Charles University
Faculty of Science

Study programme: Bioinformatics
Branch of study: Bioinformatics

Květa Brázdilová

Applications of Machine Learning for Detecting and Counting Objects in Cell Biology

Využití strojového učení pro rozpoznávání a počítání objektů v buněčné biologii

Bachelor´s thesis

Supervisor:

doc. Mgr. Pavel Stopka, Ph.D.

Prague, 2020

Prohlášení:

Prohlašuji, že jsem závěrečnou práci zpracovala samostatně a že jsem uvedla všechny použité informační zdroje a literaturu. Tato práce ani její podstatná část nebyla předložena k získání jiného nebo stejného akademického titulu.

V Praze, 4. 6. 2020

Podpis

Poděkování

Ráda bych na tomto místě poděkovala svému školiteli, doc. Mgr. Pavlu Stopkovi, Ph.D., za jeho podporu a důvěru v moje schopnosti.

Dále patří dík mému vedoucímu projektu Mgr. Ladislavu Peškovi, Ph.D., který to se mnou vydržel i přes velmi dlouhou práci na projektu. Jsem mu vděčná jak za jeho odborné znalosti, tak za vstřícnost a ochotu.

V neposlední řadě také děkuji svojí rodině za vytrvalou podporu, zejména pak své sestře za poskytnutí námětu pro tuto práci.

Acknowledgement

I would like to thank my supervisor doc. Mgr. Pavel Stopka, Ph.D. here, for his support and trust in my abilities.

Many thanks go to my project supervisor Mgr. Ladislav Peška, Ph.D., who did not give up on me despite the very long work on my project. I am grateful to him not only for his professional knowledge but also for his helpfulness and patience.

Last but not least, I would like to thank my family for their unending support, especially my sister for the idea for the topic of this thesis.

Abstract

Modern biological research generates large amounts of data, which require automation for efficient analysis. Lately, machine learning solutions are being developed for many of the problems in this field. This thesis focuses on applications of machine learning for image analysis, such as detecting cells in microscopy images and classifying them based on their phenotype. After a brief introduction to machine learning concepts, eight published methods are presented, which employ machine learning for either detecting and classifying, or counting objects in biological images. Five open-source software tools for biological image analysis, which employ some of the methods mentioned above, are introduced. A new project is also described, which aims to create a convolutional neural network for counting bacterial colonies in images of agar plates. The results of this project are discussed.

Keywords: machine learning, neural network, pattern recognition, cell biology, segmentation

Abstrakt

Současný biologický výzkum vytváří velké množství dat, která vyžadují automatizaci pro efektivní analýzu. V poslední době vznikají pro mnohé z těchto problémů řešení využívající strojové učení. Tato práce se zaměřuje na aplikace strojového učení pro analýzu obrazu, například detekci buněk v mikroskopickém snímku a jejich klasifikaci dle fenotypu. Po krátkém úvodu do strojového učení obecně bude představeno osm publikovaných metod využívajících strojové učení pro detekci nebo klasifikaci objektů v biologických snímcích. Dále bude uvedeno pět open-sourcových softwarových nástrojů pro obrazovou analýzu v biologii, které využívají některé z metod zmíněných výš. Dále je popsán nový project, jehož cílem je vytvořit konvoluční neuronovou síť na počítání bakteriálních kolonií na agarových plotnách. Na závěr jsou diskutovány výsledky tohoto projektu.

Klíčová slova: strojové učení, neuronová síť, rozpoznávání objektů, buněčná biologie, segmentace

Table of Contents

1. Introduction

Modern technologies used in biology often generate data at a much higher rate than scientists are able to analyze it. Much of this data is in the form of images, as imaging has become an important part of many methods – fluorescent microscopy, single cell analysis or histological images. Large amounts of information can be extracted from image data, but sometimes only simple answers are required, such as whether or not an object is present, where in the image is it or how many are there. This is a matter of a single glance for a human, but when there are hundreds or thousands of images to analyze and label, the task becomes laborious or even nearly impossible.

Typically, simple but repetitive and time-consuming tasks are best tackled by machine learning approaches. These may perfectly substitute the role of a human viewing all data or at least allow qualified personnel to focus on more important parts of their research. Still, many scientists are not aware of the possibilities or are unsure of where machine learning could be helpful. (Min et al., 2017) Detecting and classifying objects is a common task in which humans still outperform computers, but the technology is rapidly advancing. Counting objects in images is a different, though similar problem, and can now also be solved using machine learning.

The objective of this thesis is to provide a brief overview of some machine learning methods used to detect and count objects in cell biology. In the second part, a convolutional neural network approach to counting bacterial colonies in a photograph of an agar plate will be discussed.

1.1. Machine Learning

Machine learning (ML) is a subset of artificial intelligence, but it also makes use of knowledge from various fields, such as statistics or psychology. Machine learning algorithms use input data to solve a problem without being explicitly programmed to give a specific result. During so-called training, these algorithms use input data to find patterns and create a model for predicting the given outcome. This model contains a set of parameters, which are adjusted during training, rather than being known in advance and set by a human. Patterns found in the training data set are then generalized and used to infer the results for any other data. The learning process can either be done just once, or the algorithm can continually improve itself as it gathers more data. (Bishop, 2006; El Naqa & Murphy, 2015)

Machine learning approaches can be divided into two main categories, supervised and unsupervised learning. The first requires a set of labeled examples for training, which has to be produced by a human expert. If there is a finite set of possible outcomes, such as cell type or diseased and normal phenotype, we speak of classification. If the result is a continuous variable, for example size or weight, we use the term regression. Unsupervised learning uses a training set without labels, therefore

its output is not explicitly specified in advance. The examples can be clustered into groups based on their similarity, or density estimation is performed, which determines the distribution of data in the input space. A combination of these two options is also used, which is called semi-supervised learning.

Performance of the model can be improved by using an ensemble method. This involves aggregating the prediction of different models, of the same type or different. One way to involve multiple models is by introducing an element of randomness. Individual sub-models are trianed on random subsets of data, which is called bagging, with other optional improvements. The final prediction can be the majority of the individual predictions, or a more sophisticated method involving probabilities may be employed.

Another approach to ensemble methods is boosting, where simple models are created sequentially and each next one aims to correct the errors of the previous one. To this end, missclassified instances may be multiplied or assigned a higher weight. In this case, the resulting prediction is usually a weighted average of the outputs. (Geurts et al., 2009)

Many different ML algorithms exist, only several will be discussed here.

1.1.1. Regression Analysis

In regression analysis, the aim is to find a function expressing the relationship of one or more independent variables and the outcome variable. The most basic is linear regression, where a linear function is fit to the data. More complex solutions include polynomial regression, or use of other non-linear functions, such as logarithmic or exponential.

1.1.2. k-NN

k nearest neighbours, or *k*-NN, is another simple ML algorithm. The training involves only storing labelled examples. Then, for prediction of a test sample, the *k* most similar training samples are found and the output is determined as the most common class among the *k*, for classification, or their average, for regression.

1.1.3. Tree based methods

A decision tree is a set of binary tests structured into a tree shape, much like a flow chart (Figure 1).

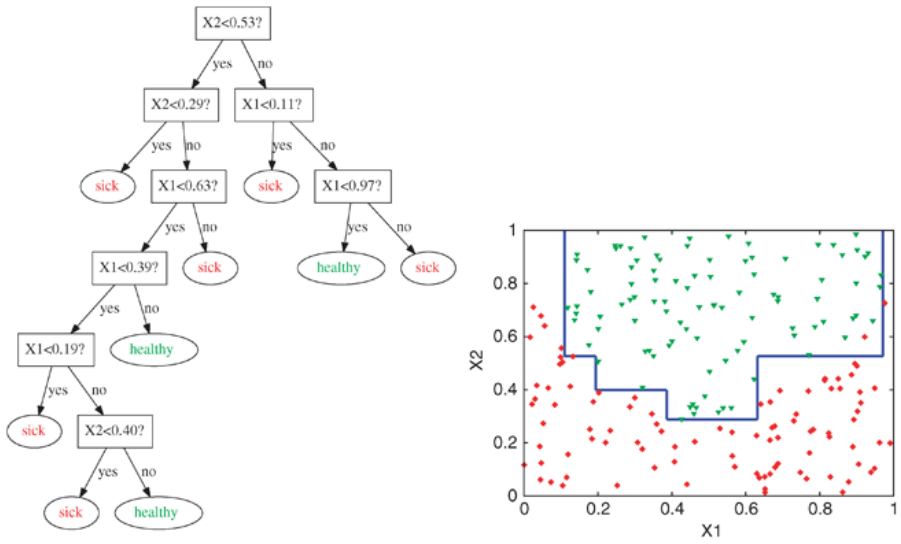

Figure 1 – Decision tree with corresponding data (Geurts et al., 2009)

The answers to consecutive tests ultimately produce the resulting class.

During the learing phase, the tree starts out as one node. Subsequent nodes with tests are added and the sample set is split, until a criterion is satisfied. When adding a test, one from a candidate set is selected based on a scoring measure. This leads to adding of tests, which provide a better separation of classes. All leaves are then labelled either by the majority class of samples present or in a more sophisticated way involving probabilities. To prevent overfitting (see next section), either stop-splitting criteria are applied while building the tree, or some of its parts are removed afterwards, which is called pruning.

As with other ML algorithms, ensemble methods may be used with decision trees. A widely used model is the Random Forest, which builds several trees on random data subsets. Additionally this algorithm can be improved e.g. by selecting only certain features in different nodes (Breiman, 2001). Boosting methods can also be applied to decision trees. (Geurts et al., 2009)

1.1.4. Artificial neural networks

This approach uses an artificial neural network with multiple layers to extract features from data, which are then used for prediction. Each layer consists of neurons, which are connected to the following layer in a certain way. A neuron gives an output based on its inputs and activation function and sends this to the neurons in the next layer to which it is connected.

A specific type of neural network is a convolutional neural network (CNN), which increases the receptive field on each layer by integrating information from a wider area through a convolution filter (illustrated in Figure 2). This makes use of the fact that signals from close locations are more likely to be linked on a lower level. CNNs draw inspiration from the human visual cortex, different parts of which pass on the signal to identify increasingly complex features of the image, similarly to layers in an artificial network. Compared to fully connected neural networks, CNNs have much fewer trainable parameters, which reduces training time and helps prevent overfitting. For image processing, the whole image is usually used as input, which saves human researchers the laborious task of choosing and extracting features themselves. Outside of biology, deep learning is widely used for speech, text and image recognition.

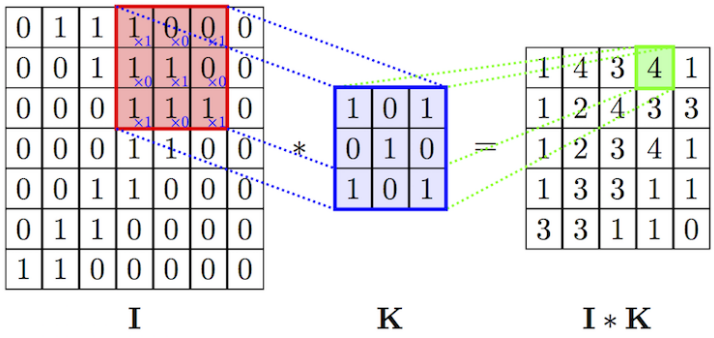

Figure 2 - Illustration of applying a convolutional filter. (Vo, 2018)

To reduce dimensions of the data, convolutions are typically alternated with pooling layers, where the information from multiple neurons is combined into one. A commonly used variant is max pooling, where the maximum of neurons in a sliding window is used. One other option is average pooling. The principle of both is illustrated in Figure 3. The most widely used activation function in CNNs is ReLU, or rectified linear unit, which returns the positive part of its input. Training a network also requires a loss function, which we aim to minimize during training by updating weights of neuron inputs by backpropagation into the hidden layers. To this end, stochastic gradient descent is used, which uses an estimation of the gradient, or slope, of the loss function to progress towards the minimum. An optional addition is momentum, which uses not only the gradient, but also the previous update, to update weights.

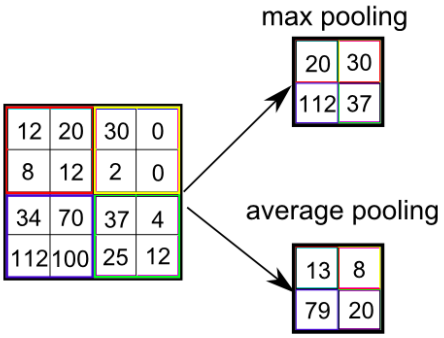

*Figure 3 - Max and average pooling. (*What Is Max Pooling in Convolutional Neural Networks? - Quora*, n.d.)*

1.1.5. SVM

Support vector machines (SVM) are a machine learning algorithm used to separate classes of samples in multidimensional space. When given a training set of examples belonging to two classes, the SVM tries to find a hyperplane to separate the classes from each other. A hyperplane in an N – dimensional space is an N-1 – dimensional subspace, that is a single point for one-dimensional data, a line in two-dimensional space or a plane in 3D space and similarly this applies to higher dimensions as well. There may be multiple separating hyperplanes, so the one with the greatest margin is chosen, in other words the one with the highest distance to the nearest datapoints. If a perfect separation does not exist, several examples may be allowed to be on the wrong side of the hyperplane. (Noble, 2006)

1.1.6. k-means clustering

k-means clustering may be mentioned as a basic exampe of unsupervised learning. It serves to divide data into *k* clusters, where each datapoint belongs to the cluster whose mean is closest to the sample in a *d*-dimensional space. Various heuristic algorithms exist to achieve this clustering.

1.2. Applications of machine learning in biology

In biological and biomedical research, there are many different tasks, which can be automated by using a machine learning solution. These include such diverse problems as counting crops in fields (Dijkstra et al., 2019), disease diagnostics (Fatima & Pasha, 2017), gene expression prediction (Chen et al., 2016), predicting secondary structure from protein sequence (Wang et al., 2016), SNP identification (Korani et al., 2019) and microscopy image analysis (Feizi et al., 2016).

Although various machine learning algorithms are applicable to these problems, recently deep learning is experiencing significant progress and seems to find use in many different fields. Within life sciences, it can be applied to problems of recognition of objects in biomedical imaging, decoding brain behavior, protein classification and many of the tasks mentioned above. (Min et al., 2017)

Here we will only focus on a small subset of the aforementioned problems.

1.3. Challenges of Machine Learning

As much as machine learning is a powerful tool, like everything, it has its weak points. Here some general pitfalls and also obstacles specific to biological data are mentioned. In each use case, many other caveats have to be considered, but these are beyond the scope of this thesis.

Firstly, in all cases, the efficiency of a machine learning method can only be as good as the input data we use. For supervised learning used in cell biology, this means tens or hundreds of thousands of images hand-annotated by human experts. Obtaining this kind of data is extremely costly and time consuming, maybe nearly impossible. Some authors bypass this problem by training on synthetic data, which can yield a model performing well when applied to real data (Xie et al., 2018).

The model needs to be specific enough in order to provide accurate predictions, however, an overcomplicated model, especially in combination with a small dataset, can result in overfitting. This means that the parameters of the model follow the training data too closely and detect the noise instead of general rules. When different data is presented to the model, it will make incorrect predictions, since it is only trained for the original data and its specificities. Maybe the easiest way to prevent overfitting is to choose a simple algorithm, but other measures are also advisable, such as cross validation (Dietterich, 1995) or penalization of model complexity while training.

On the other hand, the model must be complex enough to express the problem, otherwise it will suffer from bias and no amount of training will result in a good separation of the data. This balance between learning not enough and too much is called the bias/variance tradeoff. (Geurts et al., 2009)

In addition, the dataset has to be split into at least two parts, the train set and test set. This ensures that there will be additional data that the model has not encountered yet and is therefore suitable for testing its performance. This is the basic form of cross validation. Another option is to split the dataset into k parts and train the model k times. Each time a different subset will be kept aside for validation and the remaining $k–1$ parts used for training. (Gupta, 2017)

2. Approaches to detecting objects

The first problem addressed here is detecting objects in images, in other words determining the location and type of an object, or several, in an image. To this end, segmentation is usually used, which means dividing the image into partitions, or, more formally, assigning each pixel in the image a label. This label may distinguish background from foreground, or may divide pixels into more classes, such as individual objects. Most segmentation methods do not involve any machine learning. A widely used method with many existing modifications is thresholding. The image is first converted to grayscale and then the background and foreground are separated based on a threshold for pixel value. The objects in the foreground may be identified as connected components. Another possible non-ML approach is computing a watershed, which is inspired by a watershed in the geological sense. Pixel values represent elevation and metaphorically speaking, the image is segmented along lines running along tops of ridges.

A simple ML segmentation method uses *k*-means clustering, which selects *k* cluster centers and assigns each pixel to a cluster based on a distance metric, which may involve pixel values as well as Euclidean distance. Then cluster centers are recalculated as cluster means and the process is iterated until convergence.

Many more or less complex machine learning algorithms may be used for segmentation, examples of which are mentioned below.

2.1. U-net

The U-net was designed for microscopy image segmentation, more precisely separating nuclei from the background. The architecture is a so-called fully convolutional network, which combines the usual contracting network with subsequent layers performing upsampling. In the first part of the network, subsequent convolutions followed by ReLU (rectified linear units) are performed, then max pooling, decreasing dimensions and doubling channels. In the expanding part of the network, upsampling is performed together with convolution, which halves the channel numbers, then the feature map is concatenated with the output of the corresponding layer in the contraction part of the network to provide localization information. Two further convolutions are performed followed by ReLU. The architecture is illustrated in Figure 4. Larger feature channel numbers help propagate context information into deeper layers of the network, but that requires larger available memory. This is addressed by dividing images into tiles, context for edge tiles is provided by mirroring.

The model was trained on a set of images with corresponding segmentation maps. Since these are not readily available in large quantities, the dataset was augmented by applying deformations to the

existing images. Apart from shift and rotation, elastic deformations were applied by randomly displacing points of a grid and interpolating the surrounding values. During training, stochastic gradient descent was used with high momentum, so that the previous images would influence the one being processed. The predicted value for each pixel was obtained by a soft-max function on the final feature map giving a vector with a value near 1 for the index with maximal activation and near 0 for the others.

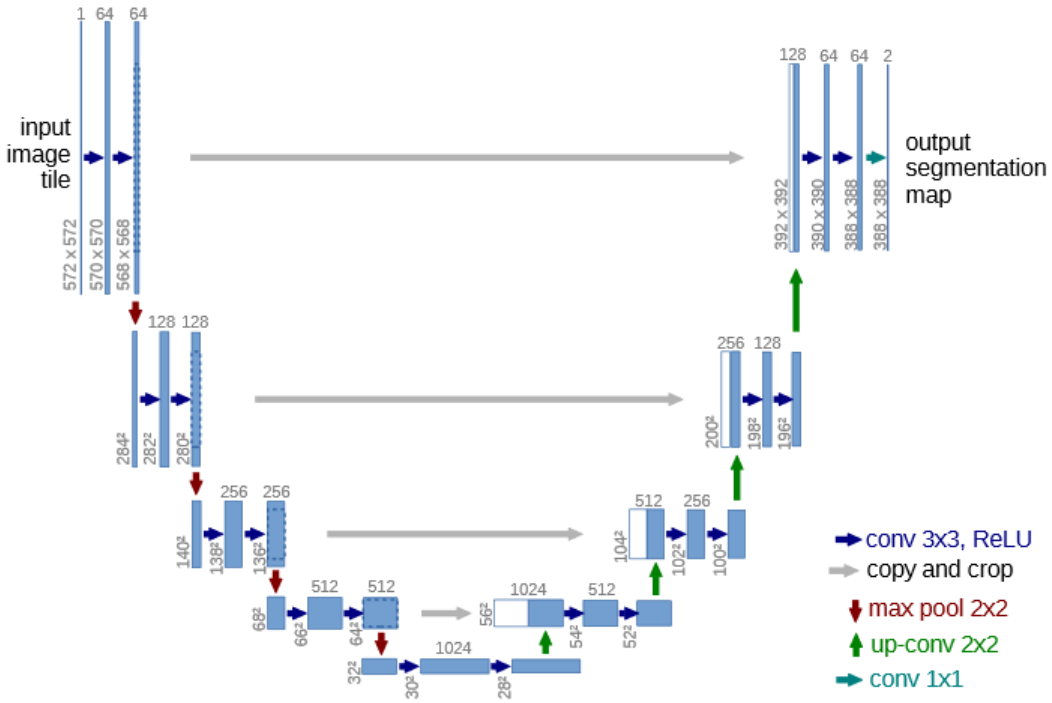

Figure 4 - U-net architecture. Blue boxes represent feature maps with channel numbers written on top and dimensions on the left. White boxes are copied feature maps and arrows coorespond to operations. (Ronneberger et al., 2015)

The loss function, defined as

$$E = \sum_{x \in \Omega} w(x) \log\left(p_{l(x)}(x)\right)$$

determines the deviation of $p_{l(x)}(x)$ from 1, where $p_k(x)$ is the approximated maximum function at pixel x, $l(x)$ is the correct label of pixel x, and w, the weight map, is used to give more importance to the narrow border between touching objects in a segmented image.

This network architecture was tested on three datasets within two segmentation competitions. The first dataset contained 30 electron microscopy images of Drosophila neuronal structures, the second consisted of 35 Glioblastoma-astrocytoma cell images taken by phase contrast microscopy and the third comprised 20 images of HeLa cells recorded by differential interference contrast. In each of these cases, the U-net outperformed the best algorithm at the time, except for one case with very dataset-specific postprocessing. (Ronneberger et al., 2015)

2.2. Faster R-CNN

Many progressive deep learning models with ever improving results exits, but few of them have been applied to biological problems. Also, implementations of published methods are rarely available, making it more difficult to replicate them. In a recent publication (Hung et al., 2019; Hung & Carpenter, 2017), the authors chose to use Faster Region-based Convolutional Neural Network (Faster R-CNN) (Ren et al., 2015), a state of the art method of its time, to automatically detect blood cells infected by malaria.

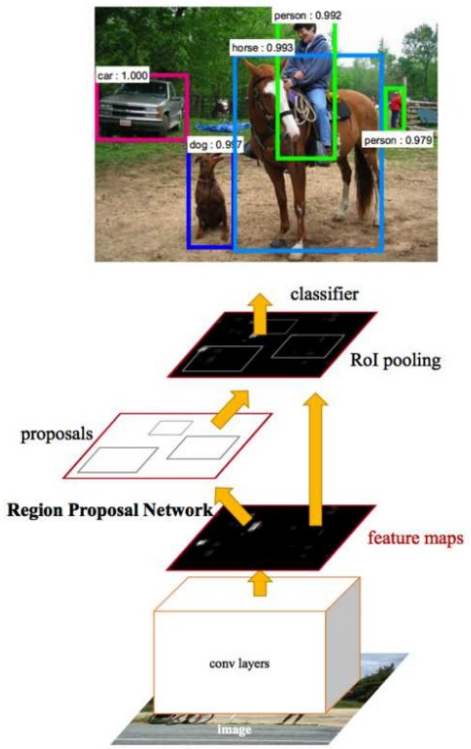

Figure 5 - Diagram of the Faster R-CNN architecture. (Hung & Carpenter, 2017)

The Faster R-CNN uses a so-called region proposal network (RPN), or a small network which proposes regions of the image that might contain an object, with a region-based CNN for classification. This is meant to speed up previously used region proposal methods, which were bottlenecks of region-based networks. The first convolutional layers are shared among the R-CNN and RPN, then a small network is slid across the image and uses fully connected layers shared for all positions of the image to produce region proposals, as shown in Figure 5. An alternating training procedure is used to enable sharing layers and efficient training of the R-CNN on fixed region proposals.

