Master's Thesis

Peptide Detection Using Convolutional Neural Network

Zaid Ur Rehman

First Examiner: Prof. Dr. Thomas Brox
Second Examiner: Prof. Dr. Oliver Schilling

Albert-Ludwigs-University Freiburg
Faculty of Engineering
Department of Computer Science
Chair for Pattern Recognition and Image Processing
&
Center of Biochemistry and Molecular Cell Research
Institute of Molecular Medicine and Cell Research

February 12th, 2018

Writing period

15. 08. 2017 – 12. 02. 2018

Examiners

Prof. Dr. Thomas Brox, Prof. Dr. Oliver Schilling

Advisers

Dr. Thorsten Falk, Dr. Lars Nilse

Declaration

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Place, Date	Signature

Abstract

Beside genome sequencing, quantitative proteomics is one of the most useful tools in modern biological and medical research. Mass Spectrometry (MS) allows the complete molecular characterization of proteomic samples which can be treated as images. Convolutional Neural Networks (CNN) are widely used for visual recognition tasks. This project aimed at processing MS spectra images for peptide feature detection using CNN's ability for solving localization and semantic segmentation problems.

Zusammenfassung

Neben der Genomsequenzierung zählt Quantitative Proteomik zu einer der wichtigsten Methoden in der modernen biologischen und medizinischen Forschung. Massenspektrometrie ermöglicht die vollständige Charakterisierung von Proteom-Proben. Die resultierenden Spektraldaten können auch als Bilddaten interpretiert werden. Convolutionale Neurale Netzwerke (CNN) sind ein weit verbreiteter Ansatz in der Bilderkennung. Das Ziel des Projektes ist es, mit Hilfe von CNNs Peptid Features in MS Spektraldaten zu identifizieren.

Contents

List of Figures

List of Tables

List of Algorithms

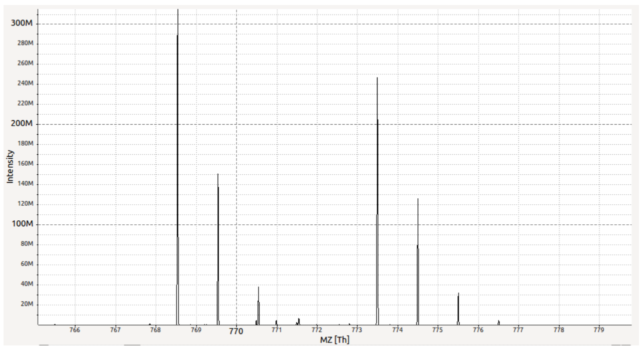
1 Introduction

Proteomics refers to the study of proteins in biology. Proteins are the end result of gene encoding and are responsible for performing a vast array of functions in organisms. They consist of long chains of amino acids joined together with covalent peptide bonds. As an arbitrary benchmark, