xxxx
Examensarbete 45 hp
November 2016

Convolutional neural networks for classification of transmission electron microscopy imagery

Sergii Gryshkevych

Masterprogram i tillämpad beräkningsvetenskap
Master Programme in Computational Science

**Teknisk- naturvetenskaplig fakultet
UTH-enheten**

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Convolutional neural networks for classification of transmission electron microscopy imagery

Sergii Gryshkevych

One of Vironova's electron microscopy services is to classify liposomes. This includes determining the structure of a liposome and presence of a liposomal encapsulation. A typical service analysis contains a lot of electron microscopy images, so automatic classification is of great interest. The purpose of this project is to evaluate convolutional neural networks for solving lamellarity and encapsulation classification problems. The available data sets are imbalanced so a number of techniques to overcome this problem are studied. The convolutional neural network models have reasonable performance and offer great flexibility, so they can be an alternative to the support vector machines method which is currently used to perform automatic classification tasks. The project also includes the feasibility study of convolutional neural networks from Vironova's perspective.

Handledare: Max Pihlström
Ämnesgranskare: Ida-Maria Sintorn
Examinator: Justin Pearson
XXXX

Contents

1 Introduction

Vironova is a Swedish Biotech company that supplies hardware and software solutions for advanced electron microscopy (EM), image analysis, nano-characterization and viral clearance testing [1]. Vironova's analyses contain a lot of EM images and many its tasks involve a classification step, therefore automatic classification is of great interest. Currently automatic classification is performed by means of support vector machines (SVM) [2], which demonstrates reasonable performance. On the other hand SVM operates on features extracted from the images rather than on raw pixel data. Determining a set of features that describes differences between classes is a non-trivial task that requires expert knowledge. This motivates interest in methods that operate on raw pixel data and are capable of learning image features themselves during training.

Convolutional neural networks (CNNs) [3] is one of such methods. CNN are designed to recognize visual patterns directly from pixel data with minimal preprocessing. One of the first successful applications of CNN to a large data set was recognition of handwritten digits [4] in the end of 80s'. Since then CNN has demonstrated excellent results in wide range of problems [5, 6]. For example CNN are used for segmentation [7, 8] and for classification [5, 9] tasks.

The goal of this project is to study and evaluate the suitability of using CNNs for automatic classification of electron microscopy (EM) images. CNN models are evaluated by benchmarking against SVM for selected problems. This project also encompasses prospecting the role of CNN technology for the Vironova EM services both as a method for automatic classification included in the software as well as being a research tool (e.g., for identifying useful particle features). This includes discussion on library licenses, operating system availability, performance and community support.

2 Problem description

One of Vironova's electron microscopy services is to classify different types of liposomes[1]. This includes determining:

[1]A liposome is a spherical vesicle having at least one lipid bilayer [10]. Liposomes were first described in 1964 by A.D. Bangham and R.W. Thorne and G. Weissman. They sug-

- The structure of the liposome.
- The presence of liposomal encapsulation of, for example or doxorubicin.

Let us call these two problems as *Lamellarity* and *Encapsulation.*

Lamellarity. The term lamellarity refers to the number of lamellae. Lamella, in cell biology, is used to describe numerous plate or disc-like structures at both a tissue and cellular level [12]. According to the number of lamellae liposomes can be *unilamellar* (single lamella) and *multilamellar* (multiple lamellae). In addition, an *uncertain* class is introduced because in some cases it is almost impossible to be certain about the lamellarity class. Liposomes can for example overlap each other. Another issue to keep in mind is that samples are imaged in cryo (frozen) conditions, so liposomes may be partly covered with pieces of ice that result in large black blobs in the EM images. Figure 1 illustrates examples of liposomes from each class.

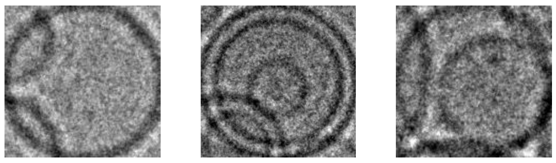


Figure 1: Lamellarity problem, 3 classes

Table 1 presents the lamellarity problem data set which contains 14 169 liposome objects. All images in this data set have been manually classified by an expert into three classes.

Table 1: Lamellarity problem

Unilamellar	12 368	87.29%
Multilamellar	1717	12,12%
Uncertain	84	0,59%

Encapsulation. Liposomes can be used as vehicles for drug delivery to various destinations in the human body [10]. A crucial part of testing this approach is to measure how many liposomes responded to drugs encapsulation and can carry them further. In the encapsulation problem liposomes are classified between the following classes: *full*, i.e. liposomes that received

gested the name "liposome" [10]. Liposomes can be classified into different types according to numerous features such as but not limited to size, number of lamellae, composition, shape, production method, etc. The classification of liposomes was first presented at a meeting of New York Academy of Science [11].

the drug substance; *empty*, i.e. liposomes that remained empty after the encapsulation attempt; *uncertain*, a class introduced with the same motivation as in the lamellarity problem. Figure 2 illustrates different classes of the encapsulation problem. Table 2 presents the encapsulation problem data set which contains 24 918 EM objects. Images in this data set are also manually classified into three classes.

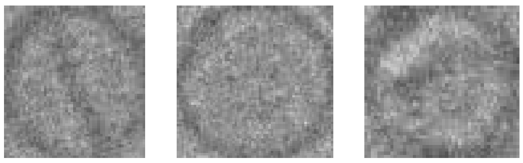


Figure 2: Encapsulation problem, 3 classes

Table 2: Encapsulation problem

Full	24 255	97.34%
Uncertain	502	2.01%
Empty	161	0.65%

3 The data

The data for this project is provided by Vironova AB. The data consists of two data sets corresponding to the lamellarity and the encapsulation problems respectively. Both data sets contain grayscale EM image cut outs of particles and a number of computed image features.

Each image depicts exactly one liposome object which is located in the center of the image. In addition, each image contains the liposome's surrounding that goes 50 pixels in each direction. The effect of the liposome surrounding on the classifier performance is described later in this report. Corresponding particle masks are also provided. Liposomes have different sizes and so do the corresponding images. The size of the liposome might be an important feature that potentially could help to describe the differences between the classes. Figure 3 shows scatter plots where the axes correspond to image width and height in pixels. Figure 4 demonstrates image size distribution inside each class.

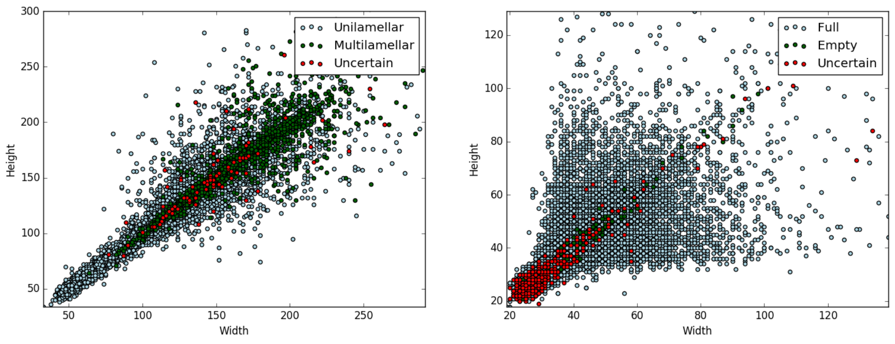


Figure 3: Scatter plots of image sizes (best viewed in color): lamellarity data set is shown to the left and encapsulation to the right. Width and height units are in pixels.

Both data sets contain the following features:

- **Circularity** is the measure how closely the shape of an object approaches that of a circle.
- **Area** expresses the extent of a particle in the plane.
- **Maximum width** in nanometres. If the particle is an ellipse, then it is a diameter along the largest axis. If the particle is a polygon, then it is the greatest distance between any two vertices.
- **Diameter** in pixels. If the particle is an ellipse, then it is a diameter along the largest axis. If the particle is a polygon, then it is the maximum value of distance transform of particle mask.
- **Length** is defined as diameter converted to nanometre units.
- **Histogram**: histogram of pixel intensity values, 32 bins.
- **Moments**: image moments μ_{20}, μ_{02}, μ_{30}, μ_{03} as defined in [13].
- **Radial Density Profile** is a 2D array, the first value in each pair is pixel intensity value and the second one is the distance in nanometres to the center of the particle. The first entry of the array corresponds to the center of the particle, then all values are averages of outward concentric rings.
- **Edge Density Profile** is defined in a similar way to the radial density profile, though the second value in each pair is defined in such way that it has 0 value at the membrane.
- **Signal to noise** is a measure that says how much the variance across the membrane is different compared to the variance along the membrane.

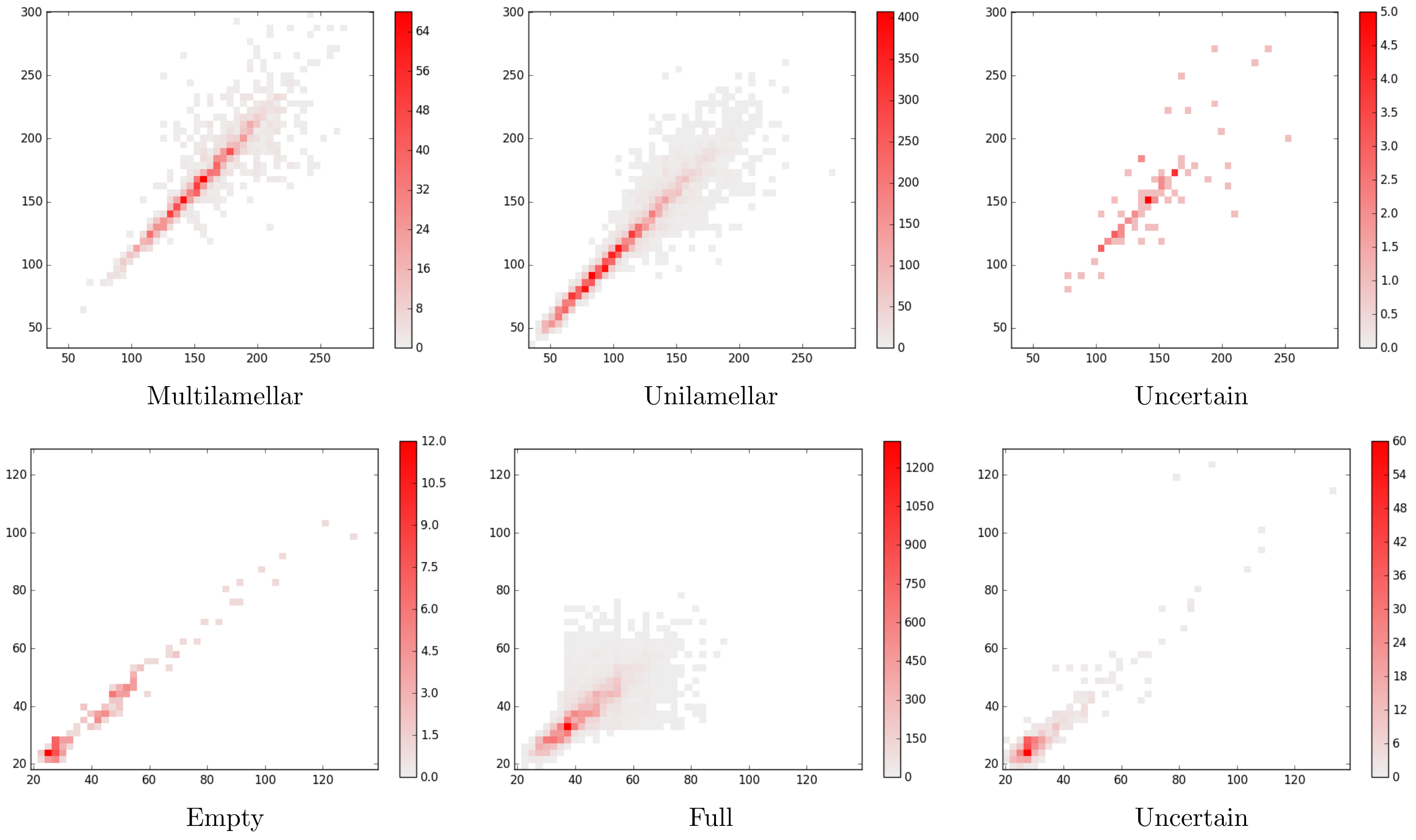


Figure 4: Distribution of image sizes in each class. The upper row corresponds to the lamellarity problem, the lower corresponds to the encapsulation problem.

4 Methodology

4.1 Support Vector Machines

The support vector machine (SVM) is one of the most influential approaches to supervised learning [2]. SVM is driven by a linear function $w^\top x + b$ [14]. In the classification tasks, SVM has the aim to determine decision boundaries that produce optimal class separation. SVM was initially designed for binary classification cases but a number of modifications were introduced for multiclass classification. One of them is the "one-against-one" approach [15]. According to this technique, $n(n-1)\frac{1}{2}$ classifiers are trained for each class, where n is the number of classes. SVM does not provide probabilities, just a class identity [14]. The `LIBSVM` [16] library is used for experiments in this project and it implements the "one-against-one" technique.

Currently automatic classification is performed by means of SVM at Vironova. The performance of the SVM classifier has shown up to 98% accuracy on the lamellarity problem and up to 87% on the encapsulation problem. The following object features are used by Vironova:

- Area
- Circularity
- Image moments μ_{20}, μ_{02}, μ_{30}, μ_{03}
- Edge density profile
- Histogram
- Internal segmentation variance

All named above features except internal segmentation variance are described in section 3. Internal segmentation variance is not available in provided data sets but its absence had insignificant impact on experiment results as shown later in this report.

4.2 Convolutional Neural Networks

Convolutional networks [3], also known as convolutional neural networks or CNNs, are defined in [14] as:

> specialized kind of neural network for processing data that has a known, grid-like topology. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Employing CNNs for this project as the main tool for performing classification tasks is motivated in the following way. Since their introduction in [3] in the late 1980's, CNNs have demonstrated excellent performance in object detection and recognition tasks. One of the most influential applications of CNNs is in the ImageNet 2012 competition that was won using a CNN model [5] with an error rate of 16.4% which was a breakthrough at that time. There are also examples of successful applications of CNNs in the microscopy image domain. For example U-Net [8] has solved a biomedical image segmentation task although provided with very limited training data. The area of CNN applications is extensive and they are being used by all leading IT companies such as Google, Facebook, Microsoft and others. The current interest for CNNs is driven mainly by three factors [17]: the availability of large annotated data sets, powerful GPU hardware, and better model regularization strategies.

CNNs are of great interest in the image analysis domain because they allow learning of image features. One does not need to be an image analysis specialist to build a powerful and robust image classification or object recognition system. Unlike many other methods in image analysis, CNNs do not require handcrafted characteristic features designed by an image analysis specialist, even though these features can be incorporated in a CNN model. For instance, the lamellarity and encapsulation problems have already been successfully solved by Vironova specialists using the SVM combined with carefully selected object features such as area, circularity, image moments and the other features discussed in section 4.1. This project studies the CNNs ability to achieve comparable result without using handcrafted features.

4.3 Architecture Overview

A CNN can be configured according to a wide variety of architectures. There is no general rule that says how to configure a CNN for a specific task but there are some best practices and general recommendations. Probably the most influential CNN architectures are VGG [18], AlexNet [5] and GoogleNet [19].

According to [14] a typical building block of a convolutional neural network consists of three stages:

1. **Convolutions**. The convolutional layer is the core building block of a CNN. Here most of the computations are done. The convolutional layer's parameters are a set of learnable filters. A filter is usually small spatially, i.e. along width and height, and extends through the full depth of the input volume. For example, if an input image to some CNN has three color channels, then the filter on the first layer will have size $n \times n \times 3$, where n is some small integer. If we decide to apply 32 such filters on the first convolutional layer the then filters on the second one will have size $n \times n \times 32$.

Convolutions can be made with filters of different sizes, different padding strategies and stride values. The stride value specifies how many pixels the filter is moved at a time. Different padding modes affect the size of the output. The most common strategies are *valid* and *same*. *Valid*, Figure 5, implies that the convolution is performed only at those positions were the filter is within the image bounds. This results in a reduced size of the layer output compared to the layer input. *Same*, Figure 6, implies zero padding in order to preserve the input size.

There are two very important assumptions about the convolutional layers: local connectivity and parameter sharing. Local connectivity implies that each neuron of the convolutional layer is not connected to all neurons in the previous layer, but only to a subset of the neurons in the previous layer. Parameter sharing implies that all neurons of the convolutional layer have the same weights. These two assumptions are illustrated in details in Figure 7.

2. **Nonlinearity**. At this stage, the linear activations produced in the previous step are run through some nonlinear activation function. One of the most widely used is ReLU which is defined as $f(x) = max(0, x)$.

3. **Pooling**. This is used to modify the output of the layer further. A pooling function replaces the output of a layer at certain positions with a summary statistic of the surrounding region. The size of this region is subject to a design decision. In practice, 2×2 pooling is often used. The most popular summary statistics is max, however others, such as average, are also possible. Pooling is usually followed by dropout for regularization purposes. Dropout is discussed in details in section 5.3. Springenberg et al. shows in [20] that max-pooling layers can successfully be replaced with convolutional layers with corresponding stride values.

A frequent source of confusion is the dimensionality of the convolutional layer output. Let us discuss it in more details. Recall that an image is usually represented as a 3 dimensional array, where the first two dimensions correspond to width and height while the third one corresponds to the number of color channels. The number of color channels is 1 for grayscale images, 3 for RGB images, etc. Let us call the third dimension depth. Note that "depth" does not correspond to the number of layers in the CNN in this context. The convolutional layer output is also three-dimensional. Width and height are controlled by parameters such as filter size, stride and padding mode. The third one, which is depth, is also controlled by a parameter, that corresponds to the number of filters we would like to use.

For example, if a convolutional network takes as input the raw grayscale 32×32 image and uses 32 3×3 filters on the first layer with unit stride and *valid* padding, then the output of the first layer would be of size $30 \times 30 \times 32$.

Technically speaking, input and output to the CNN is four-dimensional, where the first dimension is the batch size, i.e. the number of images being processed. Here it was omitted for the sake of simplicity. Note that I put depth dimension at the end, but it is really a matter of convention. Different software tools employ different conventions, so it is not uncommon to see the depth dimension on the second place, like $[b, d, w, h]$ where b is the batch size, d is depth, w is width and h is height.

Dimensionality in the depth dimension can be either decreased or increased on purpose. Sometimes it is desirable to decrease the number of depth dimensions before some computationally expensive operations. The tool that allows this is so called 1×1 convolution. For example, authors of [19] use 1×1 to reduce dimensionality in the depth dimension before the expensive 3×3 and 5×5 convolutions. Note that 1×1 convolutions affect only the depth, they leave spatial dimensions (width and height) unaffected.

Usually a number of convolutional layers is combined with a number of fully connected layers at the end. In such a case, the output of the last convolutional layer is flattened, i.e. collapsed into one dimension, and a number of fully connected layers goes to the final layer. The final layer has softmax activation and a number of nodes that is equal to the number of classes. Such an approach is simple and is efficiently implemented in many software tools but it has one important constraint. It requires input of a fixed size. Such networks can not handle images of different sizes. If it is required to build a model that can process images of arbitrary size then a fully convolutional neural network (FCNN) is a possible solution. A FCNN does not require the input to have fixed size and can be trained on images of variable size. Fully convolutional in this context means that the network does not have any fully connected layers, only convolutional. FCNNs have proven to be especially effective in semantic segmentation tasks [7]. However, FCNNs have not been used in this project.

Figure 8 summarizes all mentioned above statements and gives an example of the CNN workflow.

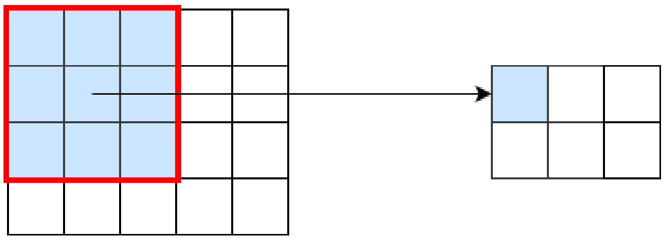


Figure 5: Illustration of the *valid* padding mode. Assume that a $3 \times 3 \times 1$ (a red frame) convolutional filter is applied to a $5 \times 5 \times 1$ input volume (to the right). Result of the convolution is to the left. The *valid* mode implies that convolution is performed only at those positions when it fully fits into the input volume, these results in reduced size of the output compared to the input.

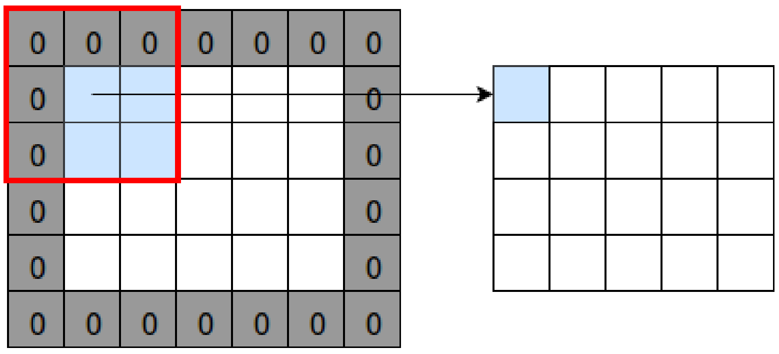


Figure 6: Illustration of the *same* padding mode. Assume that a $3 \times 3 \times 1$ (a red frame) convolutional filter is applied to a $5 \times 5 \times 1$ input volume (to the right). Result of the convolution is to the left. The *same* padding mode implies adding zero padding to the input of the size which is enough to make the output of the same size as the input, which is $\frac{n-1}{2}$, where n is the size of the filter.

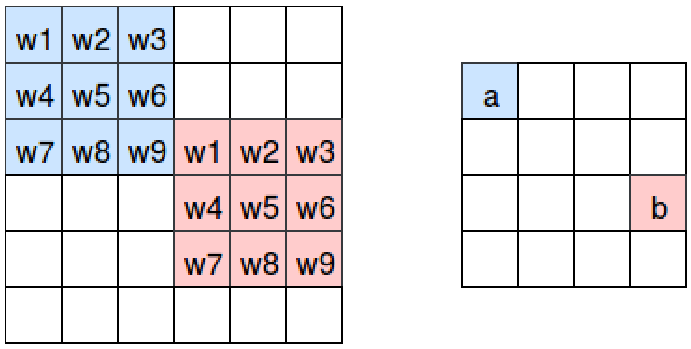


Figure 7: Illustration of local connectivity and parameter sharing (best viewed in color). The matrix on the left is input to the convolutional layer on the right. Here we assume *valid* padding mode and filter size of 3. Each neuron of the convolutional layer is connected only to a small subregion of the input. The size of the subregion corresponds to the size of the convolutional filter. For example, neuron a, highlighted in blue, is connected to 9 inputs in the upper-left corner, neuron b is also connected to only 9 inputs out of 36. Moreover, all neurons in the convolutional layer share their weights, so neurons a and b have the same weights w_i, $i \in [1, 9]$. These two assumptions reduce the number of parameters drastically. If all neurons in this example where fully connected this would result in $16 * 36 = 576$ parameters. But with local connectivity and parameter sharing the number of unique weights is only 9. Note that weights are shared only across spatial dimensions of the filter, they are still unique across the depth dimension. If the filter size was $3 \times 3 \times 3$, there would be 27 unique parameters.

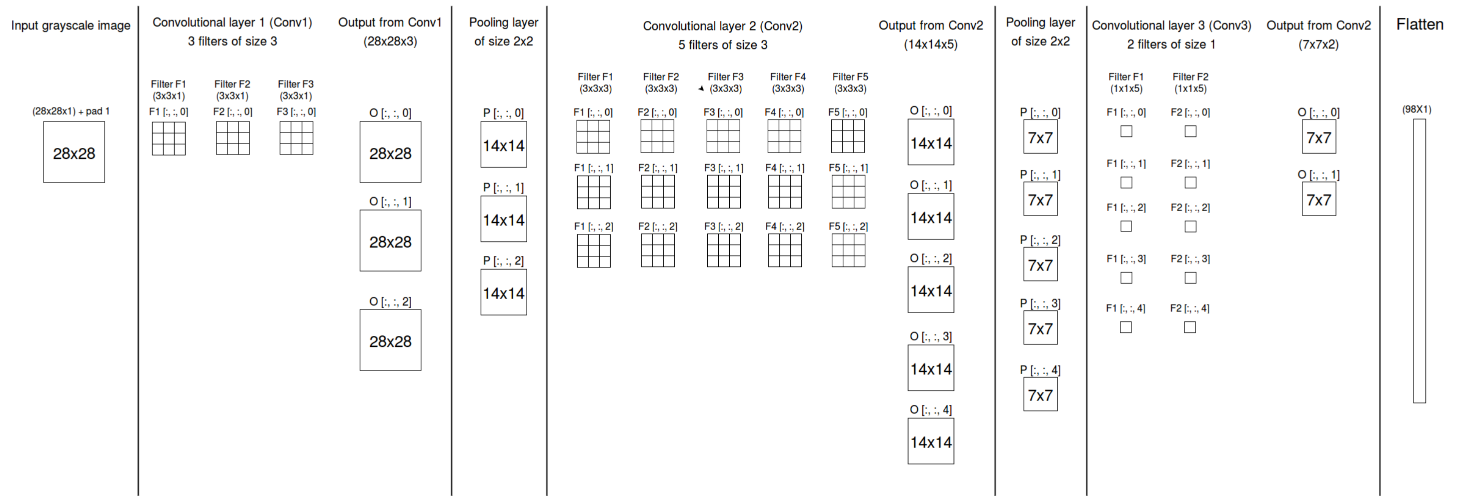


Figure 8: An example of the CNN workflow. Here we assume grayscale 28×28 input image, *same* padding mode and unit stride for all convolutional layers. The first convolutional layer consists of three $3 \times 3 \times 1$ filters, so its output is of size $28 \times 28 \times 3$. On the next layer max-pooling operation of size 2 is applied. This affects spatial dimensions, but not the depth. This layer yields output of size $14 \times 14 \times 3$. The third layer is convolutional and here 5 filters of size $3 \times 3 \times 3$ are applied. The output has size $14 \times 14 \times 5$. The input image is convolved with each filter and summed across all depth dimensions. For example, output $O[:, :, 0]$ is the sum of convolving inputs $P[:, :, i]$ with filters $F_1[:, :, i]$, where $i \in [0, 1, 2]$. The fourth layer is max-pooling of size 2 so its input is $7 \times 7 \times 5$. Lets us demonstrate depth dimension reduction with the final convolutional layer which consists of two $1 \times 1 \times 5$ filters. This layer yields $7 \times 7 \times 2$ input. Note that spatial size is unaffected while number of depth dimensions is reduced from five to two. The last layer on this illustration is flattening operation which collapses $7 \times 7 \times 2$ volume into a vector of length 98. After flattening a number of fully connected layers can be applied. Note that activation layers have been omitted on this illustration. Usually, some non-linearity, like for example ReLU, is applied to the output of the convolutional and fully-connected layers.

4.4 Deconvolutional Neural Networks

Convolutional neural networks is a powerful machine learning method that is capable of learning the image features in order to describe provided categories. However, it is usually not obvious what kind of features the model has learned. For some time convolutional neural networks have been treated as black boxes. This changed in 2014 when Zeiler et al. [17] proposed *Deconvolutional Networks (DN)* for visualization of convolutional neural networks, or rather patterns that fire network activations. In this way it is much easier to understand how convolutional neural networks "see" images.

The DN method can be summarized in the following way. The deconvolutional network goes down from activations of a given layer back to the image. In this manner DN reverses data flow of a CNN undoing the effect of convolutions. The resulting reconstructed image shows parts of the input image that caused the strongest activations.

In order to reverse the data flow of a CNN, all its components must be reversible. To invert learned filters, DN transposes them which means in practice flipping each filter vertically and horizontally. CNN usually use rectifiers as activation functions that guarantee that feature maps are always positive. Features are reconstructed using the same ReLU non-linearities. Pooling is a common building block of any CNN. Unlike convolution and rectification, pooling generally can not be undone, it is not reversible. Zeiler et al. proposed to record the locations of the maxima within each pooling region, they call these recorded values switch variables. The unpooling operation uses switches to place reconstructions into appropriate locations, setting other values within the upsampling region to zero.

Convolutional networks that replace the max pooling operation with convolution with stride can be visualized in a similar way. Springenberg et al. [20] propose modifications to the DN method that achieve similar results as in [17].

4.5 Optimization

Deep learning algorithms involve optimization in many contexts [14]. The most obvious one is finding network parameters that reduce a cost function. One design decision that has to be made is the choice of optimization method. In this project I limited the choice of optimizers to ADAM [21] (Adaptive Moment Estimation) and Stochastic Gradient Descent (SGD) [22].

Stochastic gradient descent is a stochastic approximation of the gradient descent optimization method for minimizing an objective function. SGD combined with the backpropagation algorithm is the most common method for training artificial neural networks [23].

ADAM is a novel optimization method which brings some important modifications to the SGD algorithm. ADAM's update rule uses both the average of past stored gradients and the average of past squared gradients. Both average values are exponentially decaying.

4.6 Performance measures

When evaluating a classification model, one is almost always interested in how many predictions from all predictions made that are correct. In other words we are interested in how accurate a particular model is. The accuracy of a classification system is the share of correct predictions of the total number of examples. Accuracy is one of the most frequently used performance measures in classification problems. However, accuracy alone is usually not enough to describe the predictive power of the model. Models with a given accuracy may have greater predictive power than models with higher accuracy. This is known as the Accuracy Paradox [24].

Let's illustrate the Accuracy Paradox using the encapsulation problem. Table 2 on page 4 presents the distribution of the three classes in the encapsulation data set. Consider a model that classifies all particles as full. Such a model would achieve the astonishing accuracy of 97% just by making the same prediction for all particles. Let's compare it to another model, that makes a random guess, i.e. predicts each class with $\frac{1}{3}$ probability. Such a random guess model would have an accuracy of approximately 33%, almost three times less than the first model. However, while the first model totally ignores two minority classes, the second one has 33% chance of detecting them, so it may be preferred comparing to the first one.

Imbalanced data sets is a typical reason to when accuracy is misleading and fails as a performance measure. A unambiguous way to present the prediction results of a classification model is a confusion matrix [25]. A confusion matrix is a table with the number of rows and columns both equal to the number of classes in the data set under consideration. Columns represent predicted classes and rows represent true classes. Each cell contains the number of predictions that corresponds to that particular category.

A table of confusion is a special case of confusion matrix that deals with binary classification. A table of confusion is presented in Table 3 and it is usually used to illustrate performance measures that are derived from a confusion matrix.

Table 3: Table of confusion

	Predicted True	Predicted False
Actual True	True positive TP	False negative FN
Actual False	False positive FP	True negative TN

Different performance measures are discussed and compared in [26]. The rest of this section presents performance measures that are used in this project for evaluation and comparison of different classification models.

True positive rate (TPR), also known as sensitivity or recall, measures the share of positives that have been correctly identified:

$$TPR = \frac{TP}{TP + FN}$$

True negative rate (TNR), also known as specificity, measures the share of negatives that have been correctly identified:

$$TNR = \frac{TN}{TN + FP}$$

Positive predicted value (PPV), also known as precision, measures the share of correct positive predictions:

$$PPV = \frac{TP}{TP + FP}$$

Negative predicted value (NPV) measures the share of correct negative predictions:

$$NPV = \frac{TN}{TN + FN}$$

The F_1 score is a harmonic mean of TPR and PPV, which is a measure of a test's accuracy. It reaches its best value at 1 and worst at 0.

$$F_1 = \frac{2}{\frac{1}{TPR} + \frac{1}{PPV}}$$

F_1 has been criticized for not taking true negatives into account and a number of alternative measures such as Phi coefficient, Matthews correlation coefficient and Cohen's kappa have been proposed [26]. However, let us use F_1 anyway for the sake of simplicity.

Using all measures presented above allows more detailed analysis of the classifier performance.

4.7 Loss function

This section is taken from [27].

The multi-class logarithmic loss function is a loss function that represents the price paid for inaccuracy of predictions in classification problems [28]. The formula is:

$$loss = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{i,j} log(p_{i,j}) \tag{1}$$

Where

- N – number of observations
- M – number of classes, it is three in our case
- *log* – natural logarithm
- $y_{i,j}$ – is 1 if observation i is in class j and 0 otherwise
- $p_{i,j}$ – is the predicted probability that observation i is in class j

In order to calculate multi-class logarithmic loss, the classifier must assign a probability to each class rather than simply yielding the most likely class. This fact constitutes the main difference between accuracy and logarithmic loss. In order to consider a prediction as accurate, it is sufficient that the classifier marks the correct class as the most likely one, no matter how confident the prediction is.

Figure 9 shows log loss from a single class where predicted probability ranges from 0 (the completely wrong prediction) to 1 (the correct prediction). As predicted probability moves closer to 1, log loss decreases gently to 0. On the contrary, log loss increases rapidly as predicted probability moves towards zero. It is clear from Figure 9 that the multi-class logarithmic loss heavily penalizes classifiers that are confident about an incorrect classification.

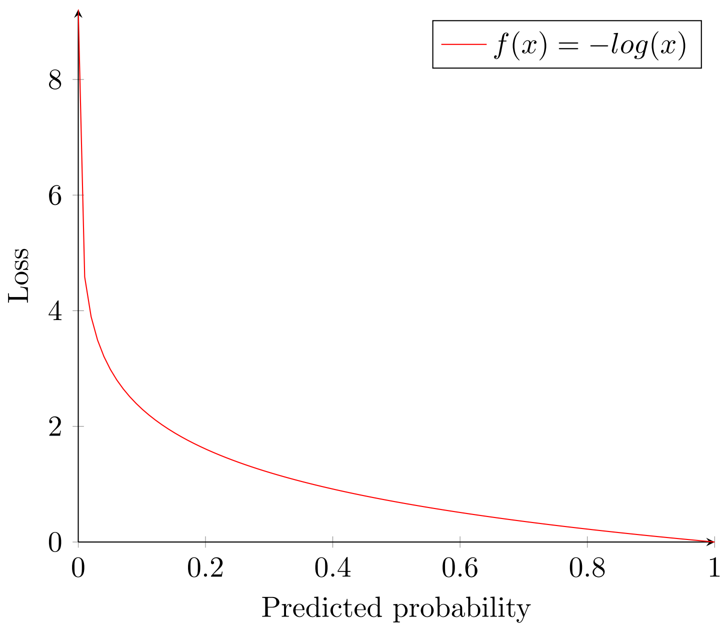


Figure 9: Log loss of a single class where predicted probability ranges from 0 to 1.

Consider an example of a binary classifier and let us take a look at the effect of various predictions for class membership probability.

- Classification that assigns equal probabilities to both classes results in loss $-log(0.5) = 0.6932$.
- Classification confident in the correct class results in loss $-log(0.9) = 0.1054$.
- Classification confident in the wrong class results in loss $-log(0.1) = 2.3026$.

In other words, the log loss function encourages moderate predictions. It is better to make all classifications neutral rather than make 50% correct classifications and 50% completely wrong predictions.

4.8 Class Imbalance Problem

The encyclopedia of Machine Learning [29] defines the Class Imbalance problem as follows: "Data are said to suffer the *Class Imbalance Problem* when the class distributions are highly imbalanced. In this context, many classification learning algorithms have low predictive accuracy for the infrequent class."

It is a serious problem because most machine learning algorithms will tend to classify all examples as the majority class and to treat examples of minority class as noise. The lamellarity and encapsulation data sets presented in section 2 are highly imbalanced. In the lamellarity data set, the majority class accounts for 87% of all samples and in the encapsulation data set, the majority class constitutes 97% of all samples. Preliminary experiments showed that classifiers trained on raw unbalanced data sets are highly biased towards the majority classes and do not generalize well. However the class imbalance problem is not uncommon. In fact, data sets are highly imbalanced in many domains, for example fraud detection, medical diagnosis, anomaly detection, telecommunications, etc.

Various solutions have been proposed to deal with this problem and they can be summarized into three groups [30]:

- *Data sampling* implies any means to produce a more or less balanced data set. It can for example be oversampling, undersampling, SMOTE (Synthetic Minority Over-sampling Technique) [31] or generating artificial examples [32]. The data sampling method usually performs best combined with data augmentation techniques.
- *Algorithm modification* is oriented towards the adaptation of base learning methods to be more attuned to class imbalance issues [33].
- *Cost sensitive learning* introduces higher penalties for misclassification of minority classes making learning algorithm more sensitive to underrepresented classes.

The methods listed above prevent classifiers from ignoring underrepresented classes during the training phase. However this is very likely to lead to another problem — overfitting. Overfitting is discussed in section 5. The rest of this section discusses methods that have been used to mitigate the Class Imbalance Problem in the lamellarity and encapsulation data sets.

4.8.1 Oversampling

Oversampling means repeating instances of underrepresented classes. Minority classes in the lamellarity data set are the multilamellar and uncertain, and in the encapsulation data set — empty and uncertain. Examples of these classes are repeated to match the number of majority class samples so that training sets become balanced. Oversampling itself does not mitigate the imbalance problem much, it works best combined with data augmentation which is discussed in section 5.5.

4.8.2 Undersampling

Undersampling, as opposed to oversampling, means excluding some samples of the majority class in order to make the training set balanced. In practice, undersampling is repeated at the beginning of each learning epoch when randomly selected samples are excluded. In such a way the learning algorithm can still use all samples for training, it just does not see all of them during all epochs. The proportion of excluded samples can vary and depends on the context. In the experiments presented in this project, undersampling implies dropping 20% of the majority class samples.

4.8.3 SMOTE

Synthetic Minority Over-sampling Technique (SMOTE) is an over-sampling approach in which the minority class is over-sampled by creating "synthetic" examples rather than by over-sampling with replacement [31]. Synthetic examples are generated by combining each minority class example with its randomly selected k nearest neighbors in feature space, where k is a parameter that depends on amount of required over-sampling. Selection of feature is a design decision and varies between different problems.

4.8.4 Artificial data

Machine learning algorithms perform best when provided with large training data sets. In many domains, including medicine and biology, expert annotation is required for preparing training and test data sets. This results in high cost and effort, so the size of data sets is often limited. In such a case artificial data is an attractive alternative to real expert-annotated data. Training of convolutional neural network models for fluorescent spot detection on artificial data is e.g., discussed and evaluated in [32].