As input data, they used images of blood smears from infected patients from three different laboratories. Two sets were used for training and one left for testing. All cells were manually annotated by an expert. Images containing only normal red blood cells, the type making up 97% of the dataset, were excluded, while images with a higher proportion of rare classes were augmented to reduce the bias towards one cell type.

Since the differences between infected classes are much more subtle than the difference between infected and normal, a two-step classification was used. Faster R-CNN detected cells and labeled them as regular red blood cells (RBC) or other, and those labeled as other were passed on to another classifier of AlexNet architecture (Krizhevsky et al., 2012). When classification was performed in one step, the accuracy attained was only 59%. When features used for classification were visualized, it was clearly visible that the model distinguishes between RBCs and the rest, but cannot differentiate

between the other cell types. With the two-step classification described above, the accuracy reached 98% and different cell types clearly clustered together in the feature plot. The accuracy of the baseline method, traditional segmentation, feature extraction and random forest, was only 50%.

Comparing counts of two human experts it was found that the accuracy of the proposed deep learning model exceeds that of manual annotation, which only reached 72%. Thus automatic classification can help in cases, where phenotypes are difficult to distinguish for a human.

2.3. Divide and Conquer with SVM

Another recent method for segmenting cells in microscopy images, fastER, uses so-called extremal regions in combination with support vector machine scoring to achieve fast and accurate segmentation (Hilsenbeck et al., 2017).

Extremal regions are defined as a set of neighboring pixels which is surrounded by pixels of higher intensity value than the maximum intensity within the region. These can be obtained by including all pixels reachable from a given seed pixel without exceeding its intensity anywhere on the path. Extremal regions can be organized into a tree structure, where the child nodes of a region represent its subsets. Figure 6 provides an example of a decomposition into extremal regions and its component tree. Additionally, each region with two child nodes is split into two sub-regions, each containing one subset region and all remaining pixels are assigned to the nearer of the two. Since extremal regions describe parts of an image that are darker, or lighter after inverting the image, than the surrounding area, they can be effectively used as candidate regions for detecting cells.

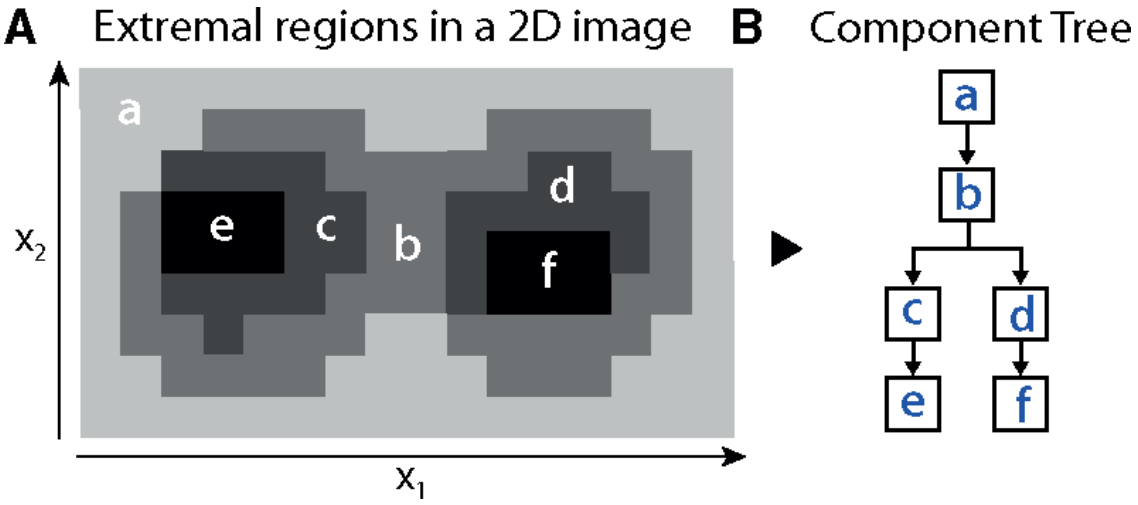

Figure 6 - Example of decomposition of an image into extremal regions (A) and the corresponding component tree (B). (Hilsenbeck et al., 2017)

After detecting all extremal regions in an image, nine features are then extracted from each and an SVM is used to produce a score, which gives the likelihood of the region corresponding to a single

cell. A divide and conquer approach is used to find a set of non-overlapping regions with high scores. The recursive formula *F(x)* selects the candidate set with the highest average score, where candidates are either the region *x* itself, its two sub-regions, if applicable, or the result of a recursive call of *F* on all regions corresponding to child nodes of node *x*. The desired segmentation is obtained by applying *F* to the whole image.

To decrease the time complexity of the segmentation algorithm, the method processes regions from leaf nodes and uses an efficient calculation of features from such statistics, which can be simply added to the parent region when progressing through the tree without computation or storage overhead. Seed pixels for the regions resulting from recursive segmentation of each given component are also stored in its node.

Samples for training the SVM were obtained automatically from manually labeled images. All extremal regions with a 20% or greater overlap with negatively labelled areas were used as negative samples. For each positive label, the best extremal region was selected based on sensitivity and specificity of its overlap with the positive area.

Two real and one synthetic dataset of microscopy images were used to evaluate the method and compare its performance to other solutions, including „ilastik“ (Sommer et al., 2011), U-net (Ronneberger et al., 2015) and CellProfiler (Carpenter et al., 2006). fastER clearly outperformed the others in terms of speed. Three different metrics were defined for accuracy: F-score using precision and recall, number of cell merge errors and Jaccard index, which quantifies the overlap between cell region and segmented region. The methods produced different results on each dataset and none of them was conclusively better than the others. However, fastER tended to have metric results close to the best performing method for each dataset.

This method has the advantage of time-efficient computation, which allows analysis of large datasets in reasonable time, while preserving accuracy of state-of-the-art methods. In addition, it does not require manual setting of parameters. Unlike many other publications mentioned in this thesis, the source code for fastER is open-source and compiled binaries are available. Another feature that sets it apart is the use of a relatively simple ML method in combination with a clever recursive algorithm.

2.4. The Weka Framework

The Waikato Environment for Knowledge Analysis, or WEKA (Hall et al., 2009), is a framework implementing various machine learning algorithms. It provides a user interface for convenient use of image preprocessing, classification, regression, clustering and attribute selection methods and visualization of results. The framework is quite dated, but some of the tools mentioned in the Tools section still utilize it, individual usages are described in more detail there.

3. Approaches to counting objects

Counting objects is quite a different problem from detecting and classifying them, even though it seems closely related. Often the approach used is detecting them first and performing image segmentation, since once detected, it must be easy to count objects, but this needs not be the case. The issue is often extreme crowding of objects and their overlap in the image, typically in photos of a crowd of people or a traffic jam. When the desired output is merely the number of objects in an image, determining their precise locations is unnecessary. Some methods take advantage of this and estimate the number without localization (Lempitsky & Zisserman, 2010).

The need to determine the number of objects is common for surveillance camera footage or counting wildlife or trees in images of landscape, but this thesis focuses on applications in cell biology. These are typically counting cells or subcellular structures, e.g. nuclei, in microscopy images, but can also include macroscopic objects, such as bacterial colonies grown on solid media.

Many solutions use neural networks, in particular deep learning (Marsden et al., 2017; Oñoro-Rubio & López-Sastre, 2016; Walach & Wolf, 2016). This is generally a method chosen for imaging data, whether the goal is to classify or to count. However, other types of approaches also exist, such as Bayesian regression (Chan & Vasconcelos, 2012) and linear regression to estimate the object density (Lempitsky & Zisserman, 2010).

3.1. Density counting

Lempitksy and Zisserman (Lempitsky & Zisserman, 2010) were among the first to translate the counting problem to estimation of the density function. This is a function of the pixel values and the count of objects in the whole image or its section can be obtained by integrating the function over the given area.

The authors used training data with dotted counts, i.e. a set of two-dimensional coordinates for each image representing the counted objects. The ground truth function was defined as a Gaussian kernel in place of each user-defined dot. For objects near the edge of the image, part of the probability mass is outside of the image boundary, so the sum across the image is not exactly equal to the object count. For each pixel, the value of F is the sum of evaluations of all Gaussian kernels at that pixel.

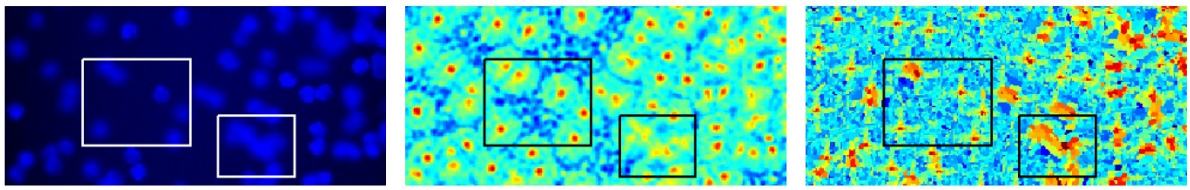

Figure 7 - Comparison of detection and density method. Left: input image with two highlighted areas. Centre: detection confidence map produced by an SVM. Right: density map. (Lempitsky & Zisserman, 2010)

The goal is to find the linear transformation, which would approximate the density function at pixel p. The parameter vector w needs to be learned, then the function at each pixel given vector w is as follows:

$$\forall p \in I_i, F(p|w) = w^T x_p^i$$

The learning process minimizes the loss function, which is a sum of the distances between ground truth and estimated density.

$$w = argmin_w(w^T w + \lambda \sum_{i=1}^{N} D\left(F_i^0(.), F_i(.|w)\right))$$

The distance metric was specifically defined here. Per-pixel distances do not reflect the nature of the counting problem, using only the total count loses valuable positional information. The MESA distance was defined to use both. It is a sum of count differences over all box subarrays of the image. D_{MESA} has three desirable properties. It is related to the counting problem, it is not sensitive to noise, since positive and negative deviations cancel each other out, and it is sensitive to the layout of dots.

The MESA distance can be calculated efficiently using the following formula:

$$D_{MESA}(F_1, F_2) = \max\left(max_{B\in\mathrm{B}} \sum_{p\in B}\left(F_1(p) - F_2(p)\right), max_{B\in\mathrm{B}} \sum_{p\in B}\left(F_2(p) - F_1(p)\right)\right)$$

Finding the inner maxima is a maximum subarray problem – there are efficient algorithms for finding them.

The learning problem is solved by quadratic programming over all possible sub-arrays of an image using a standard iterative cutting-pane procedure. Only a small subset of constraints is used in the beginning and after each iteration j, the ones that are violated the most are added. Iterations continue until compliance is achieved for all constraints within a factor of $(1+\varepsilon)$.

The performance of the framework described in this article was then compared to several baseline counting approaches on two datasets. The first dataset included artificial images of bacterial cells as seen by fluorescent microscopy. Baseline methods for this dataset were counting by regression,

counting by detection with an SVM-based detector and special software for detecting cells in fluorescent microscopy images. The approach described above outperformed all described methods. Figure 7 shows an example prediction of counting by detection and the proposed method.

The second dataset contained annotated surveillance camera images of a busy street. In comparison with counting by segmentation and counting by regression, the density learning approach proved to be as good or even better.

The training time could be long, for some datasets and parameter values even up to hours. However, once trained, the counter can be used in real time. (Lempitsky & Zisserman, 2010)

3.2. Regression Forest for Predicting Density Map

Inspired by the previous approach, another team choose to count objects in an image by estimating a density map and then integrating over it (Fiaschi et al., 2012). Here also the ground truth is defined by a Gaussian kernel around each user-defined annotation.

Instead of predicting the values directly for each pixel, this approach uses overlapping patches of the image as input and output. Several features are computed to produce v channels. A mapping

$$F: P_{in} \rightarrow P_{out}$$

is learned where P_{in} is the input patch with v channels and P_{out} is the output patch.

The method used here for predicting each patch is a regression forest. The splitting in each node is based on thresholding a function given by the value at a certain pixel position and channel. The best function and threshold pair is chosen so that the responses in each subset given by the split are as close as possible to the subset average response.

For predicting outputs for unseen data, all patches in leaves reached by the test input are averaged. These patch predictions are then averaged over a subset of patches containing a given pixel to produce the pixel prediction.

