into equal parts based on pool size and returns one number from image subsets. Most commonly used pooling is maxpooling. It takes image subset and outputs the maximum value from that subset, another example is average pooling which return image subset average. Pooling main role is to reduce the number of parameters in a network, making the training process faster and network more computationally manageable (Goodfellow et al. 2016).

The opposite process of pooling is upsampling. It repeats input across a defined number of columns and rows. Upsampling is a computationally cheap way to increase image from one layer to the next. Its role is to achieve desirable image size.

1.6. Objective of the experiment

1. Create and optimize CNN architecture with the capability to predict the probability density for protein position and estimate the number of proteins from single molecule localization microscopy.
2. Develop an algorithm for protein localization from the CNN localization probability output.
3. Benchmark the model by measuring the CNN performance with Jaccard index and RMSE.

2. Materials and methods

This graduation thesis describes CNN that takes as input super resolution image and outputs image containing protein localization probability. Second part of this graduation thesis describes an algorithm used for protein localization bases on CNN predicted probabilities. After proteins have been localized Jaccard index and RMSE are calculated. Algorithms were written in Python 3.7, Keras library (Chollet 2015) back by TensorFlow was used to write CNN.

2.1. Super-resolution image generation

The first step in super-resolution image generation is to simulate protein coordinate positions. To do it x and y protein coordinates are sampled from uniform distribution, if the distance between two coordinate point is smaller than critical distance one coordinate will be moved by value sampled from a normal distribution with mean zero and standard deviation equal to critical distance. Critical distance is equal to the smallest assumed distance between two proteins in super-resolution image. When scaled, these coordinates represent output data for CNN. The next step in super-resolution image generation is to assign fluorophores to protein positions. Number of fluorophores assigned to a protein is sampled from a geometric distribution whose parameter is set to 0.2. Fluorophores are positioned around proteins by adding protein coordinate to values sampled from a normal distribution with mean zero and standard deviation equal to super-resolution microscopy localization precision. Fluorophores localizations are then rendered with Gaussian blur to create final super-resolution image. The code is based on DEEP-STORM paper (Nehme 2018).

Images were simulated with Code 1. (see Appendix). Code 1 was written by Carlo Manzo. Generated input images have 45 proteins per image, pixel size of 25 nm, super resolution localization precision of 1 nm and protein dimensions of 0.2 nm. Rendering factor 5 meaning after rendering pixel size is 5. The output image is a grid with zeros except for protein position coordinates where it is one. The output image is twice the size of input image, meaning that the zoom factor is 2.

2.2. Image normalization and standardization

To secure better convergence properties input image set is normalized and standardized. First all values are converted to float values. Every image is then individually scaled to zero to 1 range using the formula in Figure 5.

$$new\_image = \frac{image - image.minimum}{image.maximum - image.minimum}$$

Figure 5. Image normalization formula.

After normalization image is standardized to mean zero with standard deviation equal to 1 using the formula in Figure 6.

$$new\_image = \frac{image - image.mean}{image.standard\_deviation}$$

Figure 6. Image standardization formula.

Code for image preprocessing can be found in the Appendix, Code 2.

2.3. CNN architecture

Convolution process used in a network is composed of 3 layers. First layer does standard convolution without applying any activation function and without a bias. Stride is one in horizontal and vertical direction and filter size is 5 x 5. Convolution results are normalized by batch in a second layer. Third layer applies a ReLu activation function to normalize data. This layer will be named conv_bn_relu.

The network has encoder-decoder architecture. Encoder stage consists of repeating two conv_bn_relu followed by maxpooling layer. Every pooling layer halves image output size. This layer schedule doubles number of filters in every block starting from 16 to 128. Decoding stage starts after and uses the same block structure except instead of maxpoolng it uses upsampling. Number of filters is halved in every block from 128 to 8 and output image size is doubled. Last layer in a network is a conventional layer with applied sigmoid function.

2.4. Training

The network was trained with 2000 pairs of network input/output images with train validation split 70 : 30.

Only two classes of output prediction exist. Either the coordinate is a protein location labeled with one or it is not in which case it is labeled with zero. Binary loss function is one best suited for the given problem and is used to train a network. The network was trained with Adam optimizer over 100 epochs and batch size of 32. Learning rate was set to 0,001 with a new learning rate set if the validation data set loss has not been reduced for 5 epochs. The newly selected learning rate is half of previous value. Training process has been run on The Institute of Photonic Sciences (ICFO) server. Code for CNN architecture and training can be found in the Appendix, Code 3.

2.5. Localization algorithm for CNN localization probabilities and validation

The first stage of the network localization algorithm is to convert the probability distribution given by the network to location coordinates. Network probabilities predictions are converted to a range from 0 to 1. This conversion makes it easy to choose a threshold value above which network predictions are considered valid. Pixels are then grouped to a same group if they are above threshold value and if they are one pixel away from a group in all directions.

Regions' centroids are calculated afterwards, they represent protein localization. Localization is than followed by validation.

Performance metrics are Jaccard index and root mean square error (RMSE).

Jaccard index

Jaccard index is defined as the ratio between number of elements in overlapping sets and number of elements in union set. The element is in overlapping set if the Euclidean distance between centroids is smaller than 20 nm. Union set is calculated as the sum of elements in both sets minus overlapping set.

Root mean square error (RMSE)

Root mean square error, shown in Figure 7., is the square root of the quotient of summed squared Euclidean distances and number of samples.

$$\boldsymbol{RMSE} = \sqrt{\frac{\sum_i^n (\boldsymbol{y_{pred}} - \boldsymbol{y_{corr}})^2}{\boldsymbol{n}}}$$

Figure 7. RMSE formula y_{pred} is algorithm prediction, y_{corr} is correct value and n is number of elements used in calculation

The first step of validation is to calculate Euclidean distances are between all predicted protein localizations and correct protein localizations. Points corresponding to smallest Euclidean distance are then classified to overlapping set if their Euclidean distance is smaller than 20 nm. Overlapping threshold value of 20 nm is chosen as it is the typical localization precision of STORM method. If point is in an overlapping set its Euclidean distance will be used to calculate RMSE. Centroids corresponding to that distance will be removed from a list of Euclidean distances and new smallest value will be chosen and process repeated until all Euclidean distances are processed. Last step is to calculate the Jaccard index and RMSE after the algorithm has gone through all images. Descriptive localization and validation algorithm can be found in the Appendix, Code 4.

3. Results

Figures 8. and 9. show side by side comparisons between a) simulated super-resolution microscopy image before rendering with specified correct protein localizations and b) super-resolution image after rendering, c) CNN network prediction and d) image with true and predicted protein position. Examples given in Figures 8. and 9. clearly show CNN ability to reduce the space of possible fluorophores localizations in a non trivial way for super-resolution images. To give a clearer example of CNN localization ability Figure 10. shows post-CNN prediction analysis steps required to localize fluorophores. Visualized steps are described in a CNN validation algorithm, Figure 10. includes steps until Euclidean distance calculation.

