



UPPSALA
UNIVERSITET

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



UPPSALA
UNIVERSITET

**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 type of liposomes. This includes determining structure of a liposome and presence of a liposomal encapsulation. Vironova has a lot of electron microscopy images so automatic classification is of great interest. The purpose of this project is to evaluate convolutional neural networks method for solving lamellarity and encapsulation classification problems. 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 support vector machines method which currently performs automatic classification tasks.

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

Contents

1	Introduction	2
2	Problem description	2
3	The data	4
4	Methodology	7
4.1	Support Vector Machines	7
4.2	Convolutional Neural Networks	7
4.3	Deconvolutional Neural Networks	8
4.4	Optimization	9
4.5	Performance measures	10
4.6	Loss function	11
4.7	Class Imbalance Problem	13
4.7.1	Oversampling	14
4.7.2	Undersampling	14
4.7.3	SMOTE	14
4.7.4	Artificial data	14
5	Regularization	16
5.1	Weight decay	16
5.2	Noise injection	16
5.3	Dropout	17
5.4	Early stopping	17
5.5	Data augmentation	17
6	Review of deep learning software tools	20
7	Network Architectures	24
8	Results	29
8.1	Experiment set-up	29
8.2	Evaluation of data augmentation techniques	29
8.3	Evaluation of different CNN models	31
8.4	SVM	33
8.5	Impact of surrounding and masking on the input images . . .	34
9	Benchmarking	36
10	Discussion	39
10.1	Limitations	40

11 Conclusion	41
12 Future work	42

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 has a lot of EM images and many tasks involve classification step therefore automatic classification is of great interest. Currently automatic classification is performed by means of support vector machines (SVM) [2] method and it demonstrates reasonable performance. On the other hand SVM operates on features representation of images rather than on raw pixel data. Determining a set of feature that would describe difference between classes is a non-trivial task that requires expert knowledge. This motivates interest in methods that do operate on raw pixel data and are capable of learning features themselves during training.

Convolutional neural network (CNN) [3] is one of such methods. CNN are designed to recognize visual patterns directly from pixel images 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 that 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 applying CNN method 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 discussions on library licenses, operating system availability, performance and community support.

2 Problem description

One of Vironova's electron microscopy services is to classify types of liposomes¹. This includes determining:

¹A 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 suggested 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].

- Structure of liposome.
- Presence of liposomal encapsulation of for example adeno-associated virus 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 number of lamellae liposomes can be *unilamellar* (single lamella) and *multilamellar* (multiple lamellae). In addition uncertain class is introduced because in some cases it is almost impossible to be certain about lamellarity class. For example liposomes can overlap each other. Another issue to keep in mind is that samples are usually delivered frozen, so liposomes may be partly covered with pieces of ice that result in large black bulbs on EM images. Figure 1 illustrates examples of liposomes from each class.

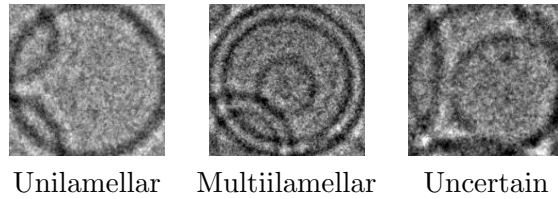


Figure 1: Lamellarity problem, 3 classes

Table 1 presents lamellarity problem data set which contains 14 169 EM images.

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 delivering throughout 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 encapsulation problem liposomes are classified between the following classes: *full*, i.e. liposomes that received drug substance; *empty*, i.e. liposomes that remained empty after encapsulation attempt. As in lamellarity problem *uncertain* class is introduced with the same motivation. Figure 2 illustrates different classes of encapsulation problem. Table 2 presents encapsulation problem data set which contains 24 918 EM images.

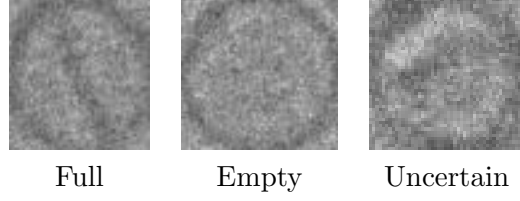


Figure 2: Encapsulation problem, 3 classes

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 lamellarity and incapsulation problems. Both data sets contain grayscale EM images of particles and a number of computed image features.

Each image depict exactly one liposome. All images are padded 50 pixels in each direction. Effect of particle surrounding on classifier performance is described later in this report. Corresponding particle masks are also provided. All images have different size. Figure 3 shows scatter plots where axes coordinates correspond to image width and height in pixels. Figure 4 demonstrates image size distribution inside each class.

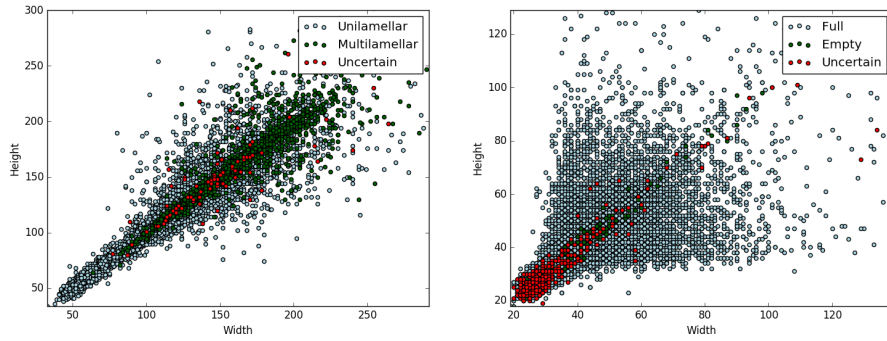


Figure 3: Scatter plots of images sizes (best viewed in color): lamellarity data set to the left hand side, encapsulation — to the rights hand side. Width and height units are pixels.

Both data sets contain following features:

- **Maximum width** in nanometres. If particle is an ellipse then it is diameter along the largest axis. If particle is a polygon then it is a maximum value of distance transform of particle mask.
- **Diameter** in pixels. If particle is an ellipse then it is diameter along the largest axis. If particle is a polygon then it is the greatest distance between any two vertices.
- **Length** is defined as diameter but it has nanometre as 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** It is a 2d array, first value in each pair in pixel is pixel intensity value and second one is distance in nanometre to the center of the particle. First entry of the array corresponds to the center of the particle, then all values are average of outward concentric rings.
- **Edge Density Profile** is defined in similar way to the radial density profile, though second value in each pair is defined in such way that it has 0 value at membrane.
- **Signal to noise** is a measure that says how much variance across the membrane is different compared to the variance along with the membrane.