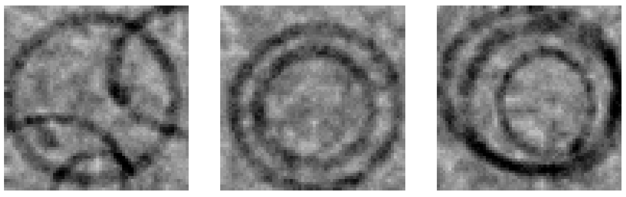


Figure 10: Artificial examples for lamellarity problem

In this project artificial data has been generated for the lamellarity problem. Artificial images of liposomes are generated by drawing circles with carefully selected randomized distortions. Background and distortions are generated using Perlin noise [34]. Perlin noise was selected because of visual similarity of its output to the actual EM images. Figure 10 presents examples of artificially generated EM images. These artificial examples contain random shift transformation so they are not centered as the real examples in Figure 1.

Note that despite visual similarity, artificial images are not based on statistical measures from the real data set. Histograms and image patterns are not drawn from real distributions. With limited time and effort, this experiment should be rather seen as a proof of concept.

5 Regularization

Let us assume that an imbalanced data set has been balanced by a combination of the methods described in section 4.8. This would significantly reduce the training error which means that the model is able to describe the data generation process of the training set.

However as the result of seeing multiple copies of the minority classes, the learning algorithm will almost certainly describe random error or noise as well. Such a model would not be able to make reliable predictions on general untrained data, i. e. its generalization error would be unacceptably high. As defined in [14], "*Regularization* is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error".

The rest of this section presents regularization techniques that have been used to reduce generalization error of models trained on the lamellarity and encapsulation data sets.

5.1 Weight decay

Weight decay is a method of adding a penalty to the weight magnitudes of the hidden layer to the loss function. In this way we express preference of smaller weights and prevent weight of some hidden nodes or convolutional filters from becoming too large. Large weights can result in undesired numerical errors such as overflow or truncation error. Usually the L^2 norm of penalized layers multiplied by some constant α (typically close to zero) is added to the loss function. However other norms such, as but not limited to, L^1 can be used. On the other hand, weight decay can lead to other problems as described in [14]. Penalties on weights can cause non-convex optimization procedures to get stuck in local minima corresponding to small values of weights.

Applied to neural networks it means that weight decay can lead to "dead units". These units do not contribute much to the learning process because their input and output weights are all very small. This configuration can be locally optimal even if loss value can be significantly reduced by making the weights larger. This phenomenon was observed during the experiments presented in this project. Applying weight decay to convolutional layers resulted in numerous dead filters which worsened the classifier's performance on both training and test sets.

5.2 Noise injection

Noise robustness can be achieved by adding some noise to the model. Noise can be added at several places.

Firstly, noise can be added to the input data, in our case to EM images. Secondly, noise can be added to weights. This technique is primarily used in recurrent neural networks [35].

Thirdly, most data sets have some mistakes in the annotated labels. As shown in section 4.7, it can be very costly to maximize $\log p(y|x)$ when label y is wrong. A label smoothing technique is proposed in [14]. Label smoothing regularizes a model based on a softmax with k output values by replacing the hard 0 and 1 classification targets with targets of $\frac{\epsilon}{k}$ and $1 - \frac{k-1}{k}\epsilon$, respectively, where ϵ is an arbitrary value between 0 and 1, usually very close to 0. The label smoothing operation can be written as:

$$x \leftarrow x\,(1-\epsilon) + (1-x)\,\frac{\epsilon}{k-1} \tag{2}$$

Label smoothing is used in all experiments presented in this project.

5.3 Dropout

Dropout [36] suggests to randomly drop non–output units (nodes) along with their connections from the network. This prevents units from co–adapting too much. Dropout is implemented as a parameter between 0 and 1 that describes the probability of dropping each unit. It can be constant during all training epochs or some decay scheme may be used. Dropout may be more aggressive in the beginning of the training and smaller later when the model becomes trained. It is the same idea as with learning rate decay. Dropout is used in all experiments presented in this project.

5.4 Early stopping

Early stopping is probably the most commonly used regularization strategy in deep learning [14]. It is a simple yet powerful technique of determining the optimal number of training epochs. The idea is to split the training data into a training set used to train the model and a validation set, which is not used in the training process. Errors of both sets are monitored at each epoch. Training should be stopped when the validation set performance has not improved for some selected number of epochs. This prevents overfitting and improves generalization. The validation set error is more suitable than the training set error because training error decreases steadily over time as the model starts to overfit while validation error begins to rise again. In this project early stopping is used to determine the number of training epochs, that is later fixed during k-fold cross validation.

5.5 Data augmentation

As was already mentioned in section 4.8, training data is often limited and additional data is hard to obtain. One way to get around this problem is to introduce additional diversity in the training set by augmenting existing data. Data augmentation has proven to be especially effective for classification tasks [14] which is exactly the case in this project. A good classifier is invariant to a wide variety of transformations and can see the true class from "different points of view". New (x, y) pairs, where x is a class example and y is corresponding label, can be easily generated by applying different transformations to x. The choice of transformations is highly application dependent. For example, optical recognition models must recognize the difference between 'b' and 'd' and between '6' and '9', so horizontal and vertical flips are not suitable for this task.

In this project, data is augmented in the following way:

- Rotation in the range $[-180, 180]$ degrees with spline interpolation
- Shear transformation in the range $[0, 0.2]$
- Vertical shift in the range $[-10, 10]$ percent of total height
- Horizontal shift in the range $[-10, 10]$ percent of total width

- Zoom in the range $[0.8, 1.0]$ which means zoom by a maximum 20%
- Horizontal flip
- Vertical flip

All transformations are applied randomly. Horizontal and vertical flips are performed with 50% probability, rotation, shear, shift and zoom values are uniform random integer values in specified ranges. Data augmentation is performed in all experiments unless the opposite is stated explicitly.

Figure 11 show an example of data augmentation. Thus, a model never sees two identical examples during training.

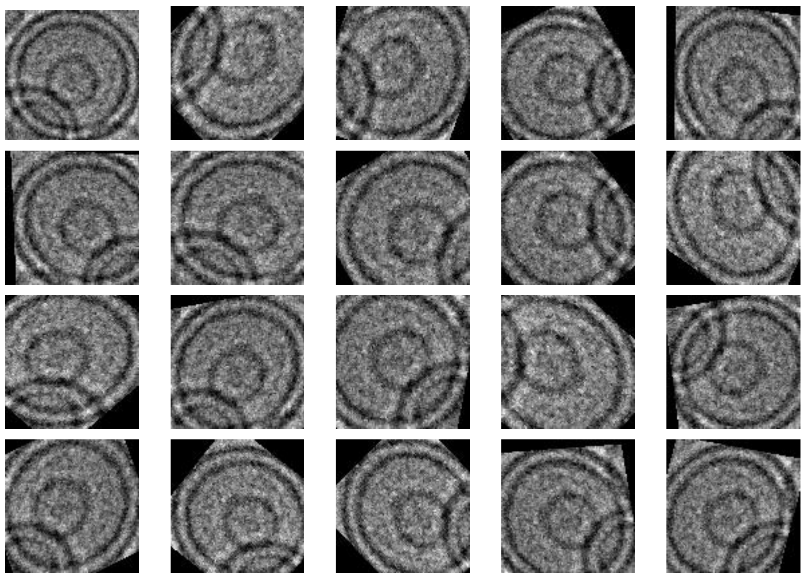


Figure 11: Examples of data augmentation. The upper left is an original image and all other images are the results of random transformations described in this section.

6 Review of deep learning software tools

A wide variety of deep learning software tools is publicly available. Most of them are open-source and are distributed under licenses that allow commercial use. The development of deep learning frameworks is driven by Universities and leading IT companies such as Google and Microsoft with extensive help from the user community. For example Theano [37] at the University of Montreal is one of the first and most influential deep learning libraries. It was initially released in 2010. Another state of the art tool is Caffe [38] by UC Berkeley. Google released the TensorFlow [39] software library in November 2015. Microsoft made its CNTK [40] tool publicly available in January 2016.

The above mentioned software tools together with some other are summarized in Table 4.

Four deep learning libraries: TensorFlow, Caffe, Torch [41] and CNTK have been compared and benchmarked in [42]. Authors have concluded that "all tested tools can make good use of GPUs to achieve significant speedup over their CPU counterparts. However, there is no single software tool that can consistently outperform other." Their tests of CNN models showed that Caffe and Tensorflow performed best on CPUs with 4 and 16 threads respectively. On GPU, Caffe, Tensorflow and CNTK were the best depending on mini-batch size and GPU type.

When selecting a deep learning framework, one should also consider user community involvement. This is an important factor as an open-source project cannot be developed and maintained well without appropriate amount of interest from the programmer's community. In order to estimate popularity of a deep learning library I gathered statistics from Stackoverflow [43] and GitHub [44]. Stackoverflow is a popular on-line programming question and answer community. Github is one of the largest web-based code repository hosting services. The source code of all deep learning libraries mentioned in this project is hosted on GitHub at the time of writing. The number of questions on Stackoverflow related to each deep learning library and number of subscribers to the corresponding code repositories are compared in Figure 12. While, according to [42], there is no clear performance leader, the programmer's community seems to have already selected its favorite deep learning library. There are almost four times more questions about TensorFlow than about its runner up Caffe and Theano and almost three times more people are watching its repository. On the other hand CNTK that showed good performance during tests performed in [42] seems to be almost ignored by the community. As of January 2017 there were only 51 questions about CNTK on Stackoverflow.

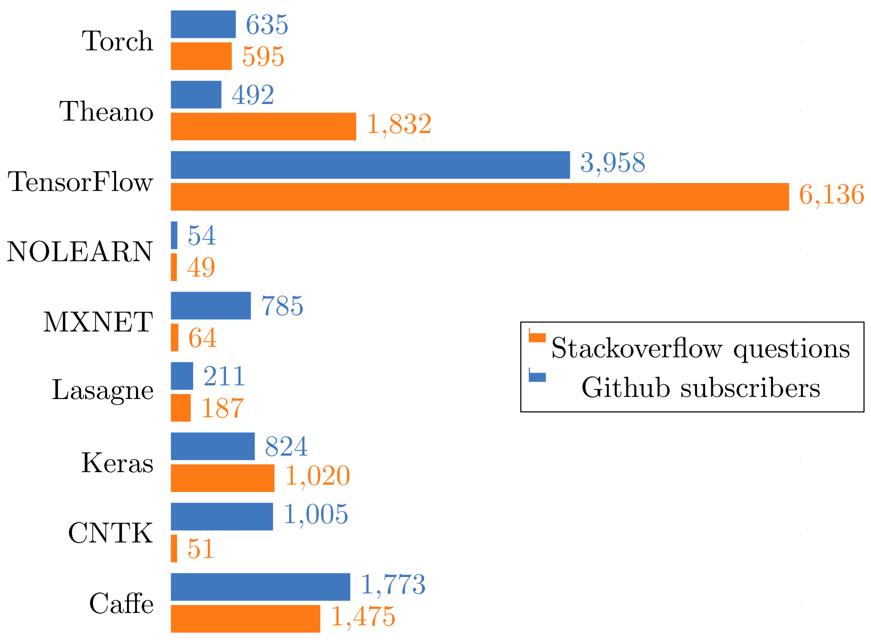


Figure 12: Popularity of deep learning frameworks as of January 2017.

Licensing is another important issue. All deep learning libraries presented here are distributed under one of the following open-source licenses [45]: MIT, Apache and BSD. The only exception is CNTK which also has a permissive license but has not adopted any of the more conventional licenses.

One should also mention deep learning libraries that are built on top of other libraries to make more a user friendly API. Lasagne and Keras [46] are such examples. Lasagne is built on top of Theano. Keras is built on top of both Theano and TensorFlow. It is easy to switch backend from Theano to TensorFlow and vice versa in Keras which makes it a very powerful tool. Keras API is lightweight and simple which makes it a great prototyping tool. A clear drawback of such "on-top" libraries is a lag between introduction of new functionality in primary libraries and in on-top libraries. Furthermore, additional layers of abstraction might reduce performance.

Note that the list of deep learning software tools from Table 4 is just a small fraction of all available tools. I decided to include only those tools that are well established in terms of documentation, community support and developer's reputation. This absolutely does not mean that other tools are of poor quality, but their learning curve might be steeper compared to the well established tools.

When it comes to programming language APIs, Python is a primary choice of the majority of the deep learning tools. Nearly all tools mentioned in this report have well documented extensive Python APIs. Linux is the preferred operating system for deep learning environments. There is a number of machine learning frameworks available for Microsoft .NET framework and one of the most promising is Accord.NET [47]. However at the time of writing it does not provide CNN functionality and its community support and development resources can not be compared to the leading deep learning tools.

All experiments presented in this project have been performed using TensorFlow and Keras software tools on a machine with Linux operating system. Both TensorFlow and Theano support Windows, so all experiments can be replicated on a Windows machine.