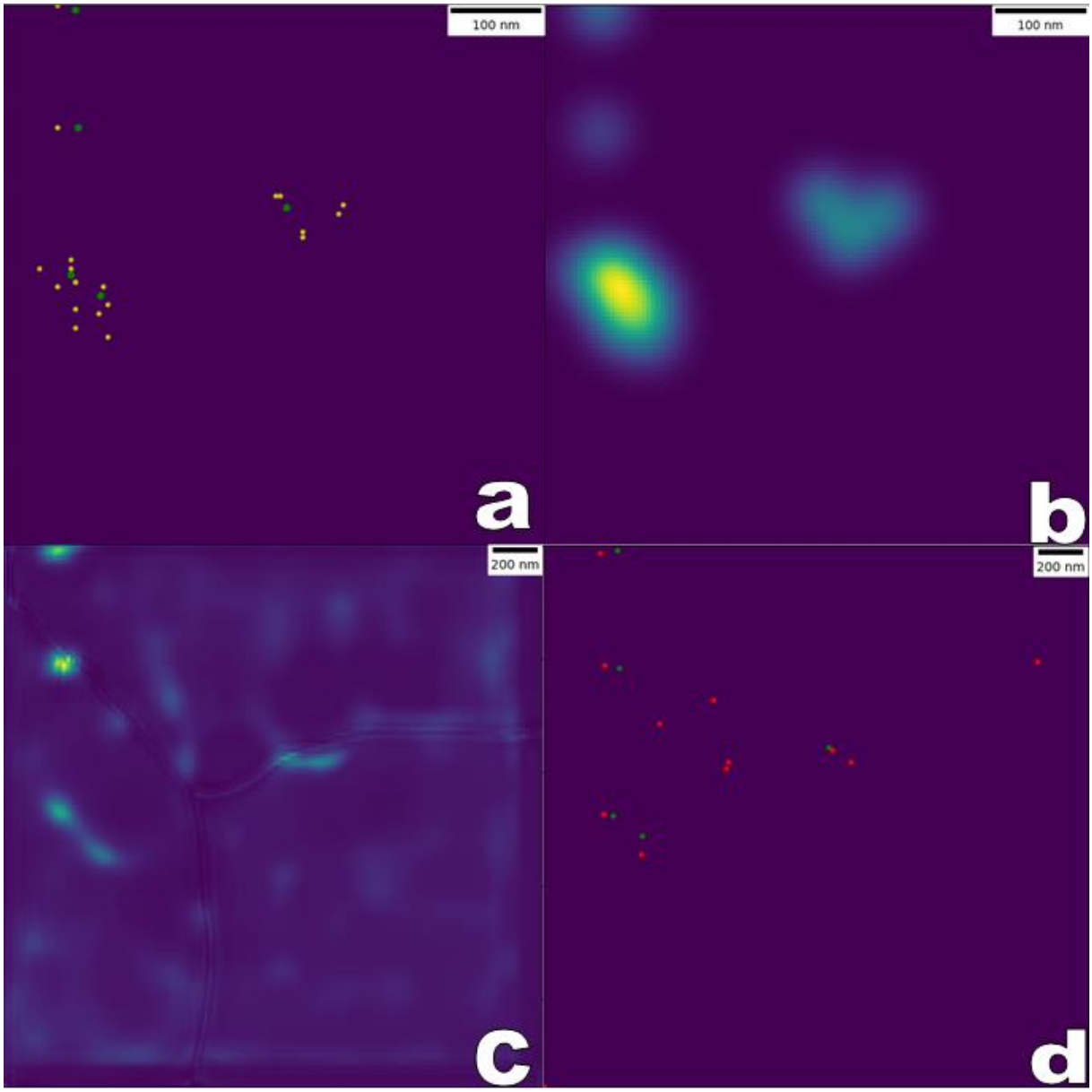

Figure 8. Image creation steps (a, b) and protein localization prediction steps (c, d) for images with 5 proteins per image. a) shows first two steps in image generation with Code 1, green dots represent protein location and yellow represent fluorophores location, b) final product of Code 1. is rendered microscopy image for STORM method, c) localization probability predicted with CNN used in this graduation thesis, d) final output of localization algorithm, red dots represent protein localizations predicted at the threshold value of 0.2 , green dots represent true protein localizations.