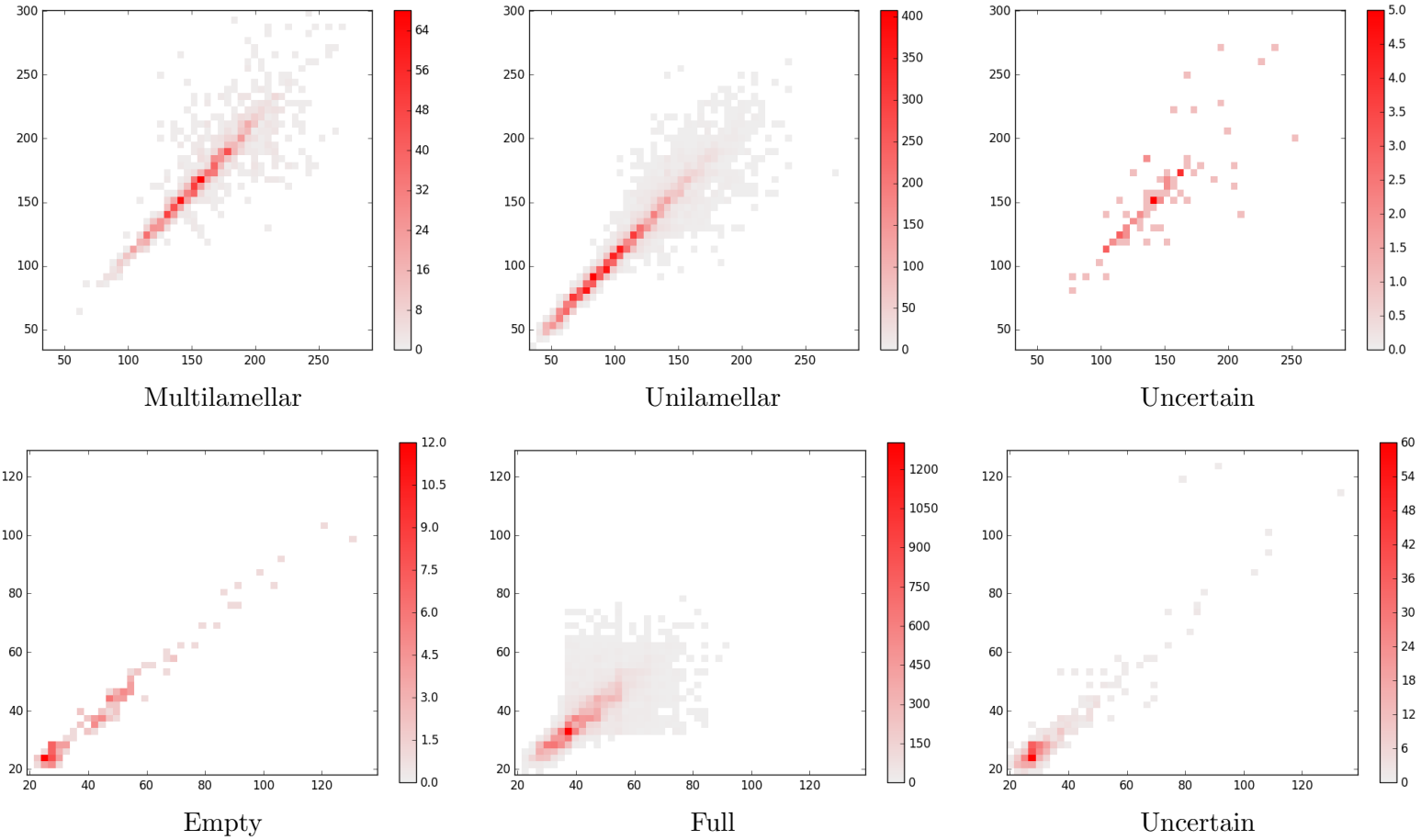


Figure 4: Distribution of image sizes in each class. Upper row corresponds to lamellarity problem, lower corresponds to encapsulation problem.

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 classification tasks SVM has the aim to determine decision boundaries that produce optimal class separation. SVM was initially designed for binary classification case but a number of modifications had been introduced for multiclass classification. One of them is "one-against-one" approach [15]. According to this technique $n(n-1)\frac{1}{2}$, where n is number of classes, classifiers are trained for each class. SVM does not provide probabilities, just a class identity [14]. LIBSVM [16] library is used for experiments in this project and it implements "one-against-one" technique.

Currently automatic classification is performed by means of SVM at Vironova. Performance of SVM method has shown up to 98% accuracy on lamellarity problem and up to 87% on encapsulation problem. The following 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

a 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 of CNNs for this project as the main tool for performing classification tasks is motivated in the following way.

Since 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 ImageNet 2012 competition that was won using a CNN model [5] with error rate of 16.4% which was a breakthrough at that time. There are also examples of successful application of CNNs in microscopy images domain. For example U-Net [8] has solved biomedical image segmentation task provided very limited training data set. Area of CNNs application is extensive and they are being used by all leading IT companies such as Google, Facebook, Microsoft and others. 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 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 of image analysis CNNs do not require handcrafted convolutional filters designed by an image analysis specialist, even though these features can be incorporated in a CNN model. For instance, lamellarity and encapsulation problems have already been successfully solved by Vironova specialists using SVM and carefully selected image features such as area, circularity, image moments and other discussed in section 4.1. This projects studies CNNs ability to achieve comparable result in fully unsupervised manner.

4.3 Deconvolutional Neural Networks

Convolutional neural networks is a powerful method of machine learning that is capable to learn image features in order to describe provided categories. However it is almost always not obvious what kind of feature has the model learned. For some time convolutional neural networks have been treated as a black box. This has 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 do convolutional neural networks "see" images. Figure 5 shows an example visualizations of features learned by a CNN.

The DN method can be summarized in the following way. The deconvolutional network goes down from activations of a given layer back to image. In this manner DN reverses data flow of a CNN undoing 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 rec-

tifiers 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. propose to record the locations of the maxima within each pooling region, they call these recorded values switch variables. Unpooling operation uses switches to place reconstructions into appropriate locations setting other values within upsampling region to zero.

Convolutional networks that replace max pooling operation with convolution with stride can be visualized in the similar way. Springenberg et al. [18] propose modifications to DN method that achieve similar results as in [17].

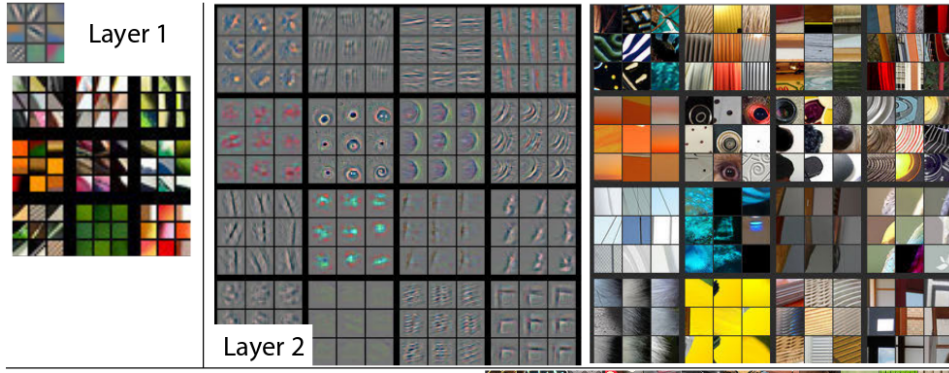


Figure 5: Visualization of features in a fully trained model. Figure is taken from [17].

DNN method may be used as supporting 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 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.

4.4 Optimization

Deep learning algorithms involve optimization in many contexts [14]. The most obvious one is finding network parameters that reduce a cost function. One of design decisions that one has to make is choice of optimization method. In this project I limited the choice of optimizers to ADAM [19] and Stochastic Gradient Descent (SGD) [20].

4.5 Performance measures

When evaluating a classification model one is almost always interested in how many predictions from all predictions made 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 in 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 predictive power of the model. Models with a given accuracy may have greater predictive power than models with higher accuracy. This is known as Accuracy Paradox [21].

Let's illustrate Accuracy Paradox using encapsulation problem. Table 2 on page 4 presents distribution of three classes in encapsulation data set. Consider a model that classifies all particles as full. Such model would achieve 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 random guess model would have 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 example when accuracy is misleading and fails as a performance measure. An unambiguous way to present the prediction results of a classification model is a confusion matrix [22]. Confusion matrix is a table with number of rows and columns that are both equal to 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 is dealing with binary classification. Table of confusion is presented in table 3 and it is usually used to illustrate performance measures that are derived from 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 [23]. The rest of this section presents performance measures that are used in this project for evaluation and comparing of different classification models.

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

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

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

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

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

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

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

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

F_1 **score** is a harmonic mean of TPR and TNR, which is a measure of a test's accuracy. F_1 score reaches its best value at 1 and worst at 0. F_1 has been criticized for not taking true negatives into account [23] but for the sake of simplicity let us use it anyway.

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

Presented above measures allow more detailed analysis of classifier performance and can clearly illustrate eventual bias of a classifier towards some class or classes.

4.6 Loss function

This section is taken from [24].

The multi-class logarithmic loss function is a loss function that represents the price paid for inaccuracy of predictions in classification problems [25]. The formula is:

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

Where

- N – number of observations
- M – number of classes, it is five 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 classifier marks the correct class as the most likely one, no matter how confident the prediction is.

Figure 6 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 6 that the multi-class logarithmic loss heavily penalizes classifiers that are confident about an incorrect classification.