Table 4: Comparison of characteristics of deep learning frameworks

	Linux		Windows		Mac OS		Language bindings		
	CPU	GPU	CPU	GPU	CPU	GPU	Python	C++	License
Caffe	✓	✓	✓	✓	✓	✓	✓	✗	BSD 2
CNTK	✓	✓	✓	✓	✗	✗	✓	✓	Permissive
Keras	✓	✓	✓	✓	✓	✓	✓	✗	MIT
Lasagne	✓	✓	✓	✓	✓	✓	✓	✗	MIT
MXNET	✓	✓	✓	✓	✓	✓	✓	✓	Apache 2.0
NOLEARN	✓	✓	✓	✓	✓	✓	✓	✗	MIT
TensorFlow	✓	✓	✓	✓	✓	✓	✓	✓	Apache 2.0
Theano	✓	✓	✓	✓	✓	✓	✓	✗	BSD 3
Torch	✓	✓	✓	✓	✓	✓	✗	✓	BSD 3

7 Network Architectures

Five network configurations are tested in this project. I start with the simplest configuration possible — one convolutional layer followed by two fully connected layers and gradually add additional convolutional layers. Let us call the networks LipNet-x, see Figure 13, where x stands for the number of convolutional layers.

- **LipNet–1** is the simplest possible configuration and a starting point for the experiments. It consists of one convolutional block with max pooling and dropout followed by two fully connected layers.
- **LipNet–2** is an extension of LipNet–1 with one additional convolutional block.
- **LipNet–4** is mainly inspired by VGG–11 architecture. Two convolutional blocks with max-pooling and dropout are followed by a fully connected layer. All convolutional layers have the ReLU activation function, the *same* padding mode, 3×3 filter size, 32 channels and stride equal to 1. The input size is fixed to 28×28 pixels.
- **LipNet–4c** is inspired by ideas of fully convolutional networks and replaces max-pooling with convolution with a stride as in [20]. So max-pooling layers are replaced with convolutional layers with 2×2 stride, flatten operation is replaced with 1×1 convolution with 3 channels. Input size is fixed to 28×28 pixels.
- **LipNet-6** is a modification of LipNet–4 with an additional convolutional block. Additional convolution with stride requires bigger input size so images are resized to 32×32 pixels.

Classes are one-hot encoded [48] as demonstrated in Table 5.

Table 5: Class encodings

Class	Encoding
Multilamellar	`100`
Unilamellar	`010`
Uncertain	`001`
Empty	`100`
Full	`010`
Uncertain	`001`

Thus, the output layer is always represented by three nodes and each of them yields a value between 0 and 1 that is interpreted as a probability of belonging to a corresponding class. Softmax [49] activation function guarantees that all node values are in the range $[0, 1]$ and sum up to 1.

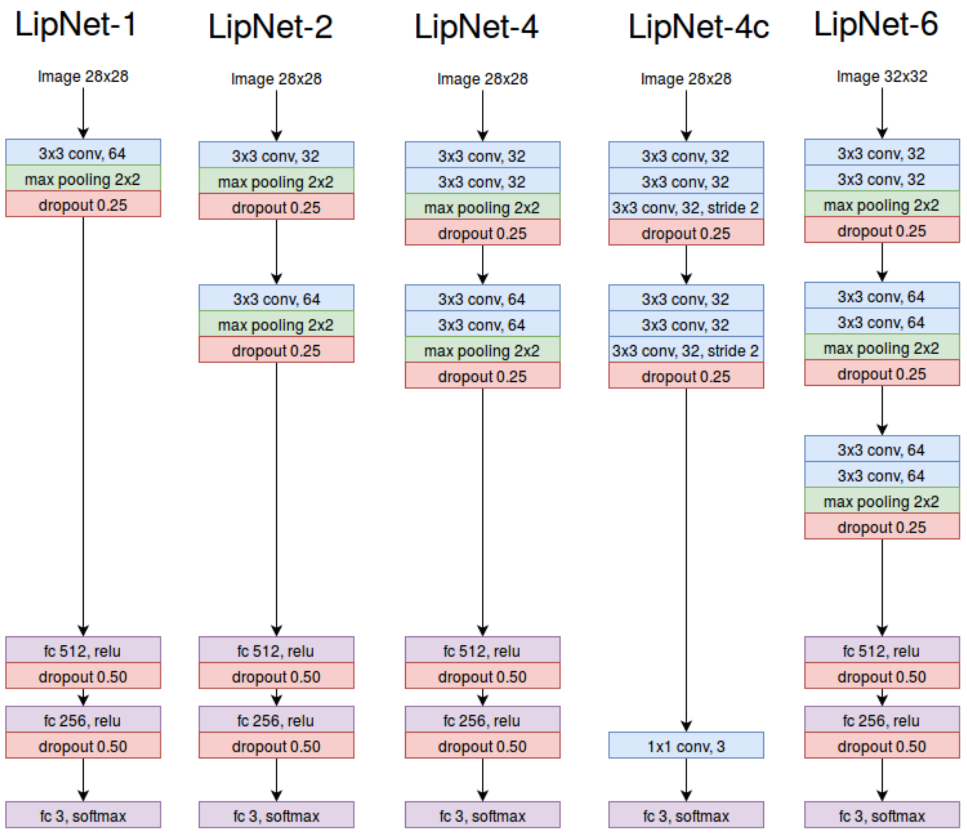


Figure 13: Network configurations that have been tested in this project. All networks assume *same* padding mode, unit stride and ReLU activations.

Figure 14 shows an example of visualized convolutional filters of the trained LipNet–4 model. Filters are visualized with a seismic color map in order to clearly see negative and positive values: blue color corresponds to values less than zero; red — greater than zero and finally white corresponds to zero values. Figure 15 shows example output of different convolutional layers of LipNet–4 model.

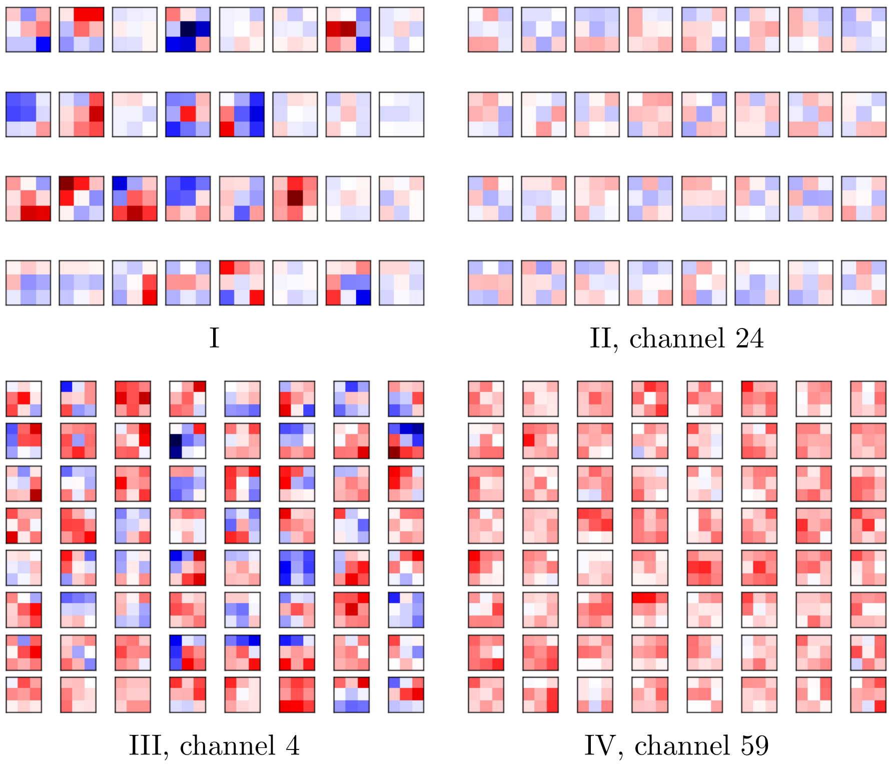


Figure 14: Visualization of convolution kernels

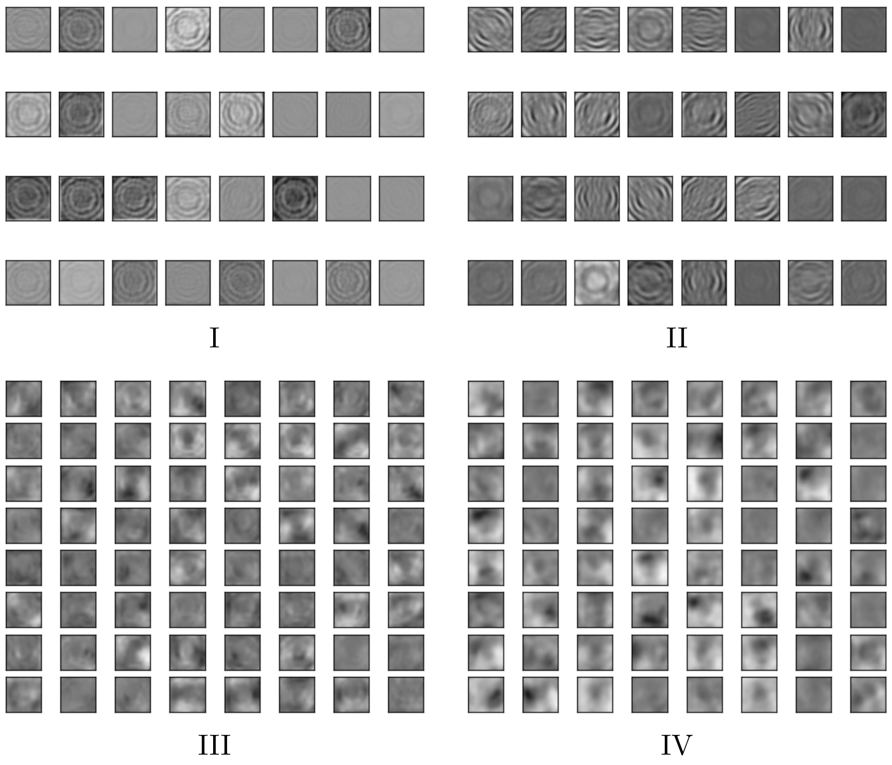
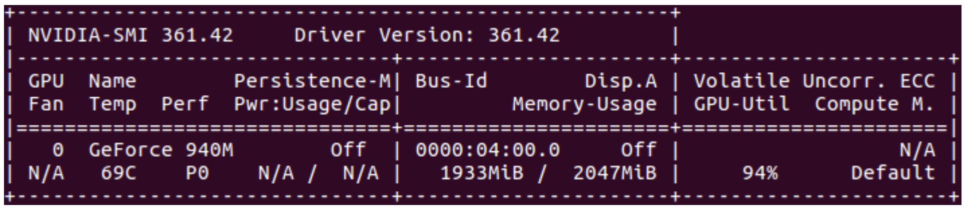


Figure 15: Visualization of the output from the convolutional layers.

8 Results

8.1 Experiment set-up

LipNet networks were trained using Keras deep learning library with GPU based Tensorflow backend on a Linux machine. Keras allows to easily switch backend between Tensorflow and Theano so it is possible to run the same code in Windows environment as well. It takes between 60 and 130 seconds per epoch to train LipNet models on a machine with a GeForce 940M video card. GPU utilization statistics, see Figure 16, is invoked using the `nvidia-smi` command on Linux.



```
+------------------------------------------------------+
| NVIDIA-SMI 361.42     Driver Version: 361.42         |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce 940M        Off  | 0000:04:00.0     Off |                  N/A |
| N/A   69C    P0    N/A /  N/A |   1933MiB /  2047MiB |     94%      Default |
+-------------------------------+----------------------+----------------------+
```

Figure 16: GPU utilization statistics.

All models are evaluated using a 5-fold cross-validation [50] technique. The data is split into 5 folds using stratified sampling which means that class proportions are preserved. One fold is used as the test set while the four others serve as training sets and the experiment is repeated five times, each time with a new fold as the test set.

Training sets are balanced using oversampling while test sets are left unbalanced. During training, the training data is augmented as described in 5.5. Data augmentation transformations are applied randomly to each training example, so the probability to see the same training example twice is quite low.

Input samples are shuffled at the beginning of each epoch. Training data is fed in batches and model parameters are updated after each batch. Batch size is fixed to the default value of 32 images. If a validation set is used, then one can see that at the end of some epochs the validation error is less than the test error. This happens because the model parameters are updated after each batch, so validation which is done at the end of each epoch is performed after training on the whole training set, while test error is an average error of all batches within an epoch.

8.2 Evaluation of data augmentation techniques

Evaluation of the data augmentation techniques is performed in the following way. All data is divided in five folds with preserved class proportion. In turn, each of them serves as a test set while the other four constitute a training set. All training sets are balanced using oversampling, test sets are not balanced.

The Model architecture used is the LipNet-4. Images contain only liposome objects, i.e. no liposome surrounding, and masks are not applied.

The first experiment is performed without any data augmentation, this corresponds to the first x axis tick denoted as *None* in Figure 17. Then, experiments are repeated adding augmentation techniques successively. For example the *Flip* tick means that this experiment was performed using *Rotation*, *Shift* and *Flip* techniques.

The y axis in Figure 17 represents five-fold average normalized true positive rate for each respective class.