The performance of the model was tested on a dataset of fluorescence microscopy images and another dataset of surveillance camera images of pedestrians. In the first case, this method outperformed the previous state-of-the-art method (Lempitsky & Zisserman, 2010), in the second case it performed only slightly worse.

3.3. CNN for Predicting Density Map

Xie, Noble and Zisserman adopted the idea from the first mentioned approach (Lempitsky & Zisserman, 2010) to translate the problem of counting to estimation of the density function (Xie et al.,

2015, 2018). They used a fully convolutional regression network (FCRN) to estimate the density function and count cells in microscopy images. The aim was not to count by detection, rather to detect cells as a side effect of their counting. Additionally, the authors attempted to train the model solely using artificially generated data.

Same as the previous authors, they searched for a function between image I and its density map, denoted as $F: I(x) \rightarrow D(x)$. For the ground truth, superposition of Gaussian kernels for each dot annotation was used. The model was trained to predict a density map by minimizing the mean square error between the prediction and the ground truth map.

Most convolutional neural networks use convolution in combination with ReLU (rectified linear units) and max pooling, which leads to reduction of the image dimensions. Since here the desired output is a density map of dimensions equal to those of the original image, first convolution and max pooling is performed and then the size is increased back to original by upsampling.

Two architectures were proposed, FCRN-A and FCRN-B. FCRN-B, shown in Figure 8, uses fewer poolings and larger convolution kernels in comparison to FCRN-A, which preserves more information and therefore the network contains about three times as many trainable parameters. However, FCRN-A has a larger receptive field, which proved useful when cell clumps are present in the image. Only a small number of layers was used, as cells tend to be small in comparison to the whole image and do not require deep networks.

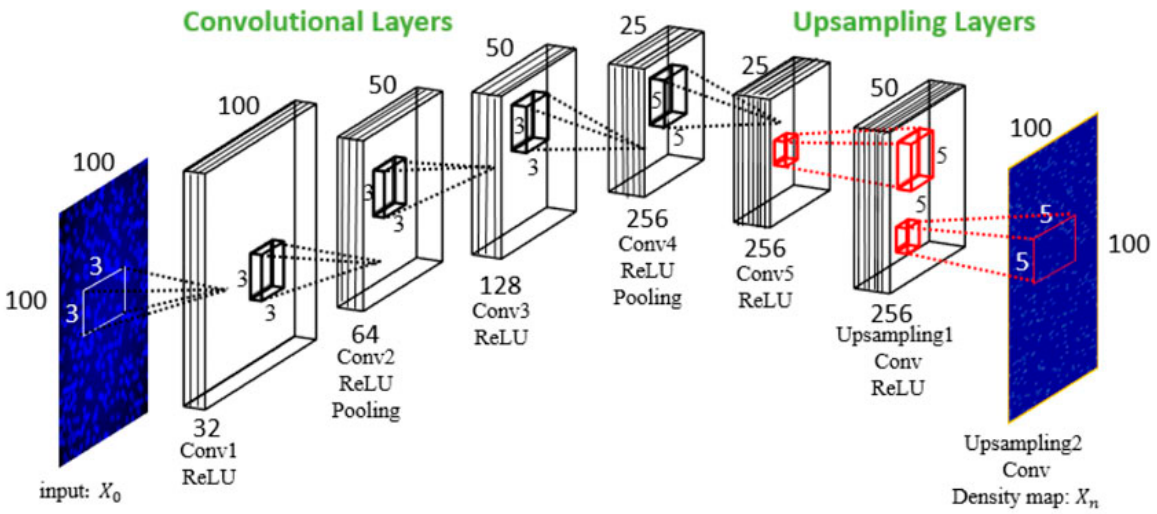

Figure 8 - FCRN-B architecture. (Xie et al., 2015)

Input frames were sampled from annotated images to increase the amount of training data and these frames were then normalized and the ground truth density maps were scaled, so that the model would fit the Gaussian peaks and not just the zero background. After pretraining on the sampled frames, the

model was trained more precisely on whole images. Mean square error was used as a loss function for stochastic gradient descent optimization.

The model was trained on synthetic data and then its performance was tested on another synthetic dataset and four datasets of real microscopy images, images of retinal pigment epithelial cells, embryonic stem cells, plasma cells and precursor T-Cell lymphoblastic lymphoma. For all these cell types, counting cells is relevant for diagnosis and/or research.

For synthetic data, the density map was estimated first and then integrated to obtain the count of cells in the image. FCRN-A performed slightly better than FCRN-B and about 9% better than the state-of-the-art method at the time of publication. According to the authors, there are three main sources of error: noise in the input data, incorrect predictions for cells near image borders and large clumps of cells.

On real data, FCRN-B produced better results and fine tuning on real images even improved them for both models. It is suggested that this might be because the cell clumps in real data are smaller than in synthetic data. Cells can be detected by obtaining local maxima in the predicted density map. In the retinal pigment epithelial cell test image, FCRN-B counted 699 cells where there were 705, and in the T-Cell test image the estimated count was 1473 out of 1502, without fine-tuning, which gives errors of 0.85% and 1,93%, respectively. (Xie et al., 2015, 2018)

3.4. Redundant Counts by Deep Learning

A recent publication (Paul Cohen et al., 2017) refers to the previously mentioned paper, but instead of predicting a density map, the authors used redundant counting across frames from the input image to create a count map. Each pixel is accounted for several times, which improves tolerance for errors. The data is not downscaled here, as in the previous case, its dimensions are kept the same by adding padding along the edges, and so a correct number of redundant counts is allowed.

The approach is to use a smaller network to process the image in a fully convolutional manner. For image I a matrix $F(I)$ is produced, which represents counts in individual frames of the size of the receptive field of the network used. $F(I)$ is compared with the target T, which is constructed the following way. Annotated cells are summed across all pixels belonging to the receptive field corresponding to the point $[x, y]$ in the target image T.

$$T[x,y] = \sum_{x',y' \in R(x,y)} L[x',y']$$

The Inception architecture (Szegedy et al., 2016) is adapted for this solution. Multiple convolutions are performed with leaky ReLU activation. The size is decreased at two points in the network by

using large filters for down sampling. There are batch normalization layers inserted after each convolution. L1 loss function, least absolute deviation, is used. The training time was significant, but the authors found that reducing the number of layers or parameters does not give as good results. Surprisingly, neither does increasing the parameter numbers, because that leads to overfitting.

The counts are redundant, meaning that each cell is counted multiple times according to the stride of convolutions. The real count is obtained by dividing the sum of all pixels by the redundant count. Cells can be accurately counted using this method, but not precisely localized. However, this has proven to be a useful approach, since increasing the stride, and therefore reducing the redundancy, lessens the accuracy of counts.

To evaluate the performance of the proposed method, the authors first tested it on the synthetic dataset used previously (Lempitsky & Zisserman, 2010). Another dataset used for testing was created by modifying eleven images of stained bone marrow. Each image was split into quarters and annotations were corrected where necessary. The third dataset contained images of human adipocytes obtained by sampling from histological slides and resampling. This dataset was expected to be the most challenging for automated counting, since adipocytes are close together and vary in size. The model is compared to several others, including the two previously mentioned, CellProfiler (Carpenter et al., 2006) and the model using regression forests (Fiaschi et al., 2012).

The method in this paper combines ideas of previous publications (Lempitsky & Zisserman, 2010; Seguí et al., 2015) also drawing inspiration regarding the use of convolutional networks (Xie et al., 2015). It was shown to outperform all previous methods on the artificial dataset and also perform exceptionally well on the other datasets.

4. Tools

Apart from experimental approaches aiming to improve performance by upgrading used algorithms, some methods have been implemented in image analysis software, both general and specialized for biological images. Here we provide several examples of such tools, focusing on free and open-source software.

4.1. Fiji

Fiji (Schindelin et al., 2012) is an open-source software solution built on ImageJ (Abramoff et al., 2004) in order to make its functionality more modern and usable for scientists. It bundles plugins together to avoid multiple installations, performs automatic updates and provides advanced libraries with detailed descriptions of underlying algorithms. The platform is targeted at biologists without programming skills, who can benefit from its easy to use tools, as well as bioinformaticians and computer scientists, who can build their own pipelines by using a scripting language or inspect the source code and contribute a plugin.

Out of the similar available products, Fiji supports the widest choice of scripting languages, enabling the user to build more complex pipelines than using the common macro language. Scripts can be both edited and interactively executed in the Fiji environment.

Fiji makes use of machine learning algorithms through a Weka plugin (Arganda-Carreras et al., 2017), which makes all the classifiers in the Weka framework (Hall et al., 2009) available. These can be used for image segmentation, boundary detection or object detection. The default classifier used is a random forest with 200 trees and two random features per node. The Weka GUI can be viewed from the plugin and the plugin is compatible with the macro scripting language.

In the Trainable Weka Segmentation, image segmentation is translated into a pixel classification problem. A user-defined set of features is extracted, including different types of edge detectors, texture filters, noise reduction filters, membrane detectors and user-defined filters. The training can either be performed interactively, by successively adding labels by the user until a satisfactory result is achieved, or by providing the model with a large set of training data. The output is a probability map of class assignments and a segmentation. (Arganda-Carreras et al., 2017)

4.2. CellProfiler

CellProfiler is an open-source software system for cell image analysis. It was one of the first products of its type and has been much updated since the first publication (Carpenter et al., 2006). Analyzing

images from high-throughput cell microscopy has become a regular part of many biological experiments and as mentioned previously, automating this task presents a great benefit not only because of the decrease in time and labor, but also for the reason that a computer may be able to detect subtle changes and quantify them better than a human eye. Some tools for image analysis existed at the time, but most required some knowledge of programming or were specifically tailored for one situation and were unsuitable for others.

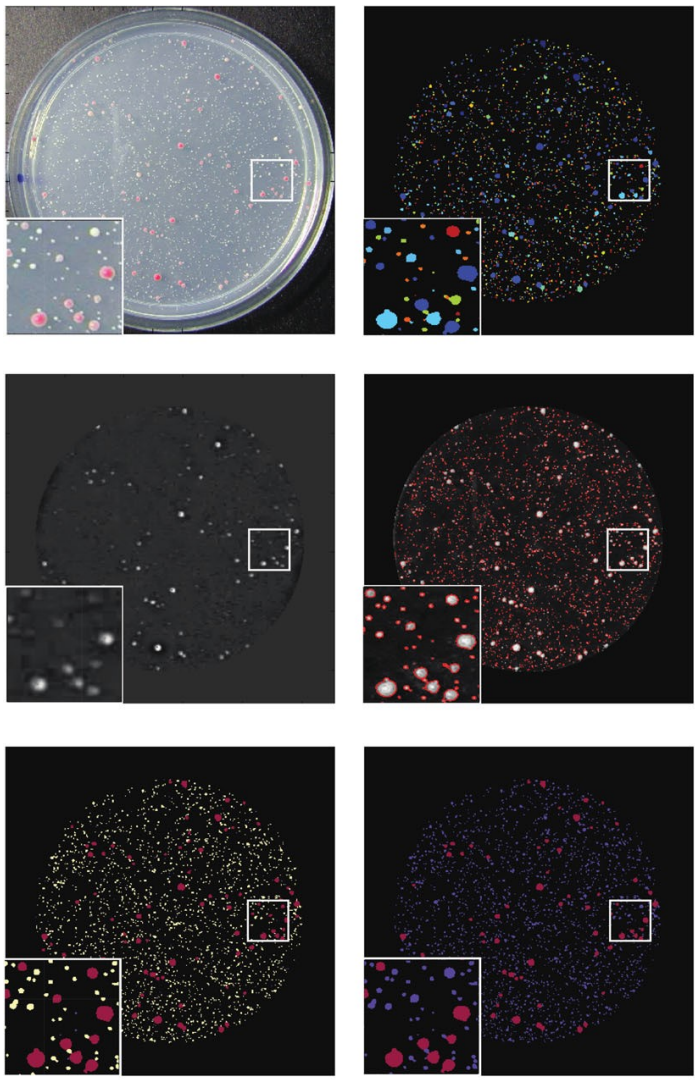

Figure 9 - Example of using CellProfiler to identify and measure yeast colonies in an image of a plate. (Bray et al., 2015)

CellProfiler was originally written in MATLAB, then was rewritten in Python since version 2.0. The analysis is modular and modules are ordered onto pipelines for specific tasks. The first part of the pipeline usually involves image processing, focusing on, but not limited to, illumination correction. The next step involves cell identification, for which various previously published algorithms were used, such as thresholding and watershed algorithm, but not machine learning in the first version. When cells are identified, various features can be measured, including shape, size, texture, or DNA content. Cell identification also serves for their counting. Apart from analyzing microscopy images,

CellProfiler can be used to identify cell colonies on an agar plate (Bray et al., 2015). An example is provided in Figure 9. Versions 1.0 and 2.0 do not yet contain any machine learning algorithms, but the authors express their wishes to implement those and to apply work by theoretical computer vision experts.