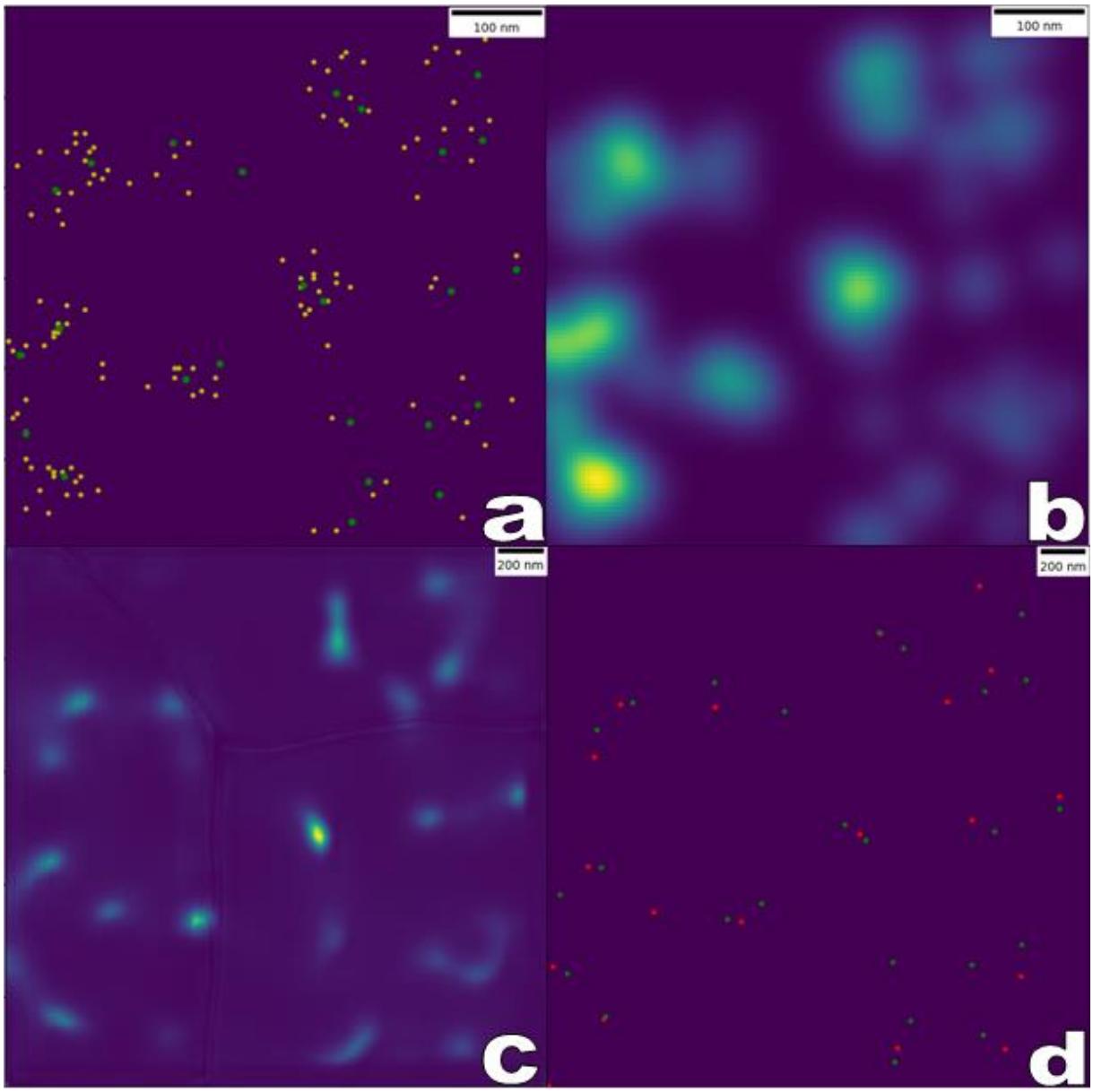

Figure 9. Image creation steps (a, b) and protein localization prediction steps (c, d) for images with 25 proteins per image. a)shows first two steps in image generation with Code 1, green dots represent protein location and yellow represent fluorophores location, b) final product of Code 1 is rendered microscopy image for STORM method c) localization probability predicted with CNN used in this master thesis, d) final output of localization algorithm, red dots represent protein localizations predicted at threshold value of 0.2 , green dots represent true protein localizations.

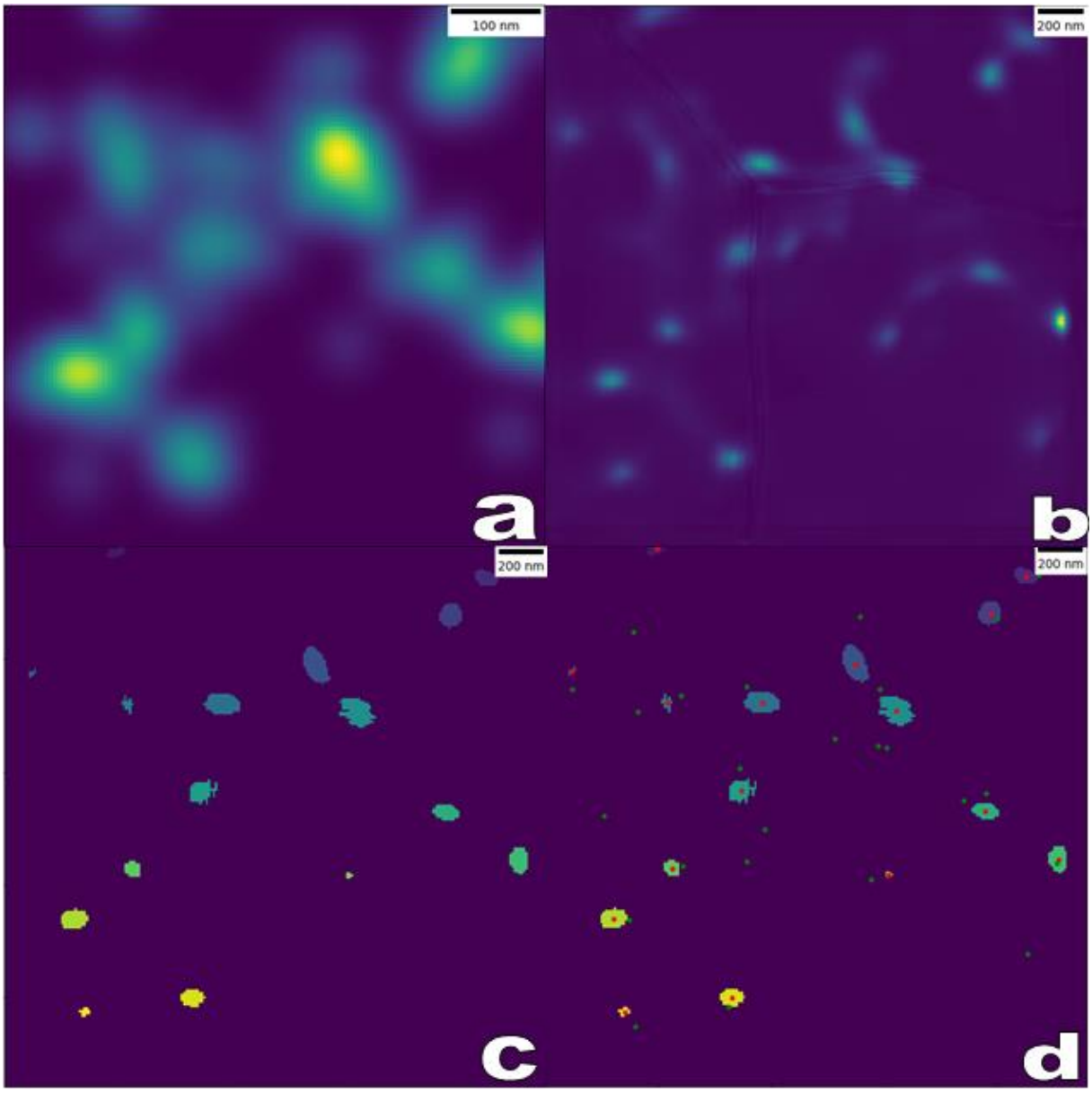

Figure 10. Validation algorithm process. a) network input image, b) CNN prediction scaled to rage from 0 to 1, c) predicted protein localizations areas with threshold value grater then 0.2, d) green dots are correct protein location and red ones are localizations predicted from areas.

As described before, the Jaccard index is a metric used to validate the localization quality. Figure 11. shows the CNN performance across images with different number of proteins. The number of proteins per images start at 5 and is increasing by 5 until 50 proteins per image. Because it is necessary to choose a threshold value for localization prediction Figure 11. shows the CNN performance at different threshold values. Threshold values start from 0.05 and is increasing by 0.05 to 0.95.

To include most protein in overlapping Jaccard index threshold value with biggest Jaccard index should be chosen. To clearly show the algorithm‘s localization ability Figure 12. is extracted from Figure 11. and shows the Jaccard index value for images with different number of proteins per images with threshold value set to 0.15 as it is a value with which most images have their maximum. It has to be noted that images with 5, 10 and 15 proteins per image have peaked at

different values. To better show where the maximums are Table 1. contains optimal threshold values across a range of proteins per image.