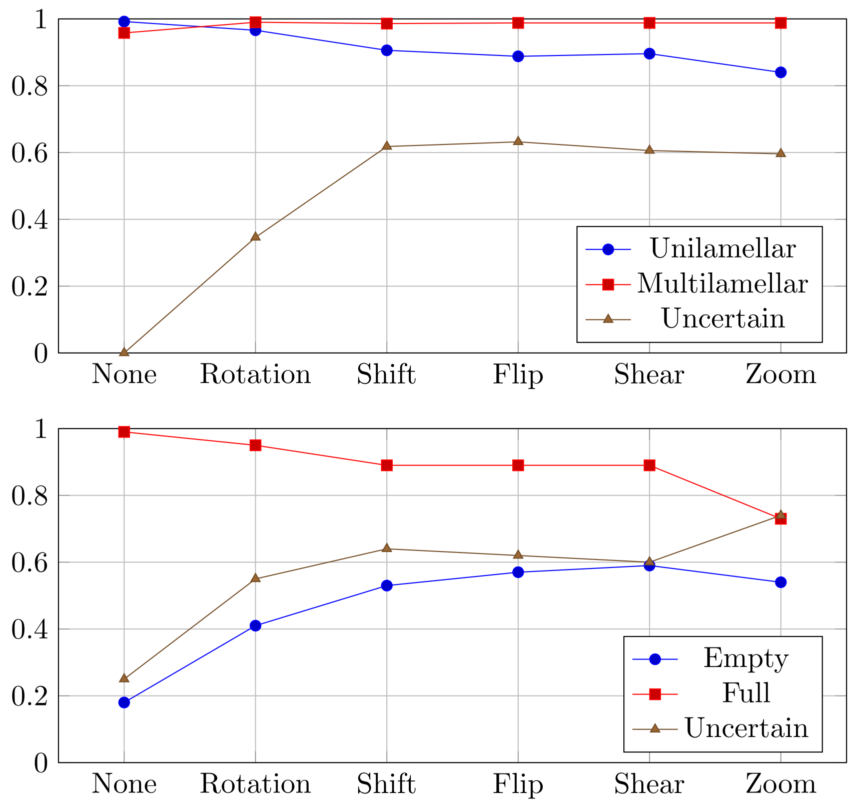


Figure 17: Effect of combining augmentation techniques successively on the average normalized true positive rate. Top — Lamellarity problem, bottom — Encapsulation.

In the **lamellarity** problem, the clear benefit of data augmentation is for the *uncertain* class. The majority class *unilamellar* and the runner up *multilamellar* are being classified with almost 100% accuracy even without any augmentation of training data. However the *uncertain* class is totally misclassified when no augmentation is performed. Rotation itself raises the true positive rate of *uncertain* to nearly 40% which makes sense. Considering the very small number (84) of examples in this class, it is not an unlikely scenario when the test set contains patterns that the model has not seen during training. For example the train and test set contain patterns that are similar but differently oriented. In such a case, rotation improves the result a lot which is the case in this experiment. Adding shift transformation brings further improvement to recognition of *uncertain* class objects. Adding flip, shear and zoom transformations do not have any positive effect on performance. In fact, more aggressive data augmentation worsens the model's ability to generalize and the performance on *unilamellar* and *uncertain* classes drops a bit when all transformations are used.

The **encapsulation** data is even more unbalanced than the lamellarity data, so without any data augmentation almost all examples are classified as *full*. Data augmentation makes clear improvement in performance, the true positive rates of the *empty* and *uncertain* classes are increasing while *full* slowly decreases from approximately 1.0 to 0.9 as rotation and shift transformations are employed. Adding additional transformations do not improve performance and the trend is quite similar to the one observed in the lamellarity problem except for the last point. Adding zoom transformation improves the model's ability to recognize *uncertain* examples while it worsens the performance for *full* examples. This is an unexpected result, since it would be natural to expect general performance improvement when zoom transformation is used. Zoom encourages the model to focus more on the core of the liposome which is the most relevant part in the encapsulation problem.

In order to analyze the effect of the zoom transformation in more detail an additional experiment was performed. This time the data was augmented using only zoom transformation. Results are presented in Table 6. The performance is approximately equal to rotation-only augmentation.

Table 6: 5-fold average normalized confusion matrix, encapsulation problem, zoom only.

	Empty	Full	Uncertain
Empty	0.43	0.13	0.44
Full	0.04	0.90	0.06
Uncertain	0.14	0.35	0.51

The best option would be to evaluate all possible combinations of transformations applied in all possible orders but that would require to run 325 experiments. Unfortunately due to hardware and time limitations it is not feasible to perform such a number of experiments.

8.3 Evaluation of different CNN models

In this section different LipNet models are evaluated and compared. The average F_1 score is recorded for each class.

Results for the lamellarity problem are shown in Figure 18. LipNet-4 is a clear leader that outperforms all other models for all classes. It is interesting to note that multilamellar liposomes are recognized by LipNet-4 with almost no error despite the fact that only 12% of all particles in the data set belong to that class compared to 87% of unilamellar particles. All other models demonstrated more or less similar results regarding recognition of unilamellar and multilamellar liposomes. LipNet-4 and LipNet-6 were the best to recognize particles of the uncertain class, clearly outperforming the other models.

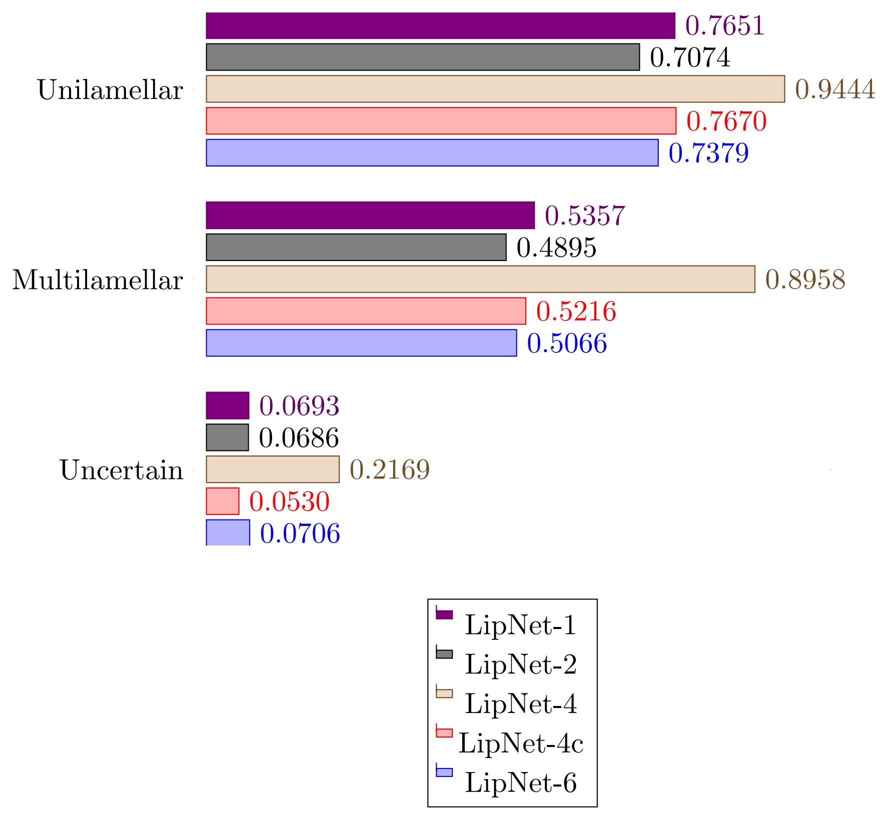


Figure 18: Lamellarity problem. F_1 scores for different CNN models.

Results for the encapsulation problem are shown in Figure 19. Unlike the lamellarity problem there is no clear leader, so no model outperforms the other for all classes. Simpler models like LipNet-1 and LipNet-2 are almost as good as their more advanced peers. LipNet-1 and LipNet-2 recognize the *full* class with the same performance as other models but they are marginally worse than LipNet-4 and LipNet-4c when it comes to the *empty* class and the *uncertain* class. LipNet-6 is marginally better than other models in recognizing *empty* and *uncertain* but it is significantly worse when it comes to the *full* class. It probably has too many parameters and overfits which leads to poor generalization.

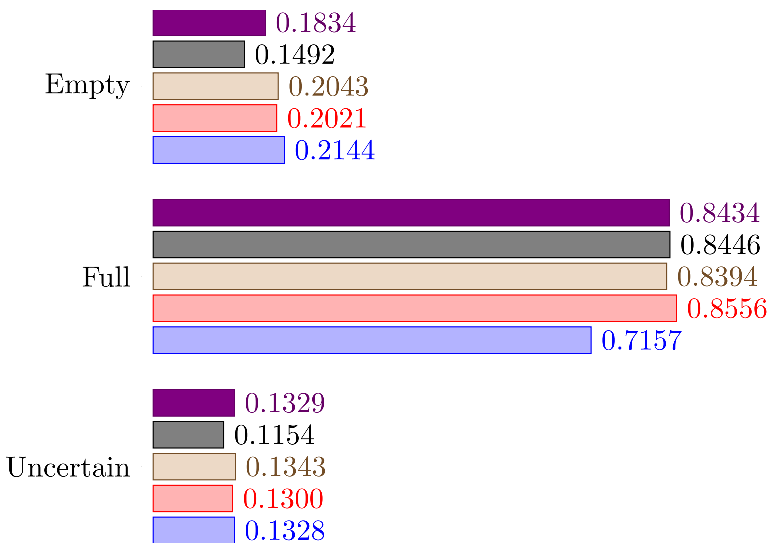


Figure 19: Encapsulation problem. F_1 scores for different CNN models.

8.4 SVM

SVM is currently used by Vironova for automatic classification and it is included in this project for benchmarking purposes. No additional experiments were performed, just those that were necessary to replicate Vironovas result. The following features are used to train the SVM models.

- Area
- Circularity
- Image moments μ_{20}, μ_{02}, μ_{30}, μ_{03}
- Edge density profile
- Histogram
- Internal segmentation variance

The input data is divided into training and test sets in the same way as for the CNN experiments. However, training sets are not balanced this time, instead class weights are adjusted inversely proportional to the class frequencies in the input data.

Tables 7 and 8 present average normalized confusion matrices for the encapsulation and lamellarity problems, respectively. Data sets are so unbalanced that total accuracy coincides with the true positive rate for the majority classes, which are *empty* and *unilamellar*, respectively.

Table 7: 5-fold average normalized confusion matrix for the encapsulation problem using the SVM classifier.

	Empty	Full	Uncertain
Empty	0.89	0.01	0.10
Full	0.04	0.87	0.09
Uncertain	0.18	0.10	0.72

Table 8: 5-fold average normalized confusion matrix for the lamellarity problem using the SVM classifier.

	Unilamellar	Multilamellar	Uncertain
Unilamellar	0.94	0.04	0.02
Multilamellar	0.00	0.98	0.02
Uncertain	0.03	0.11	0.86

8.5 Impact of surrounding and masking on the input images

Recall that each image contains a liposome object and its surrounding which goes 50 pixels in each direction. Corresponding particle masks are also available. It is hence a matter of design whether to include surrounding or not as well as if masks are to be applied. The following three alternatives have been tested:

- Input images with surrounding. All images include information about the particle's surrounding.
- Cropped images. All images are cropped to a minimum size allowed while still containing the liposome object.
- Cropped and masked. All images are cropped and the corresponding masks are applied. This means that each image contains information only about a particular particle and nothing else.

The comparison is performed in terms of F_1 score (averaged across 5 folds) for the LipNet-4 network trained on images in the modes described above.

Results of the experiments show that including surrounding to the input images leads to worse performance in both problems for all classes. The best performance is achieved when the input images are cropped and masked. A detailed comparison is shown in Figures 20 and 21.

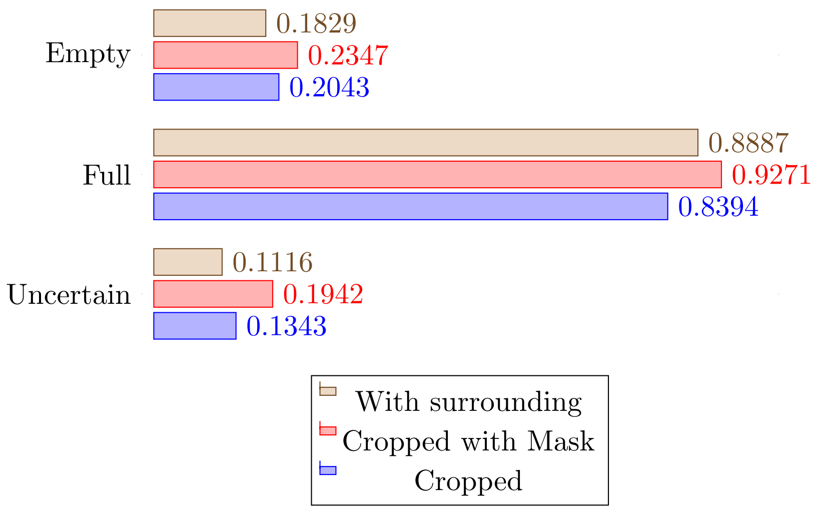


Figure 20: Comparison of the F_1 score of the LipNet-4 network for the encapsulation problem using different input image modes: with surrounding, cropped, cropped and masked.