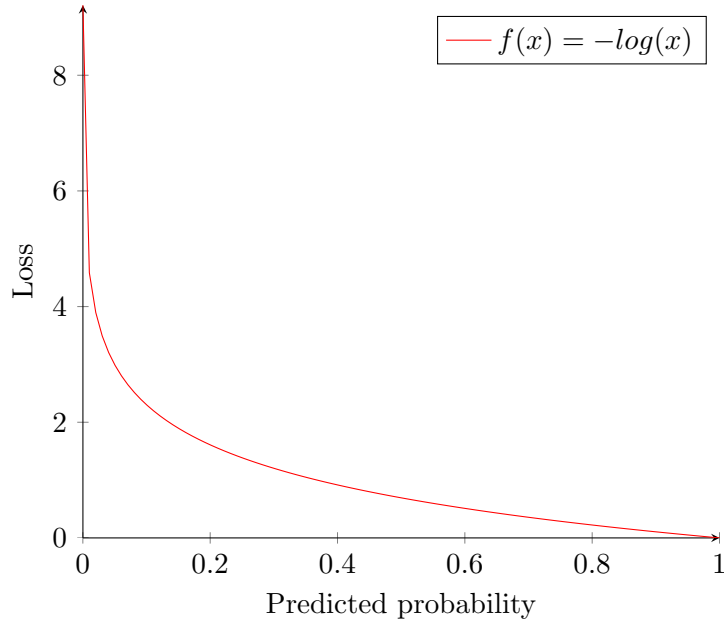


Figure 6: Log loss of a single class where predicted probability ranges from 0 to 1.

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.

4.7 Class Imbalance Problem

Encyclopedia of Machine Learning [26] defines 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 majority class treating examples of minority class as noise. Lamellarity and encapsulation data sets presented in sections 2 are highly unbalanced. In lamellarity data set majority class accounts for 87% of all samples and in encapsulation dataset majority class constitutes 97% of all samples. Preliminary experiments showed that classifiers trained on raw unbalanced data sets are highly biased towards majority classes and do not generalize. 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 [27]:

- *Data sampling* implies any means to produce more or less balanced data set. For example it can be oversampling, undersampling, SMOTE [28] or generating artificial examples [29]. Data sampling method perform usually best combined with data augmentation techniques.
- *Algorithm modification* this procedure is oriented towards the adaptation of base learning methods to be more attuned to class imbalance issues [30].
- *Cost sensitive learning* introduces higher penalties for misclassification of minority classes making learning algorithm more sensitive to underrepresented classes.

Listed above methods prevent classifiers from ignoring underrepresented classes during training phase. However this is very likely to lead to another problem — overfitting. Overfitting and how to combat it is discussed in section 5. The rest of this section discusses methods that have been used

to mitigate Class Imbalance Problem in lamellarity and encapsulation data sets.

4.7.1 Oversampling

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

4.7.2 Undersampling

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

4.7.3 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 [28]. Synthetic examples are generated by combining each minority class example with its randomly selected k nearest neighbors, where k is a parameter that depends on amount of required over-sampling.

4.7.4 Artificial data

Machine learning algorithms perform best provided 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 size of data sets is often limited. In such 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 discussed and evaluated in [29].

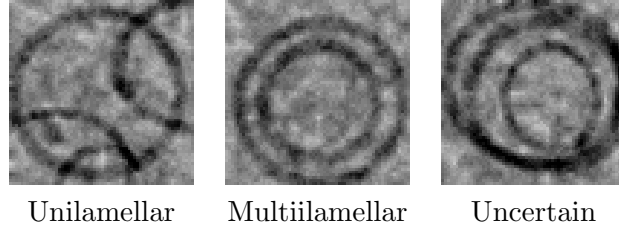


Figure 7: Artificial examples, lamellarity problem

In this project artificial data has been generated for lamellarity problem. Artificial EM images of liposomes are generated by drawing circles with carefully selected randomized distortions. Background and distortions are generated using Perlin noise [31]. Figure 7 presents examples of artificially generated EM images.

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 any combination of methods described in section 4.7. This would significantly reduce training error which means that the model is able to describe data generation process of the training set.

However as the result of seeing multiple copies of minority classes learning algorithm will almost likely describe random error or noise as well. Such model would not be able to make reliable predictions on general untrained data, i. e. its generalization error would still 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 lamellarity and encapsulation data sets.

5.1 Weight decay

Weight decay is a method of adding penalty on hidden layer weights magnitude 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 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 phenomena was observed during experiments presented in this project. Applying weight decay to convolutional layers resulted in numerous dead filters which worsened classifiers 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, in our case to EM images.

However EM images are generally quite noisy while some examples are very noisy.

Secondly, noise can be added to weights. This technique is primarily used in recurrent neural networks [32].

Thirdly, most data sets have some mistakes in the annotated labels. As shown in section 4.6, it can be very costly to maximize $\log p(y|x)$ when label y is wrong. 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. Label smoothing operation is summarized in equation 2.

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

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

5.3 Dropout

Dropout [33] 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 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 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 optimal number of training epochs. Idea is to split training data into train set used to train the model and validation set, which is not used in training process. Errors of both sets are monitored at each epoch. Training should be stopped when validation set performance has not improved for some selected number of epochs. This prevents overfitting and improves generalization. Validation set error is more suitable than training set error because training error decreases steadily over time as model starts to overfit while validation error begins to rise again. In this project early stopping is used to determine number of training epochs, that is later fixed during k-fold cross validation.

5.5 Data augmentation

As was already mentioned in section 4.7 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 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 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 . Choice of transformations is highly application dependent. For example, optical recognition models must recognize 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 range $(-180, 180)$ degrees with spline interpolation
- Shear transformation
- Vertical shift in range $(-10, 10)$ percent of total height
- Horizontal shift in range $(-10, 10)$ percent of total width
- Zoom
- 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 8 show an example of data augmentation. Thus a model never sees two exactly same examples during training.

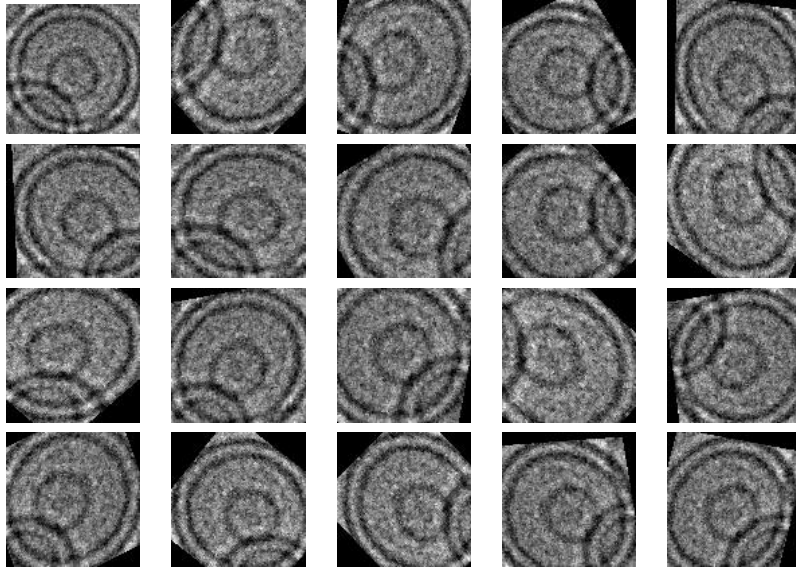


Figure 8: Example of data augmentation. Upper left is an original image, all other images are a result of random transformations described in this section.

6 Review of deep learning software tools

A wide variety of deep learning software tools is available to the public. Most of them are open-source and are distributed under licenses that allow commercial use. Development of deep learning frameworks is driven by Universities and leading IT companies such as Google and Microsoft with extensive help of the community. For example Theano [34] by 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 [35] by UC Berkeley. Google released TensorFlow [36] software library in November 2015. Microsoft made its CNTK [cntk] tool available to the public in January 2016.

Mentioned above software tools together with some other are summarized in table 4.

Four deep learning libraries: TensorFlow, Caffe, Torch [37] and CNTK have been compared and benchmarked in [38]. 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 outperforms other.” Their tests of CNN models showed that Caffe and Tensorflow performed best on CPU with 4 and 16 threads respectively. On GPU Caffe, Tensorflow and CNTK were best depending on mini-batch size and GPU type.