CellProfiler 3.0 contains significant updates compared to the previous two versions, such as 3D image processing, cloud computing or deep learning support. For the latter, the module ClassifyPixels-Unet, which can classify each pixel in an image into one of three classes: background, nucleus interior, or nuclear boundary, is used. Independently, other authors have created a module for nucleus segmentation. Another module, MeasureImageFocus, can be used for determining focus in an image. Each makes use of a pretrained model which the user can directly use, but the first also allows users to train the model on their own data. ClassifyPixels-Unet makes use of the U-net architecture described above (2.1) (Ronneberger et al., 2015).

4.3. Advanced Cell Classifier

The Advanced Cell Classifier (ACC) (Horvath et al., 2011) is a graphical software system for applying machine learning algorithms. While in the CellProfiler there are just a few machine learning tools among many others, the ACC is specifically focused on ML and implements the Weka framework (Witten & Frank, 2005).

The authors conducted an experiment to test different methods of phenotype classification, namely, to compare a manual method, a naïve Bayesian method and an artificial neural network (ANN). They used two datasets of 500 cell images from an RNAi screen and predefined a set of cell phenotypes for each. Images were first processed by CellProfiler (Carpenter et al., 2006) to obtain segmentation of cells. That was done by first thresholding the fluorescence signal to find nuclei and then creating a fixed-size ring around each nucleus for the predicted cytoplasm location. 26 various features were then extracted from these images, such as fluorescence intensity in the nucleus or cytoplasm, and their importance was calculated by the Info Gain method of the Weka framework. The two most important features are then plotted for each dataset and human experts used straight lines to separate the classes.

The artificial neural network used was the MultilayerPerceptron tool of the Weka framework. The authors state that default parameters were used, but it is not clear what is the exact architecture of the network. From the description of the framework (Witten & Frank, 2005) it seems that by default, there is an input layer with one node per attribute, one fully connected hidden layer and an output layer.

All three methods were trained on the two main features and used to classify cells to the predefined phenotypes. The accuracies and ROC analyses were compared, which showed that both machine learning methods are better than manual division and ANN clearly outperforms the Bayes method with an accuracy of 88.4%.

Next the authors explored the impact of the increase of features and found that using all 26 features instead of the most important two achieved an accuracy of 91.2%.

Unfortunately, the article focuses on showing the superiority of neural networks to basic methods which use few features and does not elaborate much on the architecture used and details of training.

In the next version of the software (Piccinini et al., 2017), the underlying Weka framework remains. New features highlighted by the authors include phenotype discovery tools, which present the user with cells that share the least similarity with the rest. When novel phenotypes are discovered this way, there are often not enough examples available for efficient training (He & Garcia, 2009). This problem can be overcome by using the available similar cell search tool, which helps to identify additional examples.

4.4. ilastik

"ilastik" (Berg et al., 2019; Sommer et al., 2011) is another freely available software framework for biological 2D and 3D imaging analysis, which makes use of interactive machine learning. It provides functionality for image segmentation, classification into user-defined classes, counting and tracking. External functionality, namely Fiji (Schindelin et al., 2012), can be used in the framework through a plugin. A plugin is also available for applying "ilastik" functionality e.g. in CellProfiler (Carpenter et al., 2006).

Unlike deep learning solutions which use images directly as input for the training, the authors of "ilastik" chose to use a random forest classifier as described in a previous publication (Breiman, 2001), which uses specific features for determining the splitting test in each node. It gives good results even when provided with a smaller number of training examples, such as several clicks or brush strokes by the user. This solves the problem of tedious data labelling. Feature extraction considers the spherical pixel neighborhood and can use pre-defied features or additional ones specified by the user. After classification, the contribution of individual features is calculated to identify the most important ones. An arbitrary number of classes can be defined by the user, such as background, cytoplasm or nuclei, and examples of each are selected by the user as interactive training labels. If the user is not satisfied with the image classification, they can add more labels to correct the prediction. These are used to further train the classifier and provide a more accurate result. Once a

satisfactory classification is achieved, the model can be applied to large amounts of data in batch mode.

The classifier is by default a random forest with 100 trees. However, other classifiers from the Python library scikit-learn can be used. Each pixel of the image is assigned one of the defined classes but in order to detect individual objects, connected component analysis has to be performed on the predicted map. In case the objects overlap significantly, a watershed algorithm or similar can be applied through the Fiji (Schindelin et al., 2012) plugin.

The authors stress the need of homogenous data and validation of the classifiers. Apart from keeping the imaging procedure the same, users can validate the predictions thanks to "ilastik"'s tool to interactively explore large datasets. After providing labels, the user can view the prediction on a different part of data to add labels if necessary. It is usually more helpful to correct wrong predictions than to add new labels. If the predictions keep changing even with a large number of labels, adding new ones will probably not help and a different solution needs to be used. While CNNs need large amounts of training data and the more is usually the better, this is not the case for random forests. The classifier also provides uncertainty measures and it is beneficial to add labels to areas where this value is high.

"ilastik" provides functionality for counting objects by estimating a density map and integrating over it. The method is described in the previous section (3.2). (Fiaschi et al., 2012)

It is also noted that since "ilastik" is a generic tool, its accuracy and speed may not reach the levels of specialized solutions for a given task.

4.5. Segmentation by Thresholding

OpenCFU is one of the first freely available open-source software solutions for counting bacterial colonies on an agar plate to be published (Geissmann, 2013). The basic method is simple segmentation by thresholding; however, the author uses sophisticated iterative refining for improving performance. Therefore, this is not a machine learning method. It is included here because it provides a comparable solution to the previously mentioned approaches and is implemented as a software package.

A grayscale image is produced first, then thresholding is applied to it beginning with a threshold of zero and increasing until no components of the foreground are found. For each value, the components are tested by various filters for their morphological properties and the score of accepted regions is increased. The filters test for "roundness" of the object by different means. The resulting score map reflects how often a certain region was a part of a circular object. Then another threshold is applied to these scores to obtain the final components. These are again tested by morphological filters and

overlapping objects are segmented by a watershed algorithm. Optionally, other postprocessing is applied.

This application was shown to outperform two other open-source tools available at the time, NICE (Clarke et al., 2010) and the Count_colony macro of ImageJ (Cai et al., 2011), both in speed and accuracy. It was also shown that OpenCFU was less prone to detect common artifacts as false positives.

AutoCellSeg (Khan et al., 2018) is a newer tool operating on the same principle as OpenCFU. First, the size and intensity of two example cells or colony forming units (CFU) indicated by the user are measured. These are used to calculate boundaries for detected objects, so the user does not need to specify explicit values.

The normalized input image is segmented by adaptive thresholding to produce a binary mask of objects. These are then tested for compliance with the predefined boundaries and optionally segmented by a watershed algorithm, which uses feedback-based parameter tuning for choice of seed points.

The method was tested exhaustively against other available software, including OpenCFU (Geissmann, 2013) and CellProfiler (Carpenter et al., 2006). AutoCellSeg was shown to produce better results on various data, including images of agar plates with bacterial colonies, images with inverted colors and fluorescence microscopy images. In addition, the tool allows batch processing of large datasets and manual correction of incorrectly segmented objects, which were features missing from many previously available software options.

5. Practical part

5.1. Objectives

The task of counting bacterial colonies grown on agar plates is common as a quantification method for colonization dynamics experiments, vaccination and infection challenge experiments, opsonophagocytic uptake and killing assays and other methods. In brief, opsonophagocytic assays are used to evaluate the uptake or killing capacity of effector cells on particles, such as bacteria. In such a case, the remaining number of bacteria, i.e. the viability count, can be assessed by plating the medium and after incubation, the CFU, or colony forming units, count is obtained. For the other two types of experiments, the procedure is similar – bacteria are also plated and colonies counted after incubation. The most common way of doing that is manual counting by researchers. This is not only very time-consuming and limits the number of samples that one laboratory worker is able to analyze in a day, but also somewhat subjective due to different possible interpretations of touching or merged colonies. (Dwyer & Gadjeva, 2014; Platt & Fineran, 2015)

Different automation methods exist for this task, both open-source and commercial. Some of the specialized tools, OpenCFU (Geissmann, 2013) and AutoCellSeg (Khan et al., 2018), were described earlier in the text. Also, other mentioned tools include functionality for CFU counting, such as CellProfiler.

The aim of project ColonyCount was to create a tool for automatic counting of *Bordetella pertusiis* colonies on agar plates. The application should take an image of such a plate as input and produce the number of colonies on it.

5.2. Materials

The data used for this experiment was a set of 90 photographs of plates of Bordet-Gengou agar supplemented with 15 % sheep's blood, with colonies of *B. pertusiis,* kindly provided by Mgr. Ludmila Brázdilová from the Laboratory of Molecular Biology of Bacterial Pathogens of the Czech Academy of Sciences. The basic appearance of these is a red plate with round, white spots, as seen in Figure 10. The agar is slightly lighter in rings around colonies due to hemolysis. The photos were taken with a conventional mobile phone camera. Image quality varied between samples, images often contained faults such as glare and reflections, some were somewhat blurred. The photos with very bad quality were excluded from the dataset. Also, some of the photographed plates were flawed by tears or bubbles in the agar. These, too, were excluded if the damage prevented clear visibility of colonies.

Each photograph was cropped to a square containing only the plate and rescaled to 980x980 pixels. Coordinates of colonies in each image were found manually and stored in separate files.

To simplify the counting task and to increase the number of samples, 500 98x98 pixel squares were sampled from each image, giving 45,000 frames. The labels for each square were determined as the number of colony centers belonging to given square. The average number of colonies per square was 3.006. Pixel values were scaled to values between 0 and 1. The models were trained on these segments and then predictions for each segment were summed across whole images. 40,500 image squares were used for training and the remaining 4500 for validation. 18 additional photos were labeled and sampled and predictions on these were used for statistical analysis.

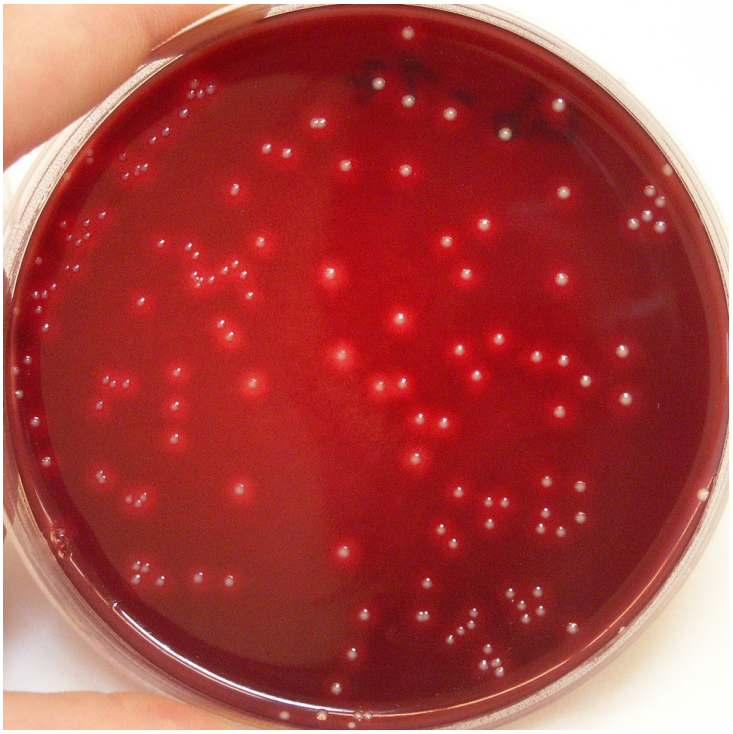

Figure 10 - Photograph of agar plate with bacterial colonies.