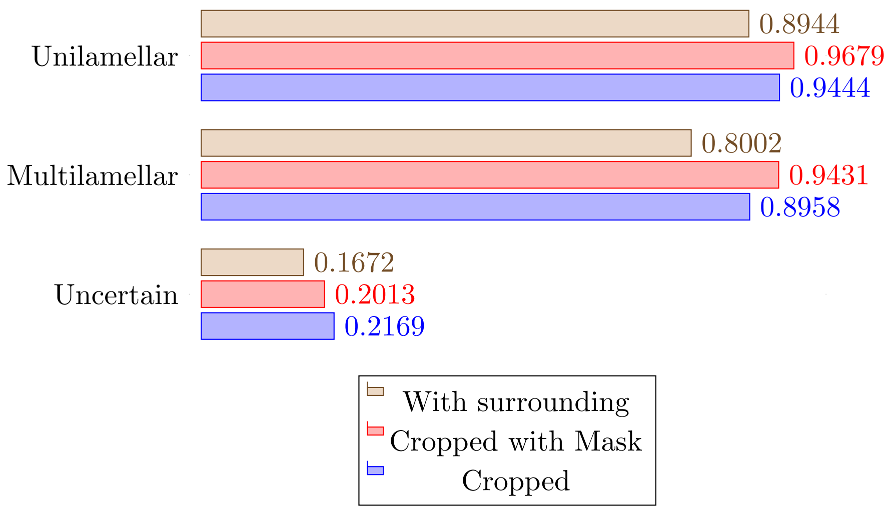


Figure 21: Comparison of the F_1 score of the LipNet-4 network for the lamellarity problem using different input image modes: with surrounding, cropped, cropped and masked.

9 Benchmarking

In this section selected CNN models are benchmarked against SVM. The LipNet-4 model has been selected for the final experiment which is comparison to SVM. In both cases data sets are balanced using oversampling, the input images are cropped, masked and augmented. All performance metrics are presented in Figure 22.

Unilamellar and multilamellar liposomes are recognized by both LipNet-4 and SVM with reasonably good results, but CNN slightly outperforms SVM. In particular CNN is better in predicting that a particle is not unilamellar. The CNN's negative predicted value is 0.7 against 0.5 for SVM. On the other hand, results are opposite when it comes to the *uncertain* class of the lamellarity problem. Here SVM clearly outperforms CNN in terms of sensitivity: SVM identifies nearly all particles of the *uncertain* class while CNN does so only in 70% of cases. However both methods yield many false positive predictions of the *uncertain* class which results in poor precision. The CNN has almost twice the precision compared to SVM but it is still very poor, just slightly above 10%. Mainly unilamellar liposomes are confused with uncertain ones by both CNN and SVM. Even if a small fraction of unilamellar liposomes are falsely identified as *uncertain*, it is enough to bring down precision because the data sets are heavily unbalanced. The unnormalised confusion matrix averaged across 5 folds is presented in Table 9.

Table 9: 5-fold average unnormalised confusion matrix, Lamellarity problem, LipNet-4.

	Unilamellar	Multilamellar	Uncertain
Unilamellar	2325	15	124
Multilamellar	4	323	16
Uncertain	2	3	11

The encapsulation problem is solved by both LipNet-4 and SVM with approximately the same performance. Results for the *full* class are the same for both classifiers. LipNet-4 is marginally better in predicting the *empty* class while SVM is slightly better in predicting the *uncertain* class. Positive predicted values of the *empty* and the *uncertain* classes demonstrate the same pattern as the *uncertain* class from the lamellarity problem. Approximately 15% of the positive *empty* predictions and 10% of the positive *uncertain* predictions are correct because some *full* liposomes are falsely classified as either *empty* or *uncertain*. Once again, even if a small fraction of *full* liposomes are misclassified this reduces precision of the minority classes drastically. Another notable result is the low negative predicted value of the *full* class. It means that classifiers yield more false negative predictions than true negative. On the other hand the positive predicted value of both the CNN and the SVM for the *full* class is almost 1 which means that there are hardly any *empty* or *uncertain* particles that are falsely classified as full. In other words it means that if a classifier says that a particle is *full* then it is most likely *full*. However, when it says the opposite it is most likely to be false. In this way the classifier is very careful in predicting a particle being *full*. It would rather yield a false negative *full* prediction than a false positive. It may be both good and bad depending on the domain. Discussion of risks and potential side effects of under- and overestimating the rate of liposomal encapsulation are beyond the scope of this project.

The unnormalised confusion matrix averaged across 5 folds is presented in Table 10.

Table 10: 5-fold average unnormalised confusion matrix, the encapsulation problem, LipNet-4.

	Empty	Full	Uncertain
Empty	20	1	12
Full	116	4201	534
Uncertain	22	16	63

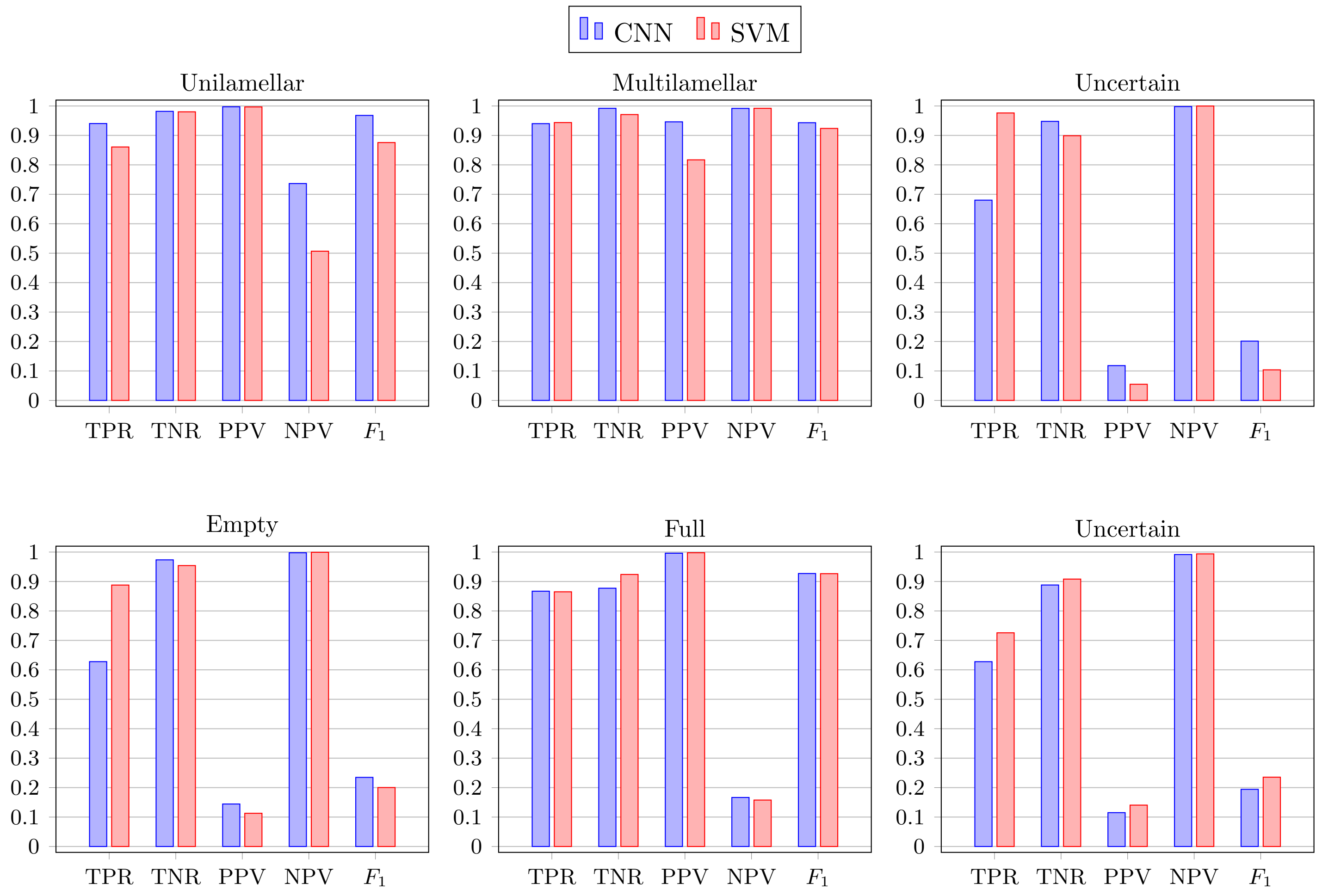


Figure 22: Comparison of the performance of SVM and CNN on the lamellarity and encapsulation data (best viewed in color). Input images are without surrounding and masks are applied.

10 Discussion

Comparison of different input image modes showed that the best performance is achieved when input images contain only particles. Including surrounding worsens the classifier's performance. The result is intuitive and expected. By excluding all information irrelevant to the particle, we make images less noisy and let the classifier concentrate on the particle. This has one drawback though: masking implies reading the mask files and performing multiplications which requires additional time. Experiments showed that in the encapsulation problem masking improved the 5-fold average F_1 score of Full and Empty classes by 10% and 15% respectively and in the lamellarity problem the F_1 score of Unilamellar and Multilamellar classes went up 2% and 5% respectively. On the other hand F_1 score of the Uncertain class in the encapsulation problem went 6% down after applying masking but I still consider masking worth applying given the increase in the performance for other classes.

Comparison of SVM and CNN classifiers has shown that both the lamellarity problem and the encapsulation problem can be solved with both methods with approximately the same performance. CNN is marginally better in predicting all classes of the lamellarity problem and some classes of the encapsulation problem, namely *empty* and *full*. SVM is better in predicting the *uncertain* class of the encapsulation problem. CNN may be preferred due to its ability to generalize to almost any classification problem. SVM performed well due to the careful choice of image features that was done by image analysis specialists. However there is no guarantee that with the same features SVM would perform well solving some other problem. On the contrary, CNN does not require input of such expert knowledge. The ability to learn descriptive image features in non-trivial domains is the cornerstone of convolutional neural networks. Given a data set of reasonable size, the CNN model can be relatively easy retrained and adopted for almost any kind of classification problem. A rough rule of thumb is that a CNN model will generally achieve acceptable performance with around 5000 labeled examples per category [14]. Of course, such flexibility has its cost. CNN models are more time consuming than SVM and still require more effort from software engineers to be set up and deployed in a production environment. However this is changing rapidly.

Deconvolutional neural networks is a powerful visualization tool that can help to understand what kind of image feature the CNN has learned. Unfortunately due to time and software limitations no experiments with deconvolutional networks have been performed. According to [51] it is quite common that convolutional neural networks trained on natural images learn features similar to a set of Gabor filters and color blobs on the first layer. Natural images means images that are neither computer generated nor shot in controlled conditions. It is hard to say what kind of features LipNet models have learned by just looking at its filters as shown in Figure 14. One can not say that LipNet networks learned anything like a set of Gabor filters or any other well known filters. One possible explanation can be that unlike data sets of natural images, Lamellarity and Encapsulation images are very similar to each other and CNN models have to catch differences that can barely be seen by a human eye. So a CNN model does not need to understand that a particle is round because all particles of all classes are round, this information would be useless in determining whether a liposome is full or empty. However a CNN would most probably learn such kind of features if the task would be to recognize liposomes among other objects of different shapes.

The DNN method may be used as an auxiliary tool. For example it can aid SVM classifier. Vironova uses SVM classifiers to solve its classification tasks and it performs well. However one must make a good choice of features in order to succeed and it is not always obvious what kind of features to select. DNN can help to decide which features are descriptive related to a certain task. In such a situation it may be helpful to train a convolutional neural network and then visualize it. Features that the CNN has learned can be added to the SVM pipeline.

10.1 Limitations

Due to time constraints an experiment with fully convolutional neural networks and variable size inputs has not been performed. The size of a particle seems to be an important factor as one can see some patterns of class distributions depending on the size, as shown in Figures 3 and 4. Firstly, due to the performance optimization, it is necessary to know in advance the size of all network layers, which requires constant input size. Convolutional neural networks can operate on a variable size input but it is a challenging task to implement that. At the time of writing this report, leading deep learning tools do not provide recipes for fully convolutional networks for classification tasks. One possible solution would be to fix the size of all images to the maximum of the sample and zero pad all smaller images up to that size. In this way a network would possibly see zero padding as a feature and distinguish between different sizes but that would require a lot of computational power. In practice, training networks with the input size 64×64 required approximately 24 hours. For example, images of multilamellar liposomes are generally greater than 100×100, so it would take an unreasonably large amount of time to train models with inputs of such size.

Due to hardware limitations it was impossible to test deeper networks with many layers. It is however doubtful that deeper network would have better generalization as the risk for overfitting increases as the network structure becomes more complicated. The LipNet-6 model which has the largest number of convolutional layers performed worse than its simpler counterparts.

This project is of research character so experiments were performed in a research environment, not in a production one. It is possible to replicate the results of this project in Windows, but it would require a significant amount of effort to adopt the presented solutions in a production environment.