When selecting deep learning framework one should also consider 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 programmers community. In order to estimate popularity of a deep learning library I gathered statistics from Stackoverflow [39] and GitHub [40]. Stackoverflow is a popular on-line programming question and answer community. Github is one of the largest web-based code repository hosting service. Source code of all deep learning libraries mentioned in this project is hosted on GitHub at the time of writing. Number of questions on Stackoverflow related to each deep learning library and number of subscribers to the corresponding code repositories are compared on figure 9. While according to [38] there is no clear performance leader, programmers community seems to have already selected its favorite deep learning library. There are almost four times more questions about TensorFlow than about its runner ups 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 [38] seems to be almost ignored by the community, as of October 2016 there were only 15 questions tagged CNTK on Stackoverflow.

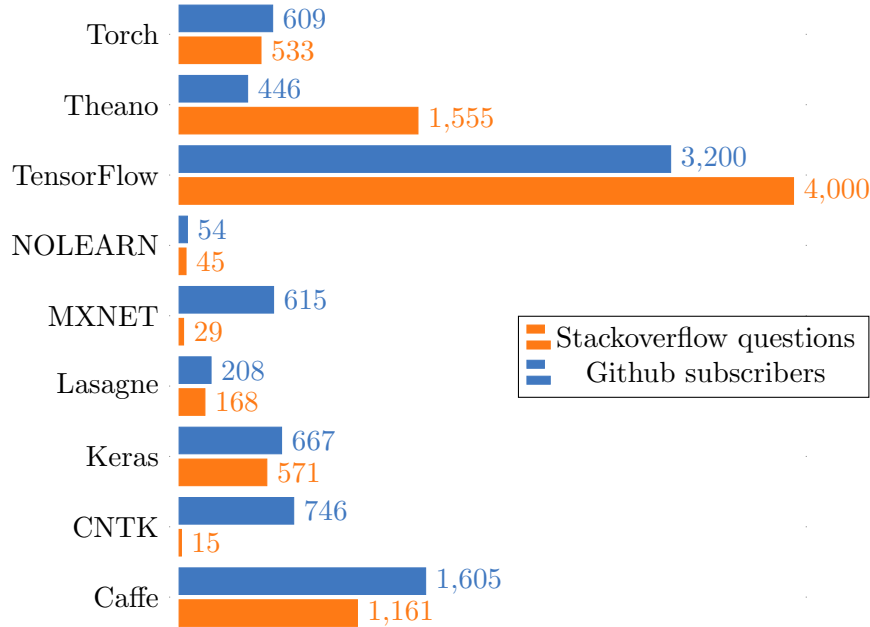


Figure 9: Popularity of deep learning frameworks as of October 2016.

Licensing is another important issue. All presented here deep learning libraries are distributed under one of the following open-source licenses [41]: MIT, Apache and BSD. The only exception is CNTK which has also 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 user friendly API. Lasagne and Keras [42] 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 PI 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.

It is worth to mention 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 well established tools.

When it comes to programming language APIs, Python is a primary choice of majority of deep learning tools. Nearly all tools mentioned in this report have well documented extensive Python API. Linux operating system is the preferable operating system for deep learning environments. As of October 2016 not all deep learning tools provide versions for Windows while Linux is supported by all of them. It might be an issue for Vironova as its software is developed for Windows operating system. There is a number of machine learning frameworks available for Microsoft .NET framework and one of the most promising is Accord.NET [43]. 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. Currently Tensorflow has no version for Windows however Keras allows to switch between Tensorflow and Keras backend, so all experiments can be replicated on a Windows machine after switching to Theano backend.

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

Table 4: Comparison of characteristics of deep learning frameworks

	Linux		Windows		Mac OS		Language bindings		License
	CPU	GPU	CPU	GPU	CPU	GPU	Python	C++	
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

CNN can be configured according to a wide variety of architectures. There is no general rule that tells how to configure a CNN for specific task but there are some best practices and general recommendations. Probably the most influential CNN architectures are VGG [44], AlexNet [5] and GoogleNet [45].

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

1. **Convolutions.** Here several convolutions are performed in parallel to produce a set of linear activations. Convolutions can have filters of different size, different padding strategies and stride values. Stride value specifies how many pixels is the filter moved at a time. Stride values greater than 1 result in output volumes spatially. Different padding modes affect size of the output. The most common strategies are *valid* and *same*. *Valid* mode implies that convolution is performed only at those positions where filter is within image bounds. This results in reduced size of layer output compared to layer input. *Same* mode implies zero padding in order to preserve input size.
2. **Nonlinearity.** At this stage linear activations produced on 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** is used to modify the output of the layer further. A pooling function replaces the output of a layer at certain position with a summary statistics of the surrounding region. Size of this region is a subject of design decision. In practice 2×2 pooling is used. The most popular summary statistics is *max*, however other such as average are also possible. Pooling is usually followed by dropout for regularization purposes. Dropout is discussed in details in section 5.3.

Talking about classification problems, a typical CNN is usually flattened at the end and a number of fully connected layers go to the final layer. Final layer has softmax activation and number of nodes that is equal to the number of classes.

Networks that are flattened and have fully connected layers require input of fixed size, they can not handle images of different size. If it is required to build a model that can process images of arbitrary size then fully convolutional neural network (FCNN) is a possible solution. FCNN do not require input to have fixed size and can be trained on images of variable size. Fully convolutional means in this context that network does not have any fully connected layers, only convolutional. Flattening operation is replaced with 1×1 convolution which leads to dimension reductionality. FCNN proved to be especially effective in semantic segmentation tasks [7].

Springenberg et al. shows in [18] that max-pooling layers can successfully be replaced with convolutional layers with corresponding stride value. Stride value specifies how many pixels is the filter moved at a time. Stride values greater than 1 result in output volumes spatially.

Five are tested in this project. I start with the simplest configuration possible — one convolutional layer followed by two fully connected layers and gradually add more and more convolutional layers. Let us call networks LipNet- x (figure 10), where x stands for number of convolutional layers.

- **LipNet-1** is the simplest possible configuration, a starting point for 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 ReLU activation function, *same* padding mode, 3×3 filter size, 32 channels and stride equal to 1. Input size is fixed to 28×28 .
- **LipNet-4c** is inspired by ideas of fully convolutional networks and proposal to replace max-pooling with convolution with stride as in [18]. So max-pooling layers are replace 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 .
- **LipNet-6** is a modification of LipNet-4 with additional convolutional block. Additional convolution with stride requires bigger input size so images are resized to 32×32 .

Classes are one-hot encoded [46] as demonstrated in table 5.

Table 5: Class encodings

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

Thus 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 [47] activation function guarantees that all node values are in $[0, 1]$ and sum up to 1.