5.3. Methods

As is common for image processing tasks, we used a convolutional neural network. Two basic architectures were considered, a classic three-layer contracting network with max pooling in two variants and a network with three layers of decreasing size obtained by averaging.

In the first case, each of three layers consisted of a convolution with a 5x5 kernel sliding across the whole image by strides of one and max pooling with a 2x2 filter and strides of two. The third layer is without pooling. The values at each of the layers after convolution was performed were summed and stored. An optional fully connected (FC) layer was added after the last convolution, whose output would replace the third sum. The output of the network was obtained by adding the last, the last two,

or all three of the intermediate sums. The size of the input frame and the convolutions were chosen so that the filter on the last layer would cover an area of 20x20 pixels, which is approximately the size of the largest colonies. This option will be further referred to as architecture A and its diagram can be seen in Figure 11.

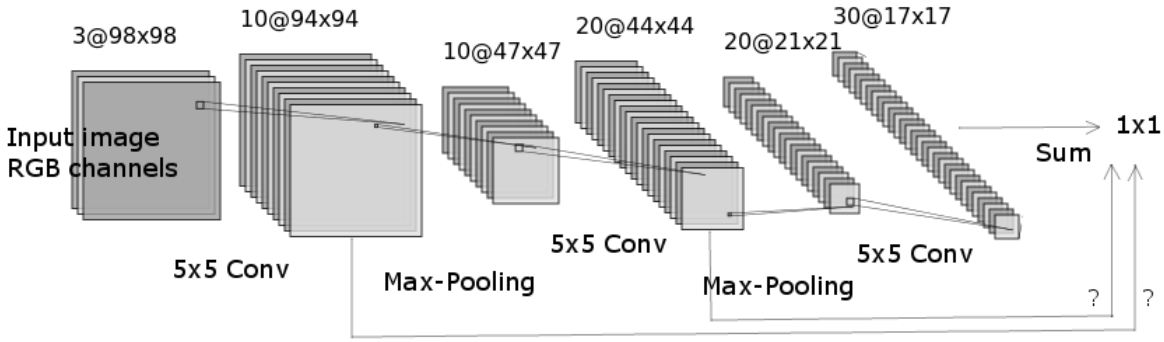

Figure 11 - Diagram of architecture A. Variant with 10, 20 and 30 filters and without fully connected layer. Question marks indicate optional inclusion of layer sums in final output.

The second architecture, called B, is similar to A. The only difference is that the output of only three filters is used for the output sum and the output of the rest is passed on to the next layer. On the last layer, outputs of all filters are used for summing. Figure 12 illustrates architecture B.

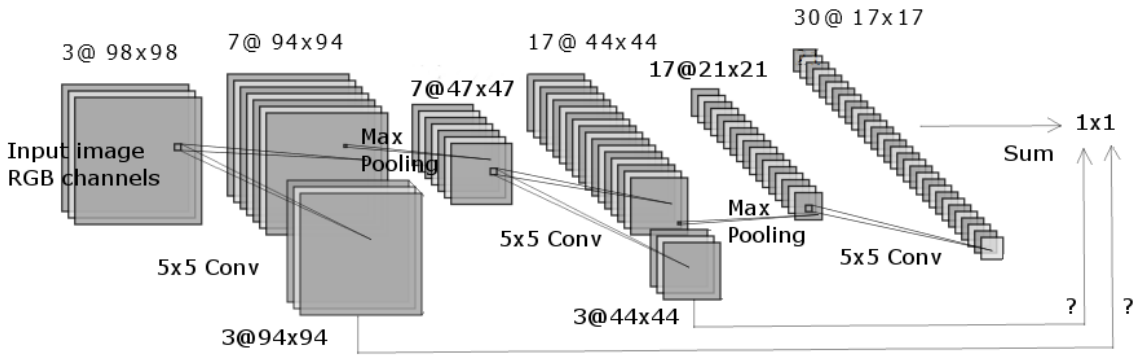

Figure 12 - Diagram of architecture B. Variant with 10, 20 and 30 filters and without fully connected layer. Question marks indicate optional inclusion of layer sums in final output.

The last variant, which will be referred to as architecture C in the following text, used convolution with max pooling only on the first layer. The output from this was contracted two times by average pooling with 2x2 filters and strides of two to get two additional layers of smaller sizes. With this pooling the image was reduced to half its size, which enabled us to use the same filters to detect objects of different sizes. The same convolution layer is applied to all three thus obtained layers, so that its weights are common for the layers and trained together. Next, the three outputs are flattened and concatenated and either summed directly, or the output results from an additional fully connected layer. This variant is shown in Figure 13.

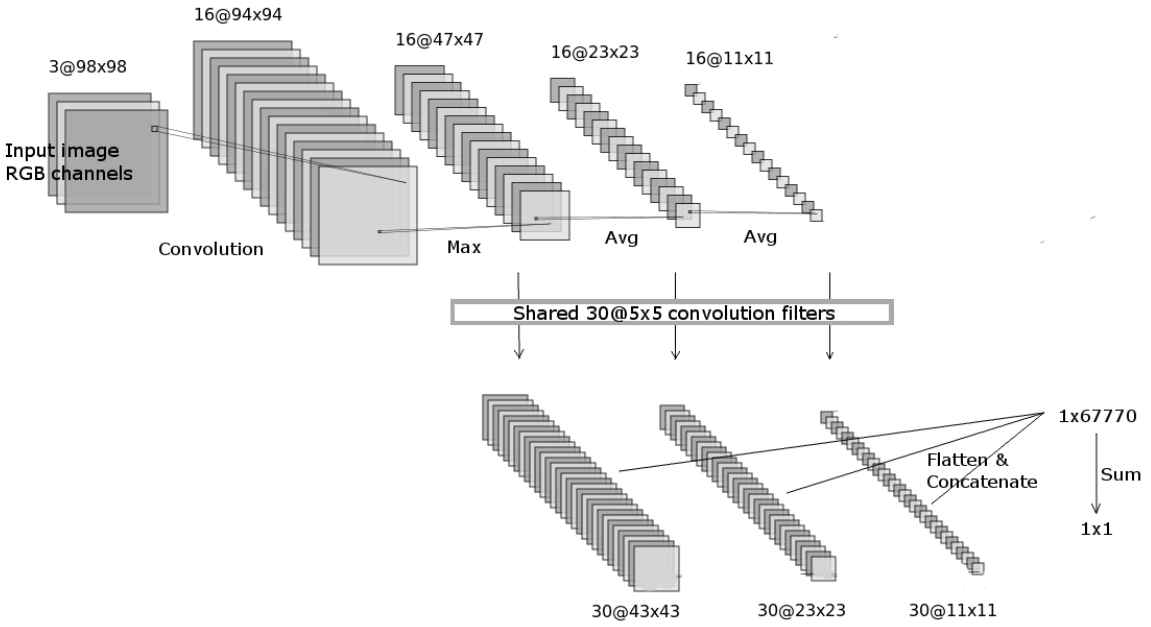

Figure 13 - Diagram of architecture C. Max is used to denote max pooling, Avg represents average pooling. Variant without fully connected layer and with filter numbers set to 16 and 30.

Two specific activation functions were defined for this project. The function listed in Table 1 as Sigmoid extended, was defined as the sigmoid function multiplied by 1.2 and shifted downwards by 0.1:

$$S_{extended}(x) = (S(x) \cdot 1.2) - 0.1$$

where $S(x)$ denotes the sigmoid function of x, i.e.

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

The resulting function retains the shape of the sigmoid but extends from -0.1 to 1.1. Unlike the sigmoid function, its output can be zero. In standard Sigmoid transformations, the output is always positive, even if small, which can create bias, since an image segment may contain zero colonies. The second function, Sigmoid shifted is obtained similarly by multiplying the sigmoid function by 1.1, shifting it downwards by 0.1 and applying ReLU to the result:

$$S_{shifted}(x) = ReLU((S(x) \cdot 1.1) - 0.1)$$

This function retains the domain of the sigmoid function, [0, 1], but since the shape was extended by multiplication, the negative part of the function is cut off by the rectifier.

These functions were used in layers which were to be summed, i.e. on the last layers of architectures A and B, the outputs of the three separate filters in B and the shared convolution in C. Where the feature map was to be passed on to further layers, ReLU was used as an activation function, or in some cases the sigmoid function.

In all three cases, the model is trained using an Adam optimizer (Kingma & Ba, 2014) with mean square error as loss function. Epoch number was set to 200, since convergence was observed in fewer epochs. Batch size was 16.

Different parameters were tested for the networks. The changed parameters were the following: learning rate, activation functions, numbers of filters on each layer, and indicators of which layer outputs to include in the final sum. Architecture variants and inclusion of the fully connected layer were also alternated.

After training models on labeled crops of images, the models were used to predict counts for whole photos by summing predictions for individual sections. For convenience of use, a graphical user interface (GUI) was created for uploading a photo and producing its predicted count. There is also an option to count colonies in the image manually and save the resulting coordinates.

5.4. Results

Tens of models were trained, 18 of which were chosen for the final evaluation. Table 1 and Table 2 list the values considered for each parameter after preliminary testing, which eliminated some other possible values. Many favorable configurations may have been overlooked, but it was impossible to train all 504 models or even try other parameter values. Table 3 lists all 18 models with their respective parameters.

Table 1 - Values of parameters for architectures A and B

Learning Rate	**Activation function on first two layers**	**Activation function on last layer**	**Numbers of filters**	**Layers included in the sum**	**Fully connected layer**
0.0001	ReLU	Sigmoid	6, 10, 16	All	Yes
0.001	Sigmoid	Sigmoid extended	10, 20, 30	Last two	No
		Sigmoid shifted	16, 30, 40	Last layer	

Table 2 - Values of parameters for architecture C

Learning rate	**Activation function on first two layers**	**Activation function on last layer and separate filters**	**Numbers of filters**	**Fully connected layer**
0.0001	ReLU	Sigmoid	6, 10	Yes
0.001	Sigmoid	Sigmoid extended	10, 20	No
		Sigmoid shifted	16, 30	

Architecture C, which applies one convolution to three feature maps decreasing in size, had generally better results than A and B, with a classical sequence of convolutions and poolings. The set of 18 best models contained 7 instances of A, with an averaged mean absolute error of 0.899 on testing image segments, 2 instances of B with mean absolute error averaged to 0.887, and 9 instances of C, with an averaged error of 0.762. The 5 best models all belong to architecture C. We compared the absolute error of couples of models with equal parameters differing in architecture on the holdout image set. The resulting p-value was 2.589×10^{-108} in the Student's T-test for architectures A and C, the p-value from the same test for architectures B and C is 4.482×10^{-154}.

It seems that the best combination was of architecture C with no fully connected layer but with a larger number of filters. All three activation functions for summed layers were represented among the three best models, although the activation function in the previous layers was always ReLU.

In architectures A and B, the best models summed only one or two layers. Using all three appears less favorable. In other trained models, coupling lower filter numbers with summing all three produced better results than with higher filter numbers (results not listed in Table 3). However, higher filter numbers are more prevalent in the final set.

Adding a fully connected layer mostly seemed to worsen the performance of models with otherwise identical parameters. Still, some of these made it into the top 18. Remarkably, the fully connected layer in architecture A had a much more disastrous effect – A models with the FC layer were among the worst tested. Several learned to predict the same number for each image segment. When the absolute error of couples of models with identical parameters, differing only in the presence of a FC layer, were compared, the resulting p-vale from Student's t-test was 2.394×10^{-10}, suggesting that it is significantly better not to include FC layers.