11 Conclusion

Five different network architectures have been tested and compared in this thesis. A network with two convolutional blocks each of which has two convolutional layers performed best for both the lamellarity problem and the encapsulation problem.

Convolutional neural networks trained on cropped and masked images demonstrated the best performance among all input image modes. Cropping and masking excludes irrelevant information from the image and in that way makes it less noisy.

The overall performance shown by the CNN models was reasonable. CNN performed approximately on the same level as SVM for both problems. CNN was slightly better than the SVM model for the lamellarity problem and achieved almost the same performance as the SVM model for the encapsulation problem. Convolutional neural networks offer greater flexibility and generalization compared to SVM because CNN does not require expert knowledge in order to choose image features. However such flexibility comes at the cost of greater computational overhead.

Regarding optimization methods, both ADAM and SGD performed well and no method outperformed the other.

The data set imbalance has proven to be a major problem. Unless the imbalance problem is addressed, classifiers are heavily biased towards the majority classes and almost ignore the underrepresented classes. It has been shown that oversampling combined with data augmentation can mitigate the imbalance problem and lead to a reasonable level of generalization. The most common image augmentation techniques like rotation, shift, flip, shear and zoom have been compared. Rotation and shift were the most helpful techniques, however it is context-dependent. For example, the zoom technique was almost as helpful as rotation for the encapsulation problem, while rotation was far more effective than zoom for the lamellarity problem. An attempt to train the CNN models on the artificial data unfortunately did not improve the performance. In fact performance of such models was very poor so the artificial data has not been used in any experiments that are presented in this report.

A number of deep learning software tools has been reviewed. As of October 2016 deep learning frameworks are still mainly a research tool and deployment of the trained models to production environments might demand significant amount of effort and time resources. However it is changing and integration of deep learning tools into production routines is expected to be easier in the near future. TensorFlow, Theano and Caffe are currently the most popular frameworks in terms of number of questions on Stackoverflow and number of subscribers on GitHub. All frameworks benefit from allocating computations on GPU and currently there is no clear leader in terms of performance.

Deconvolutional neural networks have been discussed as a promising method to help select features that would describe images from different problems in the best way.

To sum up, there are hardly any obstacles to use deep learning as a research and aid tool. Software tools reviewed in this thesis are fairly easy to set up on a Linux machine and are well documented. However it might be more difficult for Vironova to use these tools in a production environment, considering the limited support of Windows operating system and absence of C# application programming interfaces. It does not mean that it is impossible to adopt solutions presented in this report for production, but it would require a considerable amount of effort to do that.

12 Future work

Testing fully convolutional neural networks with variable size of the input images is one of the possible extensions to this project. The size of a particle might be an important feature so adopting a fully convolutional approach may improve the classifier's results.

Besides this, the results of the convolutional neural networks can be improved by expanding the training data sets. New samples might be difficult to retrieve, so alternative ways to expand the data set are of great interest. Even though an attempt to generate artificial data did not improve performance, it is a question worth to investigate further.

Another question worth investigating is combining the CNN model with another neural network trained on extracted image features. Two and more models can be joined at the output layer or earlier. Image features are extracted from the non-rescaled images so they can provide additional information that the CNN might have missed. This is potentially another source of improvement for the presented results.

References

[1] *Vironova AB company profile.* https://www.linkedin.com/company/vironova-ab. Accessed: 2016-10-21.

[2] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers". In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory.* COLT '92. Pittsburgh, Pennsylvania, USA: ACM, 1992, pp. 144–152. ISBN: 0-89791-497-X. DOI: 10.1145/130385.130401. URL: http://doi.acm.org/10.1145/130385.130401.

[3] Yann Le Cun. "Learning Process in an Asymmetric Threshold Network". In: *Disordered Systems and Biological Organization.* Ed. by E. Bienenstock, F. Fogelman Soulie, and G. Weisbuch. "Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 233–240. ISBN: 978-3-642-82657-3. DOI: 10.1007/978-3-642-82657-3_24. URL: http://dx.doi.org/10.1007/978-3-642-82657-3_24.

[4] Y. Le Cun et al. "Handwritten digit recognition: applications of neural network chips and automatic learning". In: *IEEE Communications Magazine* 27.11 (Nov. 1989), pp. 41–46. ISSN: 0163-6804. DOI: 10.1109/35.41400.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25.* Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[6] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network". In: *CoRR* abs/1312.4400 (2013). URL: http://arxiv.org/abs/1312.4400.

[7] Evan Shelhamer, Jonathan Long, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *CoRR* abs/1605.06211 (2016). URL: http://arxiv.org/abs/1605.06211.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III.* Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4. DOI: 10.1007/978-3-319-24574-4_28. URL: http://dx.doi.org/10.1007/978-3-319-24574-4_28.

[9] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. “Multi-column Deep Neural Networks for Image Classification”. In: *CoRR* abs/1202.2745 (2012). URL: http://arxiv.org/abs/1202.2745.

[10] G.V. Betageri, S.A. Jenkins, and D. Parsons. *Liposome Drug Delivery Systems*. Taylor & Francis, 1993. ISBN: 9781566760300. URL: https://books.google.se/books?id=b2FNM5mne50C.

[11] *Liposomes and their uses in biology and medicine*. New York Academy of Sciences, 1978.

[12] R. Cammack et al. *Oxford Dictionary of Biochemistry and Molecular Biology*. Oxford Reference Online. OUP Oxford, 2006. ISBN: 9780198529170. URL: https://books.google.se/books?id=XpUjsqD7lFUC.

[13] Ming-Kuei Hu. “Visual pattern recognition by moment invariants”. In: *IRE Transactions on Information Theory* 8.2 (Feb. 1962), pp. 179–187. ISSN: 0096-1000. DOI: 10.1109/TIT.1962.1057692.

[14] Ian Goodfellow Yoshua Bengio and Aaron Courville. “Deep Learning”. Book in preparation for MIT Press. 2016. URL: http://www.deeplearningbook.org.

[15] S. Knerr, L. Personnaz, and G. Dreyfus. “Single-layer learning revisited: a stepwise procedure for building and training a neural network”. In: *Neurocomputing: Algorithms, Architectures and Applications*. Ed. by Francoise Fogelman Soulie and Jeanny Herault. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 41–50. ISBN: 978-3-642-76153-9. DOI: 10.1007/978-3-642-76153-9_5. URL: http://dx.doi.org/10.1007/978-3-642-76153-9_5.

[16] Chih-Chung Chang and Chih-Jen Lin. “LIBSVM: A library for support vector machines”. In: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software available at http://www.csie.ntu.edu.tw/cjlin/libsvm, 27:1–27:27.

[17] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks”. In: *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 818–833. ISBN: 978-3-319-10590-1. DOI: 10.1007/978-3-319-10590-1_53. URL: http://dx.doi.org/10.1007/978-3-319-10590-1_53.

[18] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *CoRR* abs/1409.1556 (2014). URL: http://arxiv.org/abs/1409.1556.

[19] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *CoRR* abs/1409.4842 (2014). URL: http://arxiv.org/abs/1409.4842.

[20] Jost Tobias Springenberg et al. “Striving for Simplicity: The All Convolutional Net”. In: *CoRR* abs/1412.6806 (2014). URL: http://arxiv.org/abs/1412.6806.

[21] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). URL: http://arxiv.org/abs/1412.6980.

[22] Tong Zhang. “Solving Large Scale Linear Prediction Problems Using Stochastic Gradient Descent Algorithms”. In: *ICML 2004: PROCEEDINGS OF THE TWENTY-FIRST INTERNATIONAL CONFERENCE ON MACHINE LEARNING. OMNIPRESS*. 2004, pp. 919–926.

[23] Yann LeCun et al. “Efficient BackProp”. In: *Neural Networks: Tricks of the Trade*. Ed. by Genevieve B. Orr and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 9–50. ISBN: 978-3-540-49430-0. DOI: 10.1007/3-540-49430-8_2. URL: http://dx.doi.org/10.1007/3-540-49430-8_2.

[24] X. Zhu and I. Davidson. *Knowledge discovery and data mining: challenges and realities*. Premier reference source. Information Science Reference, 2007. ISBN: 9781599042527. URL: https://books.google.se/books?id=zdJQAAAAMAAJ.

[25] Stephen V. Stehman. “Selecting and interpreting measures of thematic classification accuracy”. In: *Remote Sensing of Environment* 62.1 (1997), pp. 77–89. ISSN: 0034-4257. DOI: http://dx.doi.org/10.1016/S0034-4257(97)00083-7. URL: http://www.sciencedirect.com/science/article/pii/S0034425797000837.

[26] David M. W. Powers. *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation*. Tech. rep. SIE-07-001. Adelaide, Australia: School of Informatics and Engineering, Flinders University, 2007.

[27] Sergii Gryshkevych, Derrick Alabi, and Max Reeves. *Shelter Animal Outcomes*. Uppsala University, Department of Information Technology. 2016.

[28] Lorenzo Rosasco et al. “Are Loss Functions All the Same?” In: *Neural Comput.* 16.5 (May 2004), pp. 1063–1076. ISSN: 0899-7667. DOI: 10.1162/089976604773135104. URL: http://dx.doi.org/10.1162/089976604773135104.

[29] Charles X. Ling and Victor S. Sheng. “Class Imbalance Problem”. In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 171–171. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_110. URL: http://dx.doi.org/10.1007/978-0-387-30164-8_110.

[30] Victoria López et al. "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics". In: *Information Sciences* 250 (2013), pp. 113–141. ISSN: 0020-0255. DOI: http://dx.doi.org/10.1016/j.ins.2013.07.007. URL: http://www.sciencedirect.com/science/article/pii/S0020025513005124.

[31] Nitesh V. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *J. Artif. Int. Res.* 16.1 (June 2002), pp. 321–357. ISSN: 1076-9757. URL: http://dl.acm.org/citation.cfm?id=1622407.1622416.

[32] Carolina Wählby Omer Ishaq Vladimir Curic. "Training of macine learning methods for fluorescent spot detection". In: (Apr. 2016).

[33] Bianca Zadrozny and Charles Elkan. "Learning and Making Decisions when Costs and Probabilities Are Both Unknown". In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '01. San Francisco, California: ACM, 2001, pp. 204–213. ISBN: 1-58113-391-X. DOI: 10.1145/502512.502540. URL: http://doi.acm.org/10.1145/502512.502540.

[34] Ken Perlin. "An Image Synthesizer". In: *SIGGRAPH Comput. Graph.* 19.3 (July 1985), pp. 287–296. ISSN: 0097-8930. DOI: 10.1145/325165.325247. URL: http://doi.acm.org/10.1145/325165.325247.

[35] K. C. Jim, C. L. Giles, and B. G. Horne. "An analysis of noise in recurrent neural networks: convergence and generalization". In: *IEEE Trans Neural Netw* 7.6 (1996), pp. 1424–1438.

[36] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[37] Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions". In: *arXiv e-prints* abs/1605.02688 (May 2016). URL: http://arxiv.org/abs/1605.02688.

[38] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).

[39] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[40] Dong Yu et al. *An Introduction to Computational Networks and the Computational Network Toolkit*. Tech. rep. Microsoft Research, 2014. URL: https://www.microsoft.com/en-us/research/publication/an-introduction-to-computational-networks-and-the-computational-network-toolkit/.

[41] Ronan Collobert, Koray Kavukcuoglu, and Clement Farabet. "Torch7: A Matlab-like Environment for Machine Learning". In: *BigLearn, NIPS Workshop*. 2011.

[42] Shaohuai Shi et al. "Benchmarking State-of-the-Art Deep Learning Software Tools". In: *CoRR* abs/1608.07249 (2016). URL: http://arxiv.org/abs/1608.07249.

[43] Lena Mamykina et al. "Design Lessons from the Fastest Q&#38;a Site in the West". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '11. Vancouver, BC, Canada: ACM, 2011, pp. 2857–2866. ISBN: 978-1-4503-0228-9. DOI: 10.1145/1978942.1979366. URL: http://doi.acm.org/10.1145/1978942.1979366.

[44] Ferdian Thung et al. "Network Structure of Social Coding in GitHub". In: *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*. CSMR '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 323–326. ISBN: 978-0-7695-4948-4. DOI: 10.1109/CSMR.2013.41. URL: http://dx.doi.org/10.1109/CSMR.2013.41.

[45] Lawrence Rosen. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004. ISBN: 0131487876.

[46] Chollet Francois. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[47] C. R. Souza. *The Accord.NET Framework*. http://accord-framework.net. Dec. 2014.

[48] D.M. Harris and S.L. Harris. *Digital Design and Computer Architecture*. Morgan Kaufmann. Morgan Kaufmann, 2013. ISBN: 9780123944245. URL: https://books.google.se/books?id=-DG18Nf7jLcC.

[49] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[50] Ron Kohavi. "A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection". In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'95. Montreal, Quebec, Canada: Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143. ISBN: 1-55860-363-8. URL: http://dl.acm.org/citation.cfm?id=1643031.1643047.

[51] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *CoRR* abs/1411.1792 (2014). URL: http://arxiv.org/abs/1411.1792.