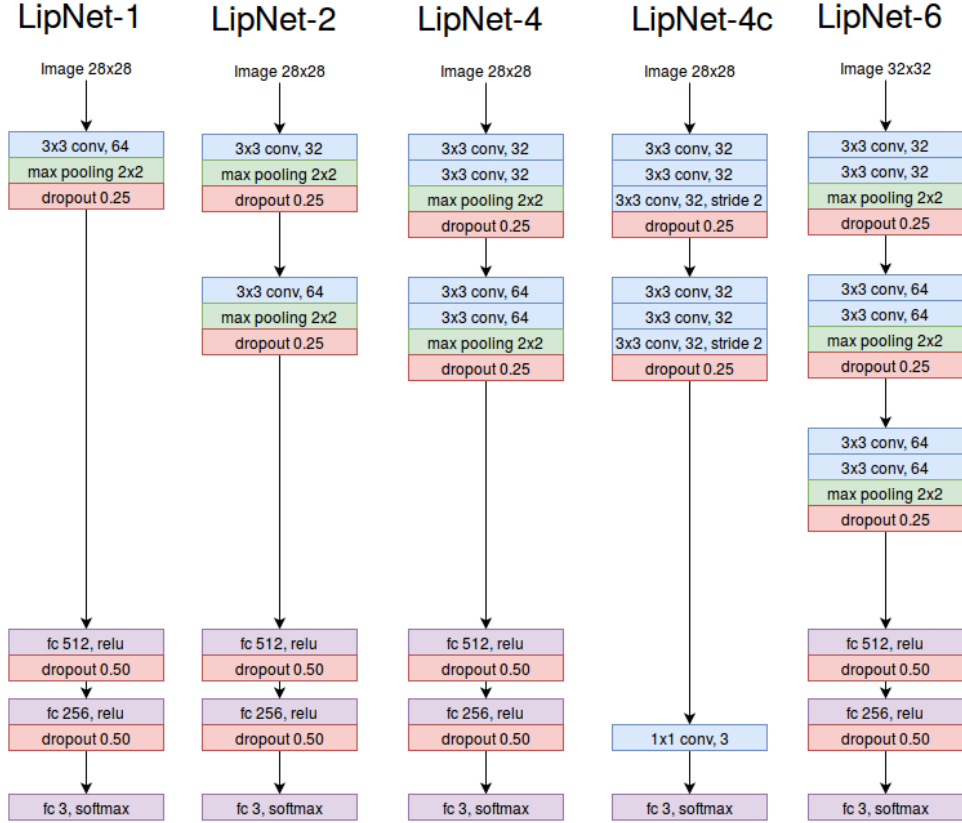


Figure 10: Network architectures.

Figure 11 shows an example of visualized convolutional filters of trained LipNet-4 model. Filters are visualized with seismic color map in order to see clearly negative and positive values: blue color corresponds to values less than zero; red — greater than zero and finally white color corresponds to zero values.

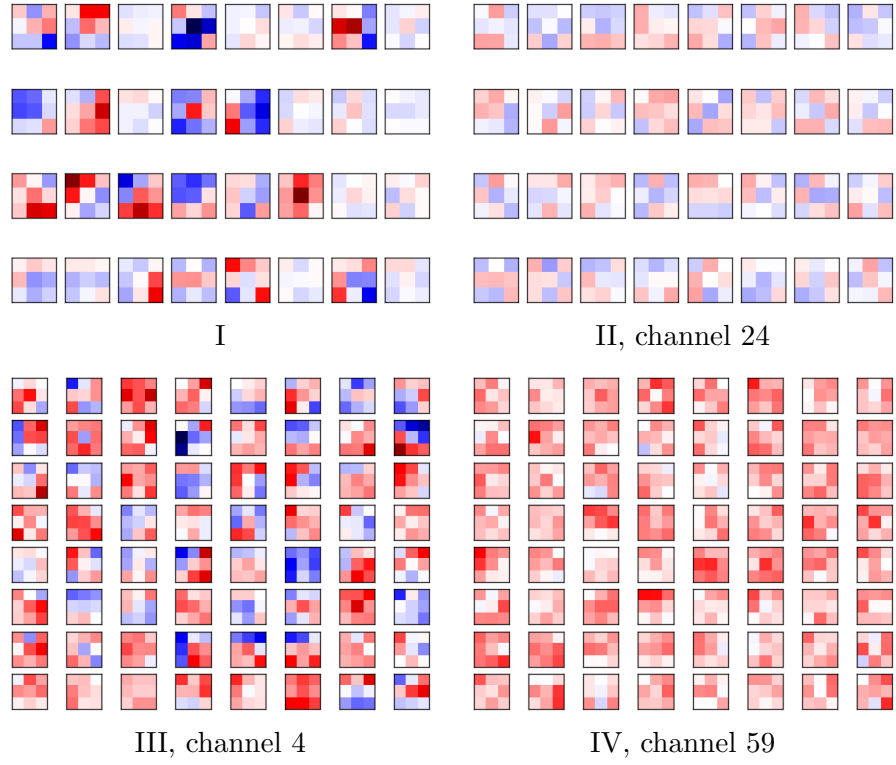


Figure 11: Visualization of convolution kernels

Figure 12 shows example output of different convolutional layers of LipNet-4 model.

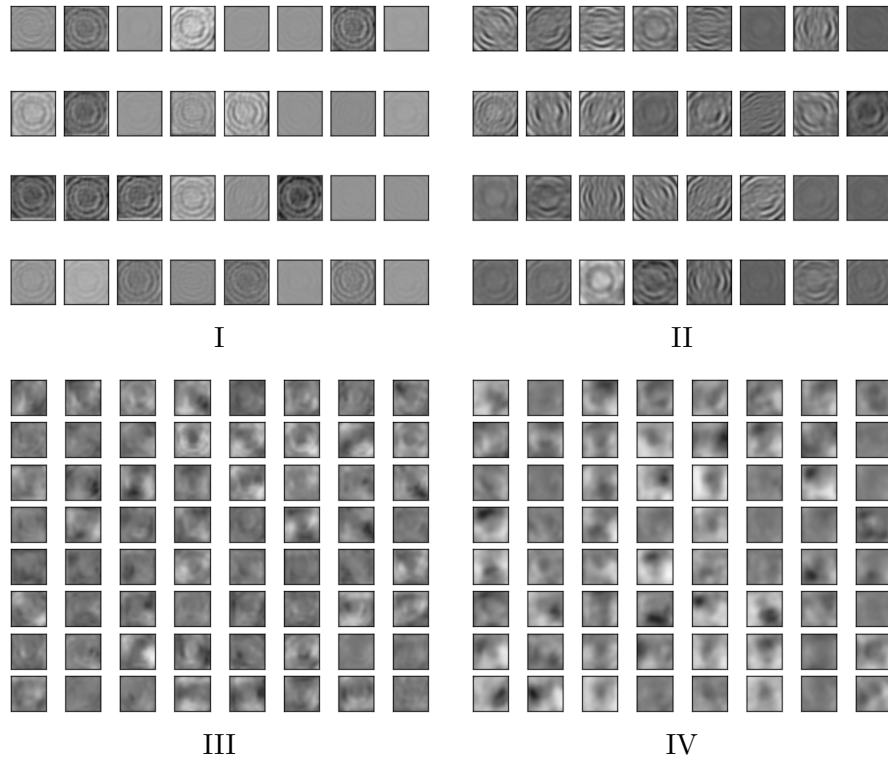
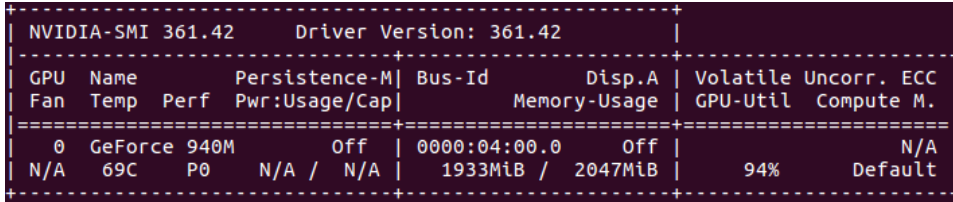


Figure 12: Visualization of convolutional layers outputs

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 GeForce 940M video card. GPU utilization statistics as on figure 13 is invoked using `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 13: GPU utilization statistics.

All models are evaluated using 5-fold cross-validation [48] technique. The data is split into 5 folds using stratified sampling which means that class proportions are preserved. One fold is used as test set while four other as training set and the experiment is repeated five times, each time with new fold as a test set.

Training sets are balanced using oversampling, test sets are left unbalanced. During training training data is augmented as described in 5.5. Data augmentation transformations are applied randomly to each training example, so the probability to see 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 default value of 32 images. If validation set is used then one can see that at the end of some epochs validation error is less than test error. This happens because 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 data augmentation techniques is performed in the following way. All data is divided in five folds with preserved class proportion and each of them serves as a test set in turn while other four as a training set. All training sets are balanced using oversampling, test sets are not balanced.

Model architecture is Lipnet-4, images are without padding and masks are no applied.

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

y axis of figure 14 represents five-fold average normalized true positive value for each respective class.