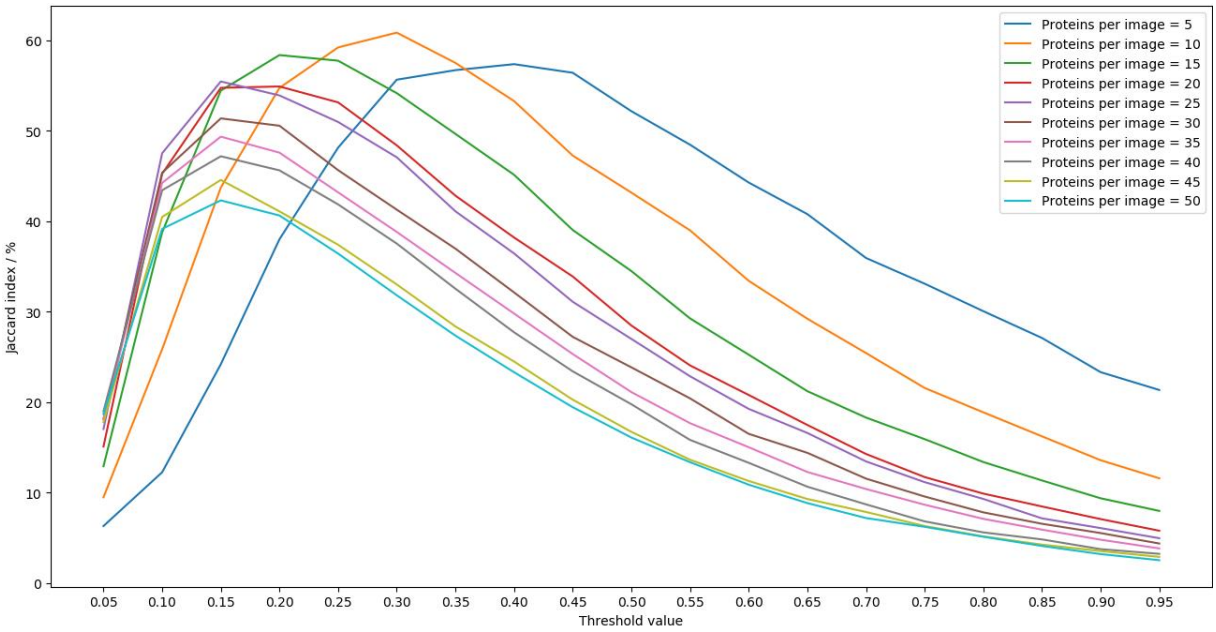

Figure 11. Jaccard index for varying number of proteins per image at different threshold values. X axis represent threshold value, y represent Jaccard index. Legend in the upper right corner annotates line with corresponding number of proteins per image.

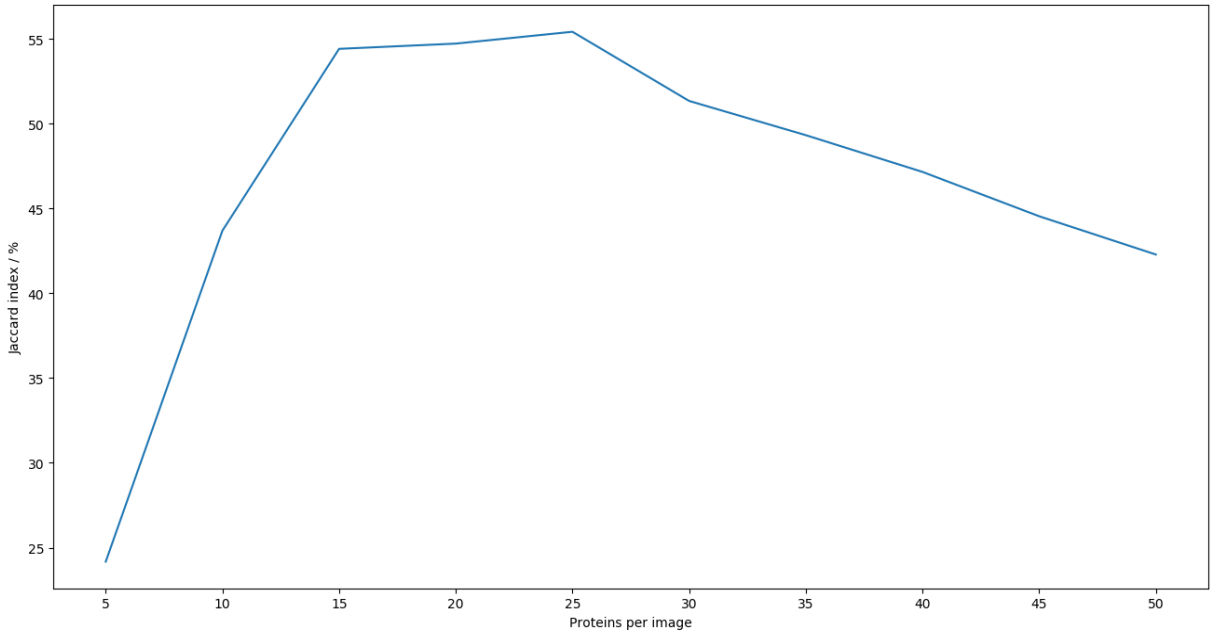

Figure 12. Jaccard index value for images with different number of proteins per image at threshold value 0.15. X axis represents number of proteins per image, y represents Jaccard index value.

Table 1. Table containing maximum values for Jaccard index across a range of proteins per image and threshold at which it is achieved.

Number of proteins per image	Threshold	Jaccard index / %
5	0.40	57
10	0.30	60
15	0.20	58
20	0.20	54
25	0.15	55
30	0.15	51
35	0.15	49
40	0.15	47
45	0.15	44
50	0.15	42

Other metric used for validation, beside the Jaccard index, is RMSE. RMSE values for images with different number of proteins per image at different threshold values are shown in Figure 14. Images used for Jaccard index testing were also used for RMSE testing.

Figure 15. is used to better show dependence between number of proteins per image and RMSE. It shows dependency at 0.95 threshold values. That threshold value was chosen because RMSE is the smallest across different image sets. Table 2. shows lowest RMSE value and corresponding threshold value for different number of proteins per image.