Interestingly, the sigmoid function as activation on the earlier layers was only advantageous in combination with a higher learning rate. Sigmoid functions have lower gradients than ReLU, which can lead to the vanishing gradient problem (Hanin, 2018), where the weight update is too small, which in turn slows down or completely stops training. Increasing the learning rate can overcome that, which is probably why we see better results with the sigmoid function in combination with a learning rate of 0.001. The higher learning rate with the usual activation function combination did not work well, and neither did the sigmoid function with a learning rate of 0.0001. Even lower learning rates were dismissed early in the experiment, since they proved to be of little use.

As for the activation function on layers to be summed, the majority of models use the sigmoid function, possibly because a higher total amount of models using it was trained. Both of the two functions defined above gave good results. Sigmoid extended worked well with all architectures, sigmoid shifted only gave good results with C.

*Table 3 - Overview of evaluated models. Asterisks mark models which had significantly higher mean absolute errors in the Student's T-test than the best-performing model, 945057. * means p-value between 0.01 and 0.0001, ** means p-value below 0.0001.*

Model	Learning rate	Layer function	Output function	Filters	Summed layers	Fully connected layer	Architecture	RMSE	Mean absolute error
945057	0.0001	relu	sigmoid	16, 30	-	No FC	C	1,178124	0,59619
354309**	0.0001	relu	sigmoid_ext	16, 30	-	No FC	C	1,134531	0,601839
77893*	0.0001	relu	sigmoid_shifted	16, 30	-	No FC	C	1,169377	0,602616
65156**	0.0001	relu	sigmoid	6, 10	-	No FC	C	1,353897	0,669362
234858	0.0001	relu	sigmoid	16, 30	-	FC	C	1,50794	0,76514
992869*	0.0001	relu	sigmoid_ext	10, 20, 30	Last two	No FC	B	1,431983	0,816647
830583**	0.0001	relu	sigmoid_shifted	16, 30	-	FC	C	1,711016	0,830504
200092	0.0001	relu	sigmoid_ext	10, 20, 30	Last one	No FC	A	1,389823	0,831983
409922**	0.0001	relu	sigmoid_ext	16, 30	-	FC	C	1,692885	0,84004
61565*	0.0001	relu	sigmoid_ext	10, 20, 30	Last one	No FC	A	1,473339	0,8632
19047**	0.0001	relu	sigmoid_ext	16, 30, 40	Last one	No FC	A	1,320175	0,864943
358466	0.001	sigmoid	sigmoid	16, 30, 40	Last one	No FC	A	1,555817	0,899607
593664**	0.001	sigmoid	sigmoid	10, 20, 30	Last one	No FC	A	1,574182	0,930533
74054**	0.001	sigmoid	sigmoid	6, 10, 16	Last one	No FC	A	1,640571	0,933155
514133*	0.0001	relu	sigmoid	16, 30, 40	Last two	No FC	B	1,667167	0,956943
82104*	0.0001	relu	sigmoid_ext	6, 10	-	FC	C	1,901413	0,963381
52218	0.0001	relu	sigmoid	16, 30, 40	Last one	No FC	A	1,698978	0,972828
497158**	0.0001	relu	sigmoid	6, 10	-	FC	C	1,883005	0,99295

5.5. Implementation

The project ColonyCount consists of two main stages: creating and training a model; and using a trained model to predict the count in an image. The source code can then be divided into four main parts: data acquisition and preprocessing, the model training script, the model loading and prediction script and a graphical user interface. Figure 14 shows the basic workflow.

The program was written in Python with use of the TensorFlow (*TensorFlow*) framework and its tf.keras submodule, which implements the Keras API (Chollet et al., 2015). Many common open-source libraries were used, such as Matplotlib, Pandas, Numpy or Tkinter.

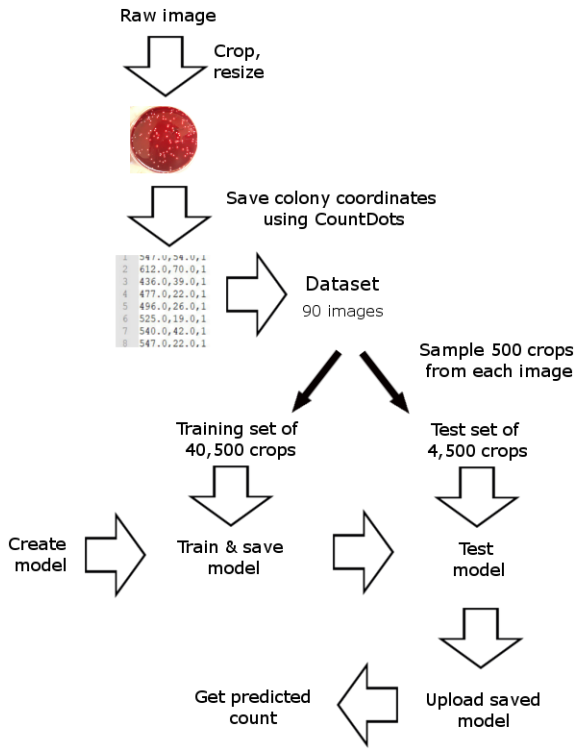

Figure 14 - Flowchart of image processing and model creation.

Training samples were labeled using a script which opens and displays an image, and records coordinates of the user's clicks, which are then saved to a file. Another module randomly samples squares of different sizes from these images and assigns each crop a label corresponding to the number of coordinates, which fall inside the given square. The crops were all saved in a separate directory, labels were listed in a csv file containing crop file names with their counts.

Two functions were written for creating a model: one for architectures A and B and one for C, both using the Keras functional API. The input layer of the model was set to accept an image section represented as a Numpy array, that is of shape (98, 98, 3). The output was a single number obtained by applying the reduce_sum function to the chosen layers. For models of all architectures, the same function was used to load training data, fit the model and log progress and training and testing results. This function also saved the trained model as .json and .h5 files.

The saved models can then be loaded by the Predictor class, which contains methods for obtaining the predicted count of a whole image and testing on an image, which also compares the predicted count with the true count, where a coordinate file is available. This class is then used in the GUI.

The graphical user interface is created using the Tkinter module and consists of one root window and optional top-level windows. The first window contains the name of the currently loaded model and

buttons for loading a model and loading an image. The former creates a new instance of the Predictor class using the chosen model, the latter opens a new window containing a canvas with the chosen image, its predicted count and, if available, the true count. A button can be used to bind click events to the canvas, which enable the user to define new colony coordinates by clicking the image. A save button becomes available, which writes the coordinates to a file and unbinds events from the canvas. This serves as a correction of an unsatisfactory prediction and also allows the use of the saved coordinates for further training. The application is terminated by closing the root window.

The source code for the application can be found at https://github.com/kvetab/Colonies.

5.6. Discussion

In order to be usable by the Laboratory of Molecular Biology of Bacterial Pathogens, the tool would have to have an error rate below 10%. We only achieved around 100%, which is much higher than acceptable. It is possible that the desired accuracy is beyond the limits of the chosen architecture, but further improvements of our method could still be explored.

Much could be tried on the input data. As with most machine learning tasks, more training data would probably improve the results, but that would mean more hand-labelled images, which are time consuming to obtain, so considering other possibilities seems preferable. One option would be to set the ground truth in a way similar to Lempitsky and Zisserman (Lempitsky & Zisserman, 2010) by letting the value around the centre of the colony to decrease smoothly instead of only counting one single point. That way, if only part of the colony was visible in an image crop, it would still influence the count.

Various pre-processing procedures could be applied to the images apart from the scaling that was used. Many segmentation algorithms work on grayscale images, which we chose not to do, since colour is an important distinguishing feature of colonies from the background, not mentioning that the task at hand is not segmentation. Other authors apply extensive feature extraction, we chose to work with the raw images as input for simplicity.

An idea for future consideration is to integrate predictions of the pretrained models over whole images and continue training on those. The difficulty here lies in the need for more labelled data, since they cannot be artificially multiplied by sampling. In order to partly solve this issue, the next round of training could only work with larger parts of images than those used here, instead of whole photos.

Another yet unexplored possibility is to approach this as a classification instead of regression and predict whether there is a colony, or its centre, at a given position. However, there we could run into those exact problems which caused the authors of previously mentioned papers (Fiaschi et al., 2012; Lempitsky & Zisserman, 2010) to give up localization of objects in images and focus on the counting itself, for example by predicting density maps.

6. Conclusion

The presented overview demonstrates that the theoretical possibilities of employing machine learning in biological imaging processing are myriad, and many more than those mentioned here may be found in the literature. Often the results of a novel method are published, but much less frequently do they find their way to the end user in the form of a software package or plugin. However important the advances of theoretical science are, it seems of little use if at least some of it does not end up being regularly used in research institutions and laboratories. Although it has to be noted that some interesting architectures, such as the U-net, have been implemented and are available as open source software; other tools often implement more basic algorithms. Biological research would benefit from more state-of-the art algorithms being implemented for practical use. In addition, the list of image processing tools here is also far from complete and commercial software was completely excluded from it.

There is a clear distinction between publications which focus on the performance of their model, comparing its accuracy and time and space efficiency with state-of-the art methods, and papers describing a product, where more attention is given to the appearance and usage of the application. The former contain a detailed description of the chosen model and its specificities but lack a usable implementation. The latter, on the other hand, are often meant for readers without a background in computer science and it can be difficult to find any details about the architecture, except for the source code.

We have seen that the field of machine learning, even its subset focusing on biology, is progressing rapidly. Even though many laborious tasks are still performed manually, we may see much more automation in the near future. Already ML methods are achieving results comparable to humans in some cases. For example, Cohen et al. (Paul Cohen et al., 2017) observed a mean absolute error of only 2.3 on one of the tested datasets. Some samples may still need to be analyzed by human experts, but the overall time spent assessing imaging data can be greatly reduced.

On the other hand, we have shown that designing and implementing a ML solution for a specific problem is not trivial. In order to be usable for the intended purpose, its average error would have to be around 10%, so approximately ten times better than what we achieved. In the future, we may work on further improving the performance of ColonyCount. Since a machine learning solution can only be as good as the input data it works with, improving the quality of training data might be a place to begin.

List of Abbreviations

ACC	Advanced Cell Classifier
CFU	Colony Forming Unit
CNN	Convolutional Neural Network
FC	Fully Connected (layer)
FCRN	Fully Convolutional Regression Network
GUI	Graphical User Interface
k-NN	k-Nearest Neighbors
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology database
R-CNN	Region-based Convolutional Neural Network
ReLU	Rectified Linear Unit
RPN	Region Proposal Network
SNP	Single Nucleotide Polymorphism
SVM	Support Vector Machine

Bibliography

Secondary sources are marked with an asterisk.