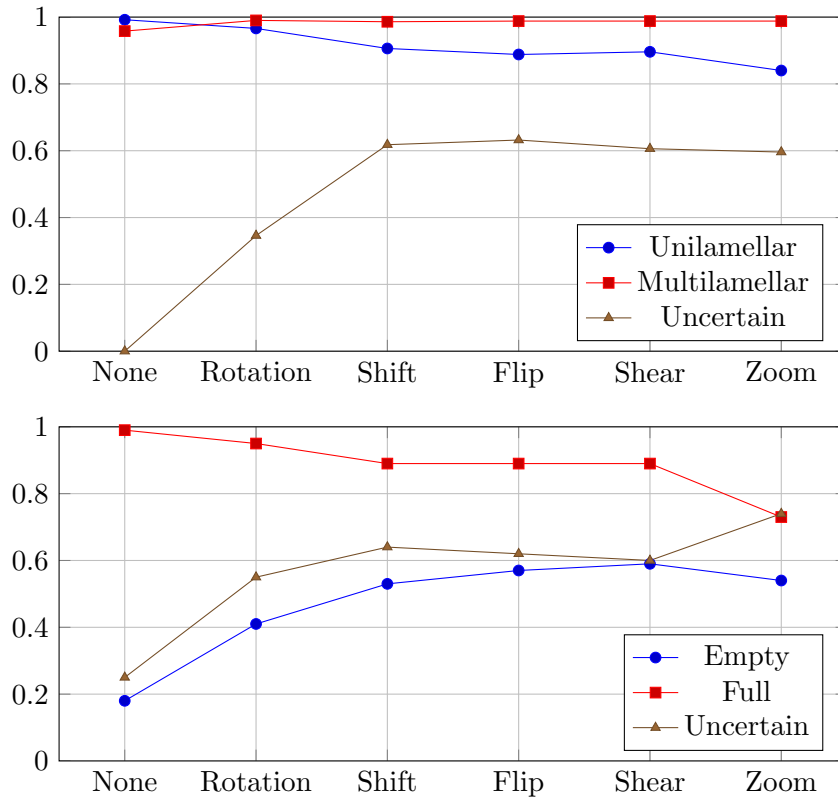


Figure 14: Effect of combining augmentation techniques successively on average normalized true positive value. Top — Lamellarity problem, bottom — Encapsulation.

In **lamellarity** problem the clear beneficiary of data augmentation is *uncertain* class. Majority class *unilamellar* and the runner up *unilamellar* are being classified with almost 100% accuracy even without any augmentation of training data. However *uncertain* class is totally misclassified when no augmentation is performed. Rotation itself raises true positive value of *uncertain* to nearly 40% which makes sense. Taking into account very small number (84) of examples of this class it is not an unlikely scenario when

test set contains patterns that the model has not seen during training. For example train and test set contain patterns that are similar but differently oriented. In such case rotation improves result a lot which is the case in this experiment. Adding shift transformation brings further improvement to recognition of *uncertain* class. Adding flip, shear and zoom transformations do not have any positive effect on performance. In fact, more aggressive data augmentation worsens model’s ability to generalize and performance on *multilamellar* and *uncertain* classes drops a bit when all transformations are used.

Encapsulation problem is even more unbalanced than lamellarity so without any data augmentation almost all examples are classified as *full*. Data augmentation makes clear improvement in performance, true positive values of *empty* and *uncertain* classes are growing 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 trend is quite similar to the one observed in lamellarity problem except the last point. Adding zoom transformation improves model’s ability to recognize *uncertain* examples while worsens performance on *full* examples. This is an unexpected result, since it would be natural to expect general performance improvement when zoom transformation is used. Zoom encourages model to focus more on the core of the liposome which is the most relevant part in encapsulation problem.

In order to analyze effect of zoom transformation in more details additional experiment has been performed. This time the data is augmented using only zoom transformation. Results are presented in table 6. 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 but that would require to run 120 experiments. Unfortunately due to hardware and time limitations it is not feasible to perform such number of experiments.

8.3 Evaluation of different CNN models

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

Results for Lamellarity problem are shown on figure15. 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 unilamellars. All other models demonstrated more or less similar results in recognition of unilamellar and multilamellar liposomes. LipNet-4 and LipNet-6 were best to recognize particles if uncertain class clearly outperforming other models.

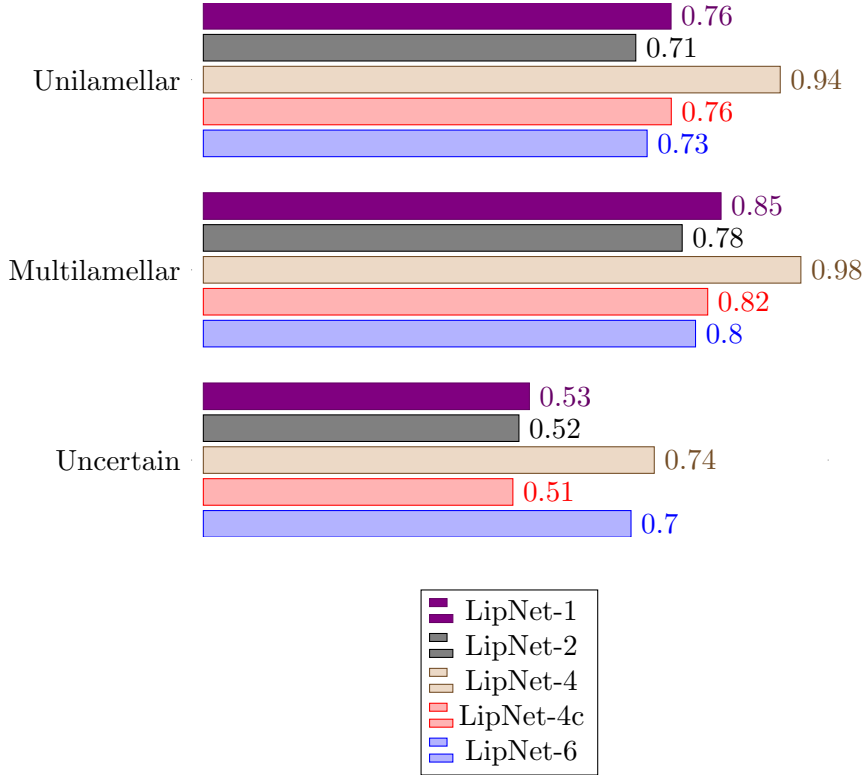


Figure 15: Lamellarity problem. F_1 scores of different CNN models.

Results for Encapsulation problem are shown on figure 16. Unlike Lamellarity problem there is no clear leader, no model outperforms other for all classes. Simpler models like LipNet-1 and LipNet-2 are superior when it comes to Empty class but they are marginally worse than LipNet-4 and LipNet-4c in recognizing Full class. LipNet-4 and LipNet-4c are best to recognize liposomes of Uncertain class but LipNet-1 result is very close. The only clear pattern is that LipNet-6 ability to generalize is the worst among tested models. Probably it has too many parameters and overfit which leads to poor generalization.

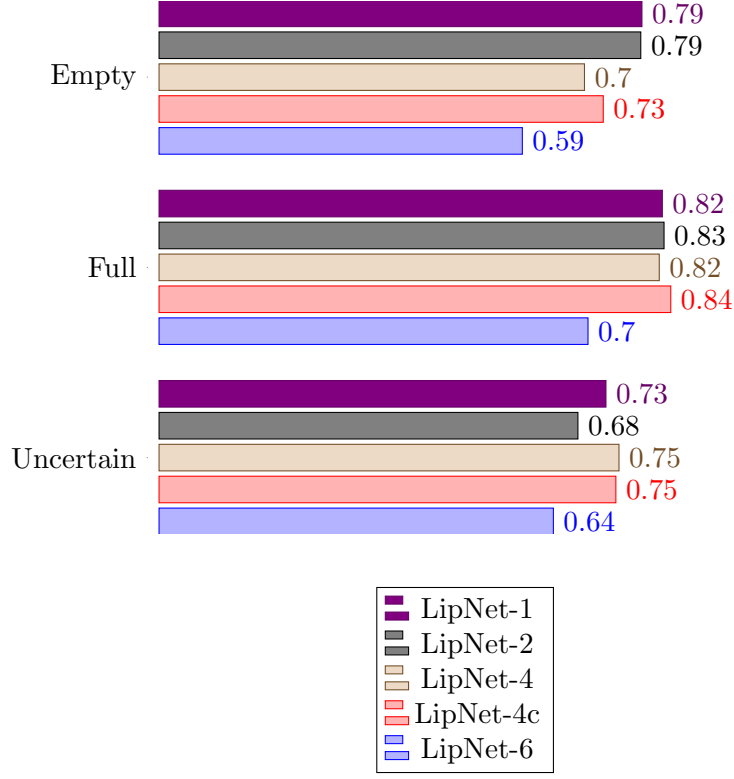


Figure 16: Encapsulation problem. F_1 scores of different CNN models.