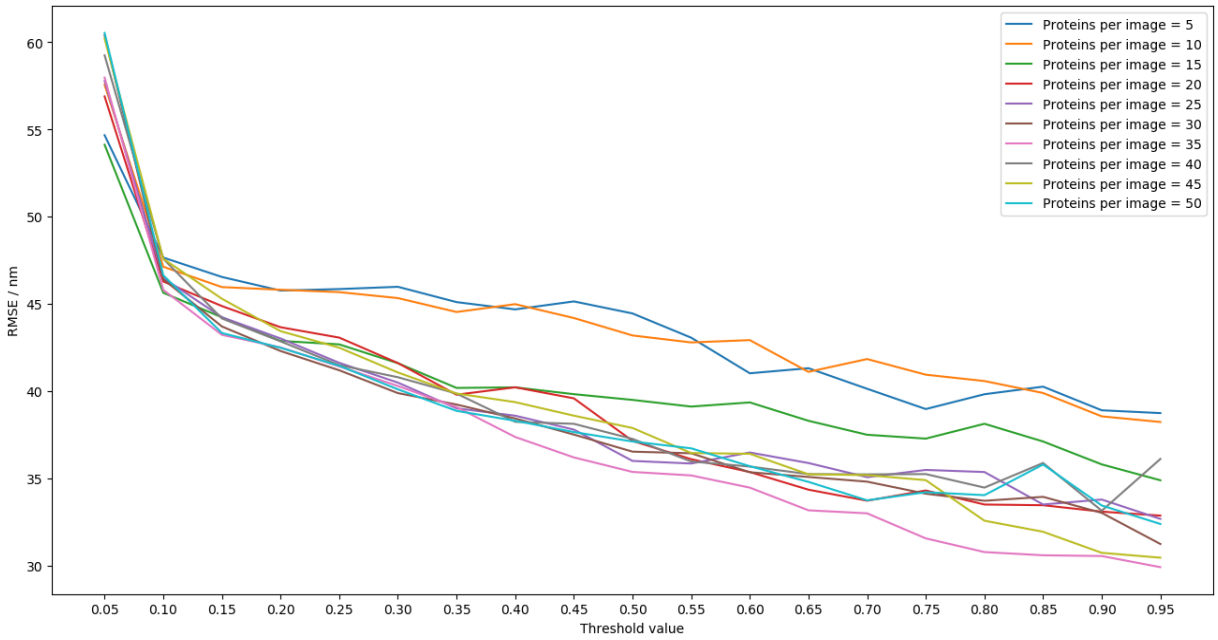

Figure 14. RMSE for varying number of proteins per image with different threshold values. X axis represent threshold value, y represent RMSE, legend in the upper right corner annotates line with corresponding number of proteins per image.

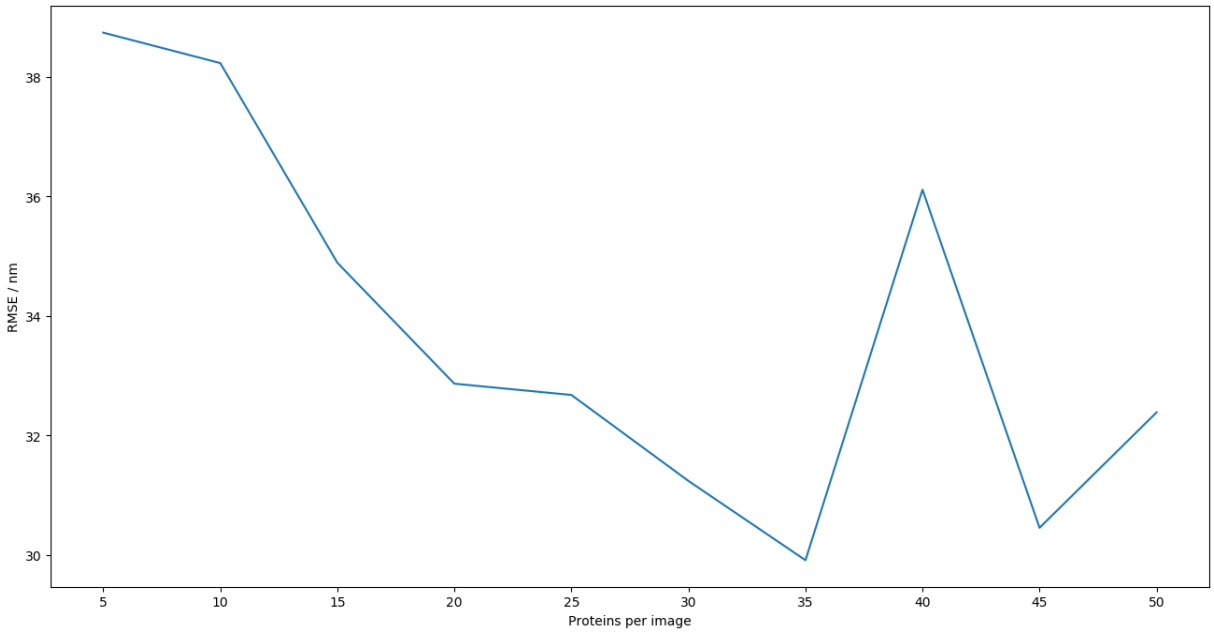

Figure 15. RMSE value for images with different number of proteins per image at threshold value of 0,95.

Table 2. Table containing minimum values for RMSE across range of proteins per image and threshold at which it is achieved.

Number of proteins per image	Threshold	RMSE / nm
5	0.95	38
10	0.95	38
15	0.95	34
20	0.95	32
25	0.95	32
30	0.95	31
35	0.95	29
40	0.90	33
45	0.95	30
50	0.95	32

4. Discussion

The goal of this graduation thesis was to train a CNN network to localize proteins from super-resolution microscopy images and to test network performance on never seen super-resolution images. To test network performance Jaccard index and RMSE were chosen. Those two parameters are currently used in the field of localization microscopy.

To show the necessity and usefulness of machine learning in localization prediction it is important to consider Figure 10. showing localization algorithm at work. As it can be seen signal strength and size do not perfectly correspond to network localization probability for that area. It is also important to note how some low signal spread out areas can be localized as one localization and other as multiple localization. This shows how network mapping from super-resolution images to localization probabilities is nonlinear and hard for humans to understand and code. Coupled with ability to simulate data this makes localization problem suitable for machine learning. It has been shown how similar problems use of machine learning has led to reduction in running time and better performance (Boyd et al.2018, Nehme et al. 2018). With the development of easy-to-use CNN packages for different programming languages this approach makes it easy to adopt this model to similar problems in a future with slight modification.

For Jaccard index two trends can be observed. First is the decline in Jaccard index for images with higher number of proteins per image. This observation is in line with previous work on localization and super-resolution image reconstruction (Boyd et al.2018). This phenomenon is caused by overlapping signals from different fluorophores close to one another. With increasing number of proteins, it is more likely to get areas with signal overlaps that are hard to localize. A second phenomenon is related to reduction in the optimal threshold value for images with more proteins per image. Figure 8. shows how network trained on images with bigger number of fluorophores naturally expects a bigger number of fluorophores in an image and threshold needs to be increased to remove low probability localization. It can be postulated that a network trained on a smaller number of fluorophores per image would give less low probability positions, on the other hand network train that way would under perform on images with high number of proteins. It can be seen in Figure 11. that network optimizes its performance at the threshold value of 0.15 meaning a lot of low value position predictions are accurate. First part of Figure 12. shows low values for Jaccard index. It is important to note that this is related to suboptimal threshold value for a range between 5 and 15 proteins per image.

The first observation for RMSE comes from Figure 15. showing a trend toward smaller values for images with bigger number of proteins per image and that increase after 35 proteins per image. For the decrease between 5 and 35 proteins per image it can seem contradictory if we consider a network performance problem in areas with overlapping signals. Possible explanations can be that network is trained on images with higher number of proteins and thus is better tuned to those cases. Increase in RMSE after 35 proteins per image can be contributed to overlapping signals. The second observation is in Figure 14. showing trend toward smaller RMSE for bigger threshold values. With an increasing threshold number of predicted

localizations and their area is reduced. It can be concluded that localizations for which network probability localization is high is most likely correct and close to real localization position.

To localize as many correct positions as possible, it is better to optimize for Jaccard index and use smaller threshold value. If goal is to find only the most precise localizations threshold should be set to high value to minimize RMSE. This dichotomy should be solved based on a problem at hand.

As machine learning task can perform only as good as the data they have been trained on and are required to work with, it is important to discuss images used in this graduation thesis. Code 1. used to create images introduces a lot of stochasticity to date. Stochasticity is introduced through the position and number of fluorophores around proteins. For this reasons Jaccard index is relatively small and RMSE is relatively large. If the images had higher amount of ambiguity Jaccard index would be even smaller and RMSE bigger, if images have a less ambiguity Jaccard index would be bigger and RMSE smaller. In practical applications data used to train neural networks should be as similar to experimental super-resolution images as possible. To secure the best performance, experimental super-resolution images with their protein localization should be used to train the network. This is not always possible and simulated images can be used, but performance will possibly drop compared to experimental super-resolution images. To optimize localization performance it is also advised to modified experiment to have a relatively constant number of proteins per image and to avoid having too many fluorophores per image.

Field of super-resolution microscopy for protein localization and counting is novel, as a consequence, there is a lack of consistency in reporting same quality metrics (Boyd et al.2018). CNN in this graduation thesis is compared with two methods that perform similar task: Alternating Descent Conditional Gradient method (ADCG) (Boyd et al. 2017) and DeepLoco (Boyd et al. 2018). This graduation thesis will use values reported by the DeepLoco paper because they tested the best algorithms in the field for Jaccard index and RMSE metric. DeepLoco is based on CNN, while ADCG is based on combines nonconvex and convex optimization techniques to remove the blur caused by diffraction. This comparison is not perfect because ADCG and DeepLoco localize fluorophores, and not the proteins that those fluorophores correspond to. Another difference is in data generation, DeepLoco method is not completely equal to the method used in this graduation thesis. Their results show DeepLoco RMSE in the range from 11,61 to 18,22 and ADCG RMSE in the range from 14,95 to 19,36 for low and high number of proteins per image respectively. And DeepLoco Jaccard index in the range from 0,89 to 0,61 and ADCG Jaccard index in the range from 0,79 to 0,53 for low and high number of proteins per image respectively. It is important to note that low number of proteins per image reported in DeepLoco paper is lower than the one used in this graduate thesis. From the comparison of the results obtained by Boyd et al. 2018 it can be concluded, that RMSE and Jacard index have worse performance for images with a small number of proteins per image. This can be contributed to CNN training with images with high number of proteins per image, and can be fixed by using images with a smaller number of proteins per image for CNN training. Jaccard index and RMSE approach values similar to those of DeepLock and ADCG when the

number of proteins per image is high. This suggests the importance of CNN training with a representative data set. Because ADCG and DeepLoco localize fluorophores, and not proteins labeled by these fluorophores it is expected for them to have better performance due to them not needing to combine multiple fluorophores to calculate one protein position. By getting Jaccard index and RMSE values similar to ones gotten for a simpler task, it can be concluded that the CNN developed in this graduation thesis has a performance comparable to state of the art algorithms used in the field.

5. Conclusion

After CNN was created and trained and its performance was evaluated on Jaccard index and RMSE following conclusions can be made:

1. CNN encoder-decoder architecture is a good option for solving protein localization problem in super-resolution imaging. Further algorithms can be used to interpret CNN probability distribution and give point localizations for proteins.

2. Proteins localized by high threshold will be closer to real protein localization then those localized with low threshold value, but algorithm will under estimate number of proteins.

3. Proteins localized by low threshold will be further away from real protein localization then those localized with high threshold value, but number of proteins localized will be closer to reality.

6. References

Betzig E., Patterson G. H., Sougrat R., Lindwasser O. W., Olenych S., Bonifacino J. S., Davidson M. W., Lippincott-Schwartz J., Hess H. F. (2006): Imaging intracellular fluorescent proteins at nanometer resolution. Science 313 (5793): 1642–1645.

Bishop C. M. (2006): Pattern Recognition and Machine Learning. Springer-Verlag, Secaucus.

Boyd N., Jonas E., Babcock H. P., Recht B. (2018): Deeploco: Fast 3d localization microscopy using neural networks. BioRxiv: 267096.

Boyd N., Schiebinger G., Recht B. (2017): The alternating descent conditional gradient method for sparse inverse problems. SIAM Journal on Optimization 27(2): 616-639.

Bre F., Gimenez J. M., Fachinott V. D. (2018): Prediction of wind pressure coefficients on building surfaces using artificial neural networks. Energy and Buildings 158: 1429–1441.

Chollet F. (2015): Keras. https://github.com/fchollet/keras.

Fricke F., Beaudouin J., Eils R., Heilemann,M. (2015): One, two or three? Probing the stoichiometry of membrane proteins by single-molecule localization microscopy. Scientific reports 5: 14072.

Goodfellow I., Bengio Y., Courville A. (2016): Deep Learning. MIT Press, Cambridge.

Hornik K. (1991): Approximation capabilities of multilayer feedforward networks. Neural networks 4 (2): 251–257.

Lee S.-H., Shin J. Y., Lee A., Bustamante C. (2012): Counting single photoactivatable fluorescent molecules by photoactivated localization microscopy (PALM). Proceedings of the National Academy of Sciences 109 (43): 17436–17441.

Nehme E., Weiss L. E., Michaeli T., Shechtman Y. (2018): Deep-STORM: super-resolution single-molecule microscopy by deep learning. Optica 5 (4): 458 - 464.

Nicovich P.R., Owen D.M., Gaus K. (2017): Turning single-molecule localization microscopy into a quantitative bioanalytical tool. Nature protocols 12 (3): 453.

Nieuwenhuizen R.P., Bates M., Szymborska A., Lidke K.A., Rieger B., Stallinga S. (2015): Quantitative localization microscopy: effects of photophysics and labeling stoichiometry. PloS one 10 (5): p.e0127989.

Rojas R. (1996): Neural Networks - A Systematic Introduction. Springer-Verlag, Berlin.