Abramoff, M. D., Magalhães, P. J., & Ram, S. J. (2004). *Image processing with ImageJ* [Article]. Biophotonics International; Laurin Publishing. http://localhost/handle/1874/204900

Arganda-Carreras, I., Kaynig, V., Rueden, C., Eliceiri, K. W., Schindelin, J., Cardona, A., & Sebastian Seung, H. (2017). Trainable Weka Segmentation: A machine learning tool for microscopy pixel classification. *Bioinformatics*, *33*(15), 2424–2426. https://doi.org/10.1093/bioinformatics/btx180

Berg, S., Kutra, D., Kroeger, T., Straehle, C. N., Kausler, B. X., Haubold, C., Schiegg, M., Ales, J., Beier, T., Rudy, M., Eren, K., Cervantes, J. I., Xu, B., Beuttenmueller, F., Wolny, A., Zhang, C., Koethe, U., Hamprecht, F. A., & Kreshuk, A. (2019). ilastik: Interactive machine learning for (bio)image analysis. *Nature Methods*, *16*(12), 1226–1232. https://doi.org/10.1038/s41592-019-0582-9

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Bray, M., Vokes, M. S., & Carpenter, A. E. (2015). Using CellProfiler for Automatic Identification and Measurement of Biological Objects in Images. *Current Protocols in Molecular Biology*, *109*(1). https://doi.org/10.1002/0471142727.mb1417s109

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Cai, Z., Chattopadhyay, N., Liu, W. J., Chan, C., Pignol, J.-P., & Reilly, R. M. (2011). Optimized digital counting colonies of clonogenic assays using ImageJ software and customized macros: Comparison with manual counting. *International Journal of Radiation Biology*, *87*(11), 1135–1146. https://doi.org/10.3109/09553002.2011.622033

Carpenter, A. E., Jones, T. R., Lamprecht, M. R., Clarke, C., Kang, I. H., Friman, O., Guertin, D. A., Chang, J. H., Lindquist, R. A., Moffat, J., Golland, P., & Sabatini, D. M. (2006). CellProfiler: Image analysis software for identifying and quantifying cell phenotypes. *Genome Biology*, *7*(10), R100. https://doi.org/10.1186/gb-2006-7-10-r100

Chan, A. B., & Vasconcelos, N. (2012). Counting People With Low-Level Features and Bayesian Regression. *IEEE Transactions on Image Processing*, *21*(4), 2160–2177. https://doi.org/10.1109/TIP.2011.2172800

Chen, Y., Li, Y., Narayan, R., Subramanian, A., & Xie, X. (2016). Gene expression inference with deep learning. *Bioinformatics*, *32*(12), 1832–1839. https://doi.org/10.1093/bioinformatics/btw074

Chollet, F., & others. (2015). *Keras*. https://keras.io

Clarke, M. L., Burton, R. L., Hill, A. N., Litorja, M., Nahm, M. H., & Hwang, J. (2010). Low-cost, high-throughput, automated counting of bacterial colonies. *Cytometry. Part A : The Journal of the*

International Society for Analytical Cytology, *77*(8), 790–797. PubMed. https://doi.org/10.1002/cyto.a.20864

Dietterich, T. (1995). Overfitting and Undercomputing in Machine Learning. *ACM Computing Surveys*, *27*(3), 326–327.

Dijkstra, K., van de Loosdrecht, J., Schomaker, L. R. B., & Wiering, M. A. (2019). CentroidNet: A Deep Neural Network for Joint Object Localization and Counting. In U. Brefeld, E. Curry, E. Daly, B. MacNamee, A. Marascu, F. Pinelli, M. Berlingerio, & N. Hurley (Eds.), *Machine Learning and Knowledge Discovery in Databases* (Vol. 11053, pp. 585–601). Springer International Publishing. https://doi.org/10.1007/978-3-030-10997-4_36

Dwyer, M., & Gadjeva, M. (2014). Opsonophagocytic Assay. In M. Gadjeva (Ed.), *The Complement System: Methods and Protocols* (pp. 373–379). Humana Press. https://doi.org/10.1007/978-1-62703-724-2_32

El Naqa, I., & Murphy, M. J. (2015). What Is Machine Learning? In I. El Naqa, R. Li, & M. J. Murphy (Eds.), *Machine Learning in Radiation Oncology: Theory and Applications* (pp. 3–11). Springer International Publishing. https://doi.org/10.1007/978-3-319-18305-3_1

Fatima, M., & Pasha, M. (2017). Survey of Machine Learning Algorithms for Disease Diagnostic. *Journal of Intelligent Learning Systems and Applications*, *09*(01), 1–16. https://doi.org/10.4236/jilsa.2017.91001

Feizi, A., Zhang, Y., Greenbaum, A., Guziak, A., Luong, M., Lok Chan, R. Y., Berg, B., Ozkan, H., Luo, W., Wu, M., Wu, Y., & Ozcan, A. (2016). Rapid, portable and cost-effective yeast cell viability and concentration analysis using lensfree on-chip microscopy and machine learning. *Lab on a Chip*, *16*(22), 4350–4358. https://doi.org/10.1039/C6LC00976J

Fiaschi, L., Koethe, U., Nair, R., & Hamprecht, F. A. (2012). Learning to count with regression forest and structured labels. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, 2685–2688.

Geissmann, Q. (2013). OpenCFU, a New Free and Open-Source Software to Count Cell Colonies and Other Circular Objects. *PLOS ONE*, *8*(2), e54072. https://doi.org/10.1371/journal.pone.0054072

Geurts, P., Irrthum, A., & Wehenkel, L. (2009). Supervised learning with decision tree-based methods in computational and systems biology. *Molecular BioSystems*, *5*(12), 1593–1605. https://doi.org/10.1039/B907946G

Gupta, P. (2017, June 5). *Cross-Validation in Machine Learning*. Medium. https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10–18. https://doi.org/10.1145/1656274.1656278

Hanin, B. (2018). Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.),

Advances in Neural Information Processing Systems 31 (pp. 582–591). Curran Associates, Inc. http://papers.nips.cc/paper/7339-which-neural-net-architectures-give-rise-to-exploding-and-vanishing-gradients.pdf

He, H., & Garcia, E. A. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, *21*(9), 1263–1284. https://doi.org/10.1109/TKDE.2008.239

Hilsenbeck, O., Schwarzfischer, M., Loeffler, D., Dimopoulos, S., Hastreiter, S., Marr, C., Theis, F. J., & Schroeder, T. (2017). fastER: A user-friendly tool for ultrafast and robust cell segmentation in large-scale microscopy. *Bioinformatics*, *33*(13), 2020–2028. https://doi.org/10.1093/bioinformatics/btx107

Horvath, P., Wild, T., Kutay, U., & Csucs, G. (2011). Machine Learning Improves the Precision and Robustness of High-Content Screens: Using Nonlinear Multiparametric Methods to Analyze Screening Results. *Journal of Biomolecular Screening*, *16*(9), 1059–1067. https://doi.org/10.1177/1087057111414878

Hung, J., & Carpenter, A. (2017). *Applying Faster R-CNN for Object Detection on Malaria Images*. 56–61. http://openaccess.thecvf.com/content_cvpr_2017_workshops/w8/html/Hung_Applying_Faster_R-CNN_CVPR_2017_paper.html

Hung, J., Ravel, D., Lopes, S. C. P., Rangel, G., Nery, O. A., Malleret, B., Nosten, F., Lacerda, M. V. G., Ferreira, M. U., Rénia, L., Duraisingh, M. T., Costa, F. T. M., Marti, M., & Carpenter, A. E. (2019). Applying Faster R-CNN for Object Detection on Malaria Images. *ArXiv:1804.09548 [Cs]*. http://arxiv.org/abs/1804.09548

Khan, A. ul M., Torelli, A., Wolf, I., & Gretz, N. (2018). AutoCellSeg: Robust automatic colony forming unit (CFU)/cell analysis using adaptive image segmentation and easy-to-use post-editing techniques. *Scientific Reports*, *8*(1), 1–10. https://doi.org/10.1038/s41598-018-24916-9

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980*.

Korani, W., Clevenger, J. P., Chu, Y., & Ozias-Akins, P. (2019). Machine Learning as an Effective Method for Identifying True Single Nucleotide Polymorphisms in Polyploid Plants. *The Plant Genome*, *12*(1). https://doi.org/10.3835/plantgenome2018.05.0023

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 25* (pp. 1097–1105). Curran Associates, Inc. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Lempitsky, V., & Zisserman, A. (2010). Learning To Count Objects in Images. *Advances in Neural Information Processing Systems*, 1324–1332.

Marsden, M., McGuinness, K., Little, S., Keogh, C. E., & O'Connor, N. E. (2017). People, Penguins and Petri Dishes: Adapting Object Counting Models To New Visual Domains And Object Types Without Forgetting. *ArXiv:1711.05586 [Cs]*. http://arxiv.org/abs/1711.05586

Min, S., Lee, B., & Yoon, S. (2017). Deep learning in bioinformatics. *Briefings in Bioinformatics*, *18*(5), 851–869. https://doi.org/10.1093/bib/bbw068

Noble, W. S. (2006). What is a support vector machine? *Nature Biotechnology*, *24*(12), 1565–1567. https://doi.org/10.1038/nbt1206-1565

Oñoro-Rubio, D., & López-Sastre, R. J. (2016). Towards Perspective-Free Object Counting with Deep Learning. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer Vision – ECCV 2016* (pp. 615–629). Springer International Publishing. https://doi.org/10.1007/978-3-319-46478-7_38

Paul Cohen, J., Boucher, G., Glastonbury, C. A., Lo, H. Z., & Bengio, Y. (2017). *Count-ception: Counting by Fully Convolutional Redundant Counting*. 18–26. http://openaccess.thecvf.com/content_ICCV_2017_workshops/w1/html/Cohen_Count-ception_Counting_by_ICCV_2017_paper.html

Piccinini, F., Balassa, T., Szkalisity, A., Molnar, C., Paavolainen, L., Kujala, K., Buzas, K., Sarazova, M., Pietiainen, V., Kutay, U., Smith, K., & Horvath, P. (2017). Advanced Cell Classifier: User-Friendly Machine-Learning-Based Software for Discovering Phenotypes in High-Content Imaging Data. *Cell Systems*, *4*(6), 651-655.e5. https://doi.org/10.1016/j.cels.2017.05.012

Platt, N., & Fineran, P. (2015). Chapter 14—Measuring the phagocytic activity of cells. In F. Platt & N. Platt (Eds.), *Methods in Cell Biology* (Vol. 126, pp. 287–304). Academic Press. https://doi.org/10.1016/bs.mcb.2014.10.025

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 28* (pp. 91–99). Curran Associates, Inc. http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi (Eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (pp. 234–241). Springer International Publishing. https://doi.org/10.1007/978-3-319-24574-4_28

Schindelin, J., Arganda-Carreras, I., Frise, E., Kaynig, V., Longair, M., Pietzsch, T., Preibisch, S., Rueden, C., Saalfeld, S., Schmid, B., Tinevez, J.-Y., White, D. J., Hartenstein, V., Eliceiri, K., Tomancak, P., & Cardona, A. (2012). Fiji: An open-source platform for biological-image analysis. *Nature Methods*, *9*(7), 676–682. https://doi.org/10.1038/nmeth.2019

Seguí, S., Pujol, O., & Vitrià, J. (2015). Learning to count with deep object features. *ArXiv:1505.08082 [Cs]*. http://arxiv.org/abs/1505.08082

Sommer, C., Straehle, C., Köthe, U., & Hamprecht, F. A. (2011). Ilastik: Interactive learning and segmentation toolkit. *2011 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, 230–233. https://doi.org/10.1109/ISBI.2011.5872394

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). *Rethinking the Inception Architecture for Computer Vision*. 2818–2826. https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Szegedy_Rethinking_the_Inception_CVPR_2016_paper.html

TensorFlow. (n.d.). TensorFlow. Retrieved May 18, 2020, from https://www.tensorflow.org/

Walach, E., & Wolf, L. (2016). Learning to Count with CNN Boosting. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer Vision – ECCV 2016* (Vol. 9906, pp. 660–676). Springer International Publishing. https://doi.org/10.1007/978-3-319-46475-6_41

Wang, S., Peng, J., Ma, J., & Xu, J. (2016). Protein Secondary Structure Prediction Using Deep Convolutional Neural Fields. *Scientific Reports*, *6*(1), 1–11. https://doi.org/10.1038/srep18962

What is max pooling in convolutional neural networks? - Quora. (n.d.). Retrieved April 30, 2020, from https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks

Witten, I. H., & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann: San Francisco. http://web.a.ebscohost.com.ezproxy.is.cuni.cz/ehost/ebookviewer/ebook/bmxlYmtfXzM1MTM0M19fQU41?sid=cd212563-f1bb-4bac-b324-2f8dcea47a97@sdc-v-sessmgr02&vid=0&format=EB&rid=1

Xie, W., Noble, J. A., & Zisserman, A. (2015). Microscopy Cell Counting with Fully Convolutional Regression Networks. *In 1st Deep Learning Workshop, Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 8.

Xie, W., Noble, J. A., & Zisserman, A. (2018). Microscopy cell counting and detection with fully convolutional regression networks. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, *6*(3), 283–292. https://doi.org/10.1080/21681163.2016.1149104