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 SVM models.

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

Training sets are not balanced, instead class weights are adjusted inversely proportional to class frequencies in the input data.

Tables 7 and 8 present average normalized confusion matrices for encapsulation and lamellarity problems respectively. Data sets are so unbalanced that total accuracy coincides with true positive rate of majority classes, which are *empty* and *unilamellar* respectively. Tables ?? and ?? present performance statistics for all 5 folds, average values are shown on figure 19.

Table 7: 5-fold average normalized confusion matrix, Encapsulation problem, SVM.

	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, Lamellarity problem, SVM.

	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 is padded 50 pixels in each direction and corresponding particle masks are available. So it is a matter of design decision whether to include padding or not as well as if masks are to be applied. Three alternatives have been tested:

- Input images with padding. All images are padded 50 pixels in each direction and include information about particles surrounding.
- Cropped images. All images are cropped to a minimum size that allows to contain particles.
- Cropped and masked. All images are cropped and corresponding masks are applied. This means that each image contains information only about particular particle and nothing else.

Comparison is performed in terms of F_1 score (averaged across 5 folds) of LipNet-4 network trained on images in different modes.

Results of experiments showed that including padding to input images leads to worse performance in both problems for all classes. The best performance is achieved when input images are cropped and masked. Detailed comparison is shown in figures 17 and 18.

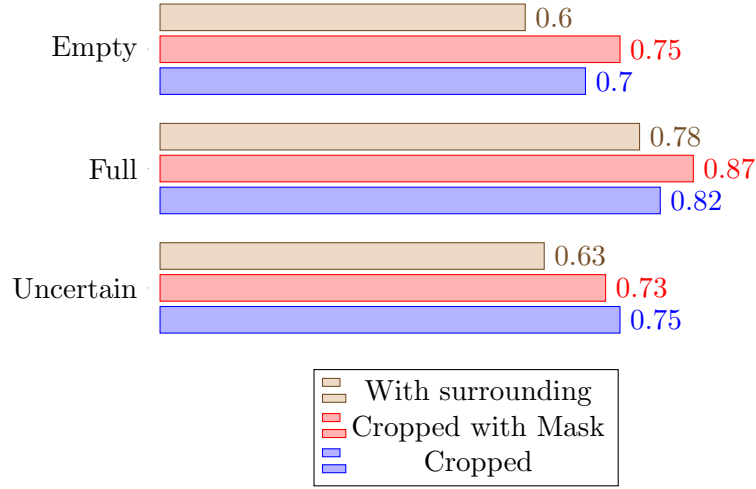


Figure 17: Encapsulation problem. F_1 score of Lipnet-4 network with different input image modes: with surrounding, cropped, cropped and masked.

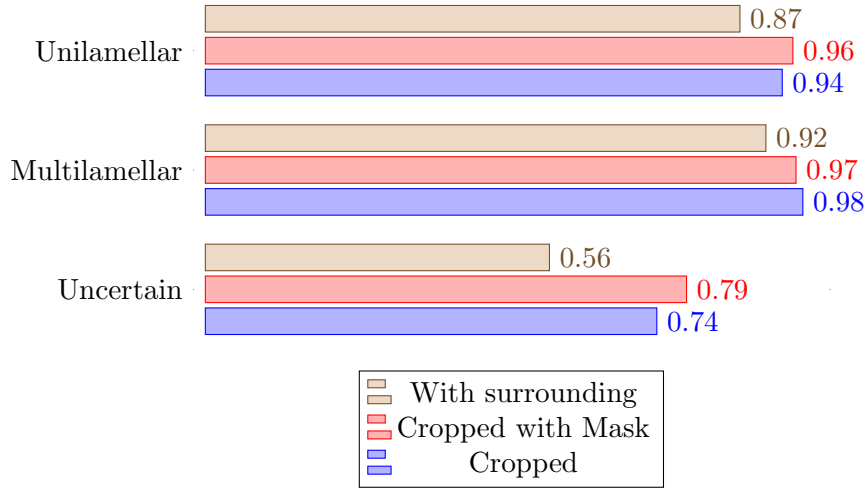


Figure 18: Lamellarity problem. F_1 score of Lipnet-4 network with different input image modes: with surrounding, cropped, cropped and masked.

9 Benchmarking

In this section selected CNN models are benchmarked against SVM. For each problem one CNN model that performed best in previous experiments has been selected for comparison to SVM. For Lamellarity problem LipNet-4 model has been selected, for encapsulation problem — LipNet-2. In both cases data sets are balanced with oversampling, input images are cropped, masked and augmented. All metrics are presented in figure 19.

Unilamellar and multilamellar liposomes are recognized by both LipNet-4 and SVM with reasonably high results. Convolutional neural network slightly outperforms SVM in this case. In particular CNN is better in predicting that a particle is not unilamellar, CNNs negative predicted value is 0.7 against 0.5 of SVM. On the other hands results are opposite when it comes to uncertain class of Lamellarity problem. Here SVM clearly outperforms CNN in terms of sensitivity: SVM identifies nearly all particles of uncertain class as such while CNN does so only in 70% of cases. However both methods yield many false positive predictions of uncertain class which results in poor precision. Even though CNN has almost double precision ans SVM it is still very poor being slightly above 10%. Mainly unilamellar liposomes are confused with uncertain by both CNN and SVM. Even if a small fraction of unilamellars are falsely identified as uncertain it is enough to bring down precision because data sets are heavily unbalanced. Averaged across 5 folds unnormalised confusion matrix 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

Particles from Encapsulation problem are generally better classified by SVM. In this case SVM outperforms CNN for all three classes however the difference is not very big, LipNet-2 is still demonstrating a reasonable level of performance. Positive predicted value of empty and uncertain class demonstrates the same patter as uncertain class from Lamellarity problem. Approximately 10% respectively 15% of positive empty and uncertain predictions are correct because some full liposomes are falsely classified as either empty or uncertain. Once again, even is a small fraction of full liposomes are misclassified this reduces precision of minority classes drastically. Another notable result is low negative predicted value of full class. It means that classifiers yield more false negative predictions than true negative. On

the other hand positive predicted value of both CNN and SVM for full class is almost 1 which means that there are hardly any empty and uncertain particles that are falsely classified as full. In other words it means that if a classifier says that a particle is full that it is mostly likely to be full, however when it says the opposite it is most likely to false. Averaged across 5 folds unnormalised confusion matrix is presented in table 10.