Rollins G. C., Shin J. Y., Bustamante C., Pressé S. (2014): Stochastic approach to the molecular counting problem in superresolution microscopy. Proceedings of the National Academy of Sciences 112 (2): E110–E118.

Rust M. J., Bates M., Zhuang X. (2006): Sub-diffraction-limit imaging by stochastic optical reconstruction microscopy (STORM). Nature Methods 3 (10): 793–795.

Von Chamier L., Laine R. F., Henriques R. (2019): Artificial intelligence for microscopy: what you should know. Biochemical Society Transactions 47 (4): 1029–1040.

Wilson D. R., Martinez T. R. (2003): The general inefficiency of batch training for gradient descent learning. Neural Networks 16 (10): 1429–1451.

7. Appendix

Code 1. Simulation of input/ output images

```
import numpy as np
import scipy as scipy
import matplotlib.pyplot as plt
import pylab
from itertools import chain
from scipy.spatial.distance
import pdist, squareform
from scipy.special import erf
from PIL import Image

def generate_molecules(pixels,n,min_d):
        xn=np.random.uniform(0,(pixels-1),n)
        yn=np.random.uniform(0,(pixels-1),n)
        z=np.stack((xn,yn), axis=-1)
        D = squareform(pdist(z)) + 1000*min_d*np.identity(n)
         while D.min()<min_d:
                nd1, ind2 = np.where(D <min_d)
                 unique = (ind1 < ind2)
                ind1 = ind1[unique]
                xn[ind1]=xn[ind1]+np.random.normal(0,min_d,ind1.shape[-1])
                yn[ind1]=yn[ind1]+np.random.normal(0,min_d,ind1.shape[-1])
                z=np.stack((xn,yn), axis=-1)
                D = squareform(pdist(z)) + 1000*min_d*np.identity(n)
         return [xn,yn]

def frame_simulation(pixels,xn,yn,stdev, disp_locs):
        n=xn.shape[-1]
```

```
        x=[]
        y=[]
        localization=np.random.geometric(p=0.2, size=n)
        for i in range(0,n):
            Xij = np.random.normal(loc=0,scale=stdev, size=int(localization[i]))
            Yij = np.random.normal(loc=0,scale=stdev, size=int(localization[i]))
            x.append(xn[i] + Xij)
            y.append(yn[i] + Yij)
        x_unchain= np.asarray(list(chain.from_iterable(x)))
        y_unchain= np.asarray(list(chain.from_iterable(y)))
        if disp_locs==1:
            fig = plt.figure(1, figsize=(5.5,5.5))
            plt.scatter(xn,yn,c='b',marker='+')
            plt.scatter(x_unchain,y_unchain,c='r',marker='.')
            plt.axis('equal')
            plt.xlim((0,pixels))
            plt.ylim((0,pixels))
            plt.gca().invert_yaxis()
            plt.show()
        return [x_unchain,y_unchain]


def MolKernel(x_inx,y_inx,x_pos,y_pos,stdev):
        [xx,yy] = np.meshgrid(x_inx,y_inx)
        img_kernel = (0.5*(erf(  np.true_divide( (xx-x_pos+0.5),(np.sqrt(2)*stdev) )  ) - erf(
        np.true_divide(  (xx-x_pos-0.5),(np.sqrt(2)*stdev) )  )))*( 0.5*(erf(   np.true_divide( (yy-
        y_pos+0.5),(np.sqrt(2)*stdev)  )  ) - erf(  np.true_divide( (yy-y_pos-0.5),(np.sqrt(2)*stdev) )
        )))
        img_kernel=img_kernel/img_kernel.sum()
        return img_kernel


def map_into_image(x,y,pixels, fact, st):
```

```
        M=np.zeros((pixels*fact,pixels*fact))
        xx=np.linspace(0,fact*pixels-1,fact*pixels)
        yy=np.linspace(0,fact*pixels-1,fact*pixels)
        for i in range(0,x.shape[-1]):
                img=MolKernel(xx,yy,fact*x[i],fact*y[i],fact*st)
                M=M+img
                 M=M/M.max()
        return np.asarray(np.uint16(M*(2**16-1)))

def protein_position_cord_sys(x, y, pixels, fact):
        x = x.astype(int)
        y = y.astype(int)
        M=np.zeros((pixels*fact,pixels*fact))
        for i in range(0,len(x), 1):
                M[y[i],x[i]] = 1
        return M
```

Code 2. Image normalization and standardization

```
import numpy as np

def preprocessing (images):
        images = images.astype('float32')
         for i in range(0,images.shape[0]):
                image = images[i]
                image = np.squeeze(image)
                min_val = image.min()
                max_val = image.max()
                image = (image - min_val)/(max_val - min_val)
                isd = np.std(image)
```

```
            imean = np.mean(image)
            image = (image- imean)/isd
            images[i] = image
    return images
```

Code 3. Image formatting for CNN and CNN architecture plus code used to save training process and trained CNN

```
from os.path import exists
from os import mkdir
import numpy as np
import tensorflow as tf
from keras import backend as K
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras.layers import Input, Activation, UpSampling2D, Convolution2D, MaxPooling2D,
BatchNormalization
from sklearn.model_selection import train_test_split
from keras.models import model_from_json, Model
from keras import backend as K
from keras import optimizers
from keras import losses
import math
from PIL import Image

def CNN_model_1(train, result, n, dirName, n_epoc, batch_size_n):
    train = preprocessing(train)
    image_train, image_test, result_train, result_test = train_test_split(train, result,
    test_size=0.3, random_state=42)
    n_pixel_image = image_train.shape[1]
    n_pixel_result = result_train.shape[1]
    image_train = image_train.reshape(image_train.shape[0], n_pixel_image, n_pixel_image, 1)
```

```
result_train = result_train.reshape(result_train.shape[0], n_pixel_result, n_pixel_result, 1)
image_test = image_test.reshape(image_test.shape[0], n_pixel_image, n_pixel_image, 1)
result_test = result_test.reshape(result_test.shape[0], n_pixel_result, n_pixel_result, 1)
def conv_bn_relu(nb_filter, rk, ck, name):
    def f(input):
        conv = Convolution2D(nb_filter, kernel_size=(rk, ck), strides=(1,1),
        padding="same", use_bias=False,
        kernel_initializer="Orthogonal",name='conv-'+name)(input)
        conv_norm = BatchNormalization(name='BN-'+name)(conv)
        conv_norm_relu = Activation(activation = 'relu',name='relu-
        '+name)(conv_norm)
        return conv_norm_relu
    return f
def CNN(input, name):
    f1 = conv_bn_relu(16,5,5,name+'F1')(input)
    f2 = conv_bn_relu(16,5,5,name+'F2')(f1)
    pool1 = MaxPooling2D(pool_size=(2,2),name=name+'Pool1')(f2)
    f3 = conv_bn_relu(32,5,5,name+'F3')(pool1)
    f4 = conv_bn_relu(32,5,5,name+'F4')(f3)
    pool2 = MaxPooling2D(pool_size=(2,2),name=name+'Pool2')(f4)
    f5 = conv_bn_relu(64,5,5,name+'F5')(pool2)
    f6 = conv_bn_relu(64,5,5,name+'F6')(f5)
    pool3 = MaxPooling2D(pool_size=(2,2),name=name+'Pool3')(f6)
    f7 = conv_bn_relu(128,5,5,name+'F7')(pool3)
    f8 = conv_bn_relu(128,5,5,name+'F8')(f7)
    up1 = UpSampling2D(size=(2, 2),name=name+'Upsample1')(f8)
    f9 = conv_bn_relu(64,5,5,name+'F9')(up1)
    f10 = conv_bn_relu(64,5,5,name+'F10')(f9)
    up2 = UpSampling2D(size=(2, 2),name=name+'Upsample2')(f10)
    f11 = conv_bn_relu(32,5,5,name+'F11')(up2)
```

```
        f12 = conv_bn_relu(32,5,5,name+'F12')(f11)
        up3 = UpSampling2D(size=(2, 2),name=name+'Upsample3')(f12)
        f13 = conv_bn_relu(16,5,5,name+'F13')(up3)
        f14 = conv_bn_relu(16,5,5,name+'F14')(f13)
        up4 = UpSampling2D(size=(2, 2),name=name+'Upsample4')(f14)
        f15 = conv_bn_relu(8,5,5,name+'F15')(up4)
        f16 = conv_bn_relu(8,5,5,name+'F16')(f15)
return f16
def buildModel(input_dim_1):
        input_ = Input (shape = (input_dim_1))
        act_ = CNN (input_,'CNN')
        density_pred = Convolution2D(1, kernel_size=(1, 1), strides=(1, 1), padding="same",
       activation="sigmoid", use_bias = False,
       kernel_initializer="Orthogonal",name='Prediction')(act_)
        model = Model (inputs= input_, outputs=density_pred)
        opt = optimizers.Adam(lr=0.001)
        model.compile(optimizer=opt, loss = 'binary_crossentropy')
return model
 checkpointer = ModelCheckpoint("%s/Model_N_prot_%d_weights.h5" % (dirName, n),
verbose=0, save_best_only=True, save_weights_only=True)
K.set_image_dim_ordering('tf')
 change_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5,
min_lr=0.00000005)
model = buildModel((n_pixel_image, n_pixel_image, 1))
 if not exists('%s/model_architecture.json' % dirName):
        with open("%s/model_architecture.json" % dirName, 'w') as f:
                f.write(model.to_json())
validation_loss = []
loss = []
```

```
        model_history = model.fit(image_train, result_train, epochs=n_epoc
        ,batch_size=batch_size_n,validation_data=(image_test, result_test),callbacks=[change_lr,
        checkpointer], verbose=2)

        validation_loss.append(model_history.history['val_loss'])

        validation_loss = np.array(validation_loss)

        validation_loss = validation_loss.transpose()

        loss.append(model_history.history['loss'])

         loss = np.array(loss)

        loss = loss.transpose()

        Out = np.column_stack((loss, validation_loss))

        np.savetxt('%s/Validation_loss_loss_N_prot_%d.dat' % (dirName, n), Out)
```

Code 4. Localization algorithm for CNN localization probabilities and validation

```
from skimage import io, measure

import numpy as np

from scipy.spatial.distance import cdist

import math


def jacc_ind_RMSE(model, network_input, network_output, threshold, pixsize, zoomfact):

        intersect = 0

        n_model = 0

        n_gt = 0

        dist_sqrt = 0

        n_assigned = 0

        n_pixel_image = network_input.shape[1]

        for im_ind in range(0,network_input.shape[0],1):

                image_x = network_input[im_ind]

                image_x = image_x.reshape(1,n_pixel_image, n_pixel_image, 1)

                prediction = model.predict(image_x)

                prediction = np.squeeze(prediction)
```

```
prediction = np.interp(prediction, (prediction.min(), prediction.max()), (0, 1))
prediction = measure.label(prediction > threshold)
centroids_model = measure.regionprops(prediction)
x_cor_m = []
y_cor_m = []
for i in range(0, len(centroids_model), 1):
        x_cor_m.append(centroids_model[i]['centroid'][1])
        y_cor_m.append(centroids_model[i]['centroid'][0])
model_cord = np.stack((x_cor_m,y_cor_m), axis = 1)
res_x = network_output[im_ind]
x_cor_gt = []
y_cor_gt = []
for i in range (0, res_x.shape[0], 1):
        for j in range(0, res_x.shape[1] ,1):
                if res_x[i,j] == 1:
                        x_cor_gt.append(j)
                        y_cor_gt.append(i)
gt_cord = np.stack((x_cor_gt,y_cor_gt), axis = 1)
euclidian_distance =  cdist(model_cord, gt_cord, metric = 'euclidean')
for i in range(0, euclidian_distance.shape[0], 1):
        mat_ind = np.unravel_index(euclidian_distance.argmin(),
        euclidian_distance.shape)
        min_value = euclidian_distance[mat_ind]
        euclidian_distance[mat_ind[0],:] = 'inf'
        euclidian_distance[:,mat_ind[1]] = 'inf'
        if min_value < 50/(pixsize/zoomfact):
                intersect += 1
                dis_nm = min_value * pixsize
                dist_sqrt += dis_nm * dis_nm
                n_assigned += 1
```

```
                n_model += len(model_cord)
                n_gt += len(gt_cord)
        jaccard_index = intersect / (n_gt + n_model - intersect)
        if n_assigned == 0:
                RMSE = math.inf
        else:
                squerd_mean_error = dist_sqrt / n_assigned
        RMSE = math.sqrt(squerd_mean_error)
return(jaccard_index, RMSE)
```

8. Curriculum vitae

Education

2017-2020: Master of Molecular Biology, University of Zagreb, Croatia

- Internship at the University of Vic laboratory for Quantitative BioImaging, Vic 2/2019 – 6/2019
- Graduation thesis: Application of convolutional neural network for protein counting and localization photographed by super-resolution microscopy - Adviser professor: Assoc. Prof Carlo Manzo and Assoc. Prof Damjan Franjević

2014-2017: Bachelor of biology, University of Zagreb, Croatia

- Student teaching associate at Microbiology course, Department of Biology, Faculty of Science, University of Zagreb
- Bachelor thesis: Slime molds (Mycetozoa) - doc. dr. sc. Sunčica Bosak

2010-2014: Medical School, Šibenik

Other projects:

- Night of Biology 2017 and 2018
- Student section for Bioinformatics at BIUS

  Member

- BIUS student research project
  Design and construction of a working microbial fuel cell

Languages

- English B2