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

	Empty	Full	Uncertain
Empty 2 24	1	7	
Full	272	3929	660
Uncertain	30	9	61

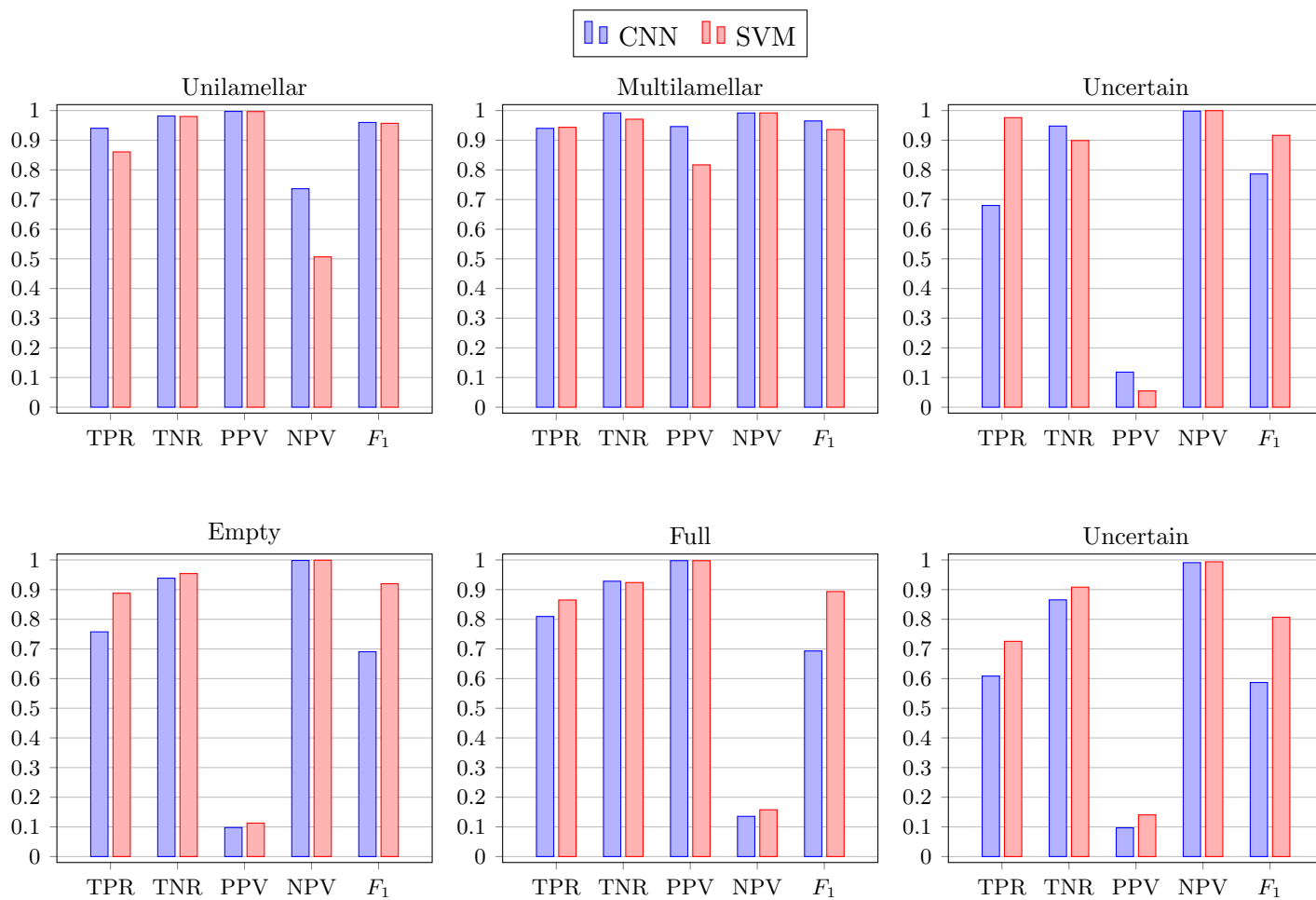


Figure 19: Comparison of SVM and CNN (best viewed in color). Input images are without surrounding and masks are not applied.

10 Discussion

Comparison of different input image modes showed that the best performance is achieved when input images contain only particles. Including surrounding worsens classifiers performance. The result is intuitive and expected. By excluding all information irrelevant to the particle we make images less noisy and let classifier concentrate on the particle. This has though one drawback: masking implies reading mask files and performing multiplications which requires additional time. Experiments showed that in Encapsulation problem masking improved 5-fold average F_1 score of Full and Empty classes by 6% and 7% respectively and in Lamellarity problem F_1 score of Unilamellar and Uncertain classes went up 2% and 7% respectively. On the other hand F_1 score of Uncertain class in Encapsulation problem went 2% down after applying masking but I still consider masking worth applying given increase of performance for other classes.

Comparison of SVM and CNN classifiers has shown that Lamellarity problem can be solved with both methods with approximately same performance. SVM achieved better performance for Encapsulation problem but CNN still demonstrated reasonable results. Despite such results CNN may still be preferred due to its ability to generalize to almost any classification problem. SVM performed well due to careful choice of image feature 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 such type of expert knowledge. Ability to learn descriptive image features in non-trivial domains is the cornerstone of convolutional neural networks. Given a data set of reasonable size CNN model can be relatively easy retrained and adopted for almost any kind of classification problem. 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 to production environment even if it is changing rapidly.

Deconvolutional neural networks is a powerful visualization tool that can help to understand what kind of image feature has the CNN learned. Unfortunately due to time and software limitations no experiments with deconvolutional networks have been performed. According to [49] 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. It is hard to say what kind of features have LipNet models learned by just looking on its filters like on figure 11. 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 be hardly 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.

10.1 Limitations

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

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

Due to hardware limitations it was impossible to test deeper networks with many layers but is doubtfully that deeper network would have better generalization as risk for overfitting increases as network structures become more complicated.

11 Conclusion

Data sets imbalance has proven to be a major problem. Unless imbalance problem is addressed classifiers are heavily biased towards majority classes and almost ignore underrepresented classes. It has been shown that over-sampling combined with data augmentation can mitigate imbalance problem and lead to reasonable level of generalization. Most common image augmentation techniques like rotation, shift, flop, shear and zoom have been compared. Rotation and shift were the most helpful techniques, however it is context-dependent. For example, zoom technique was almost as helpful as rotation in encapsulation problem, while rotation was far more effective than zoom in lamellarity problem.

Deconvolutional neural networks have been discussed and it can be concluded that is a promising method that can help to select features that would describe images from different problems in the best way.

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 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 nearest 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.

Speaking about optimization methods, both ADAM and SGD performed well and no method outperformed another one.

Convolutional neural networks trained on cropped and masked images have 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.

Five different network architectures have been tested and compared. A network with two convolutional blocks each of which has two convolutional layers performed best in Lamellarity problem. A simpler one, with two convolutional blocks and one convolutional layer per block was best for Encapsulation problem.

Overall performance shown by CNN was reasonable. CNN performed approximately on the same level as SVM in lamellarity problem and was slightly worse than SVM in Encapsulation problem. Convolutional neural networks offer greater flexibility and generalization compared to SVM because CNN method does not require expert knowledge in order to choose image features. However such flexibility comes at cost of greater computational overhead.

12 Future work

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

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

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=XpUjsqD71FUC>.
- [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] 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>.
- [19] 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>.

- [20] 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.
- [21] 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>.
- [22] 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](http://dx.doi.org/10.1016/S0034-4257(97)00083-7). URL: <http://www.sciencedirect.com/science/article/pii/S0034425797000837>.
- [23] 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.
- [24] Max Reeves Sergii Gryshkevych Derrick Alabi. *Shelter Animal Outcomes*. Uppsala University, Department of Information Technology. 2016.
- [25] 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>.
- [26] 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.
- [27] 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>.
- [28] 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>.

- [29] Carolina Wählby Omer Ishaq Vladimir Curic. “Training of machine learning methods for fluorescent spot detection”. In: (Apr. 2016).
- [30] 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>.
- [31] 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>.
- [32] 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.
- [33] 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>.
- [34] 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>.
- [35] Yangqing Jia et al. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: *arXiv preprint arXiv:1408.5093* (2014).
- [36] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <http://tensorflow.org/>.
- [37] Ronan Collobert, Koray Kavukcuoglu, and Clement Farabet. “Torch7: A Matlab-like Environment for Machine Learning”. In: *BigLearn, NIPS Workshop*. 2011.
- [38] 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>.
- [39] Lena Mamykina et al. “Design Lessons from the Fastest Q&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>.

- [40] 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>.
- [41] Lawrence Rosen. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004. ISBN: 0131487876.
- [42] Chollet Francois. *Keras*. 2015. URL: <https://github.com/fchollet/keras>.
- [43] C. R. Souza. *The Accord.NET Framework*. <http://accord-framework.net>. Dec. 2014.
- [44] 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>.
- [45] Christian Szegedy et al. “Going Deeper with Convolutions”. In: *CoRR* abs/1409.4842 (2014). URL: <http://arxiv.org/abs/1409.4842>.
- [46] 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>.
- [47] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.
- [48] 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>.
- [49] 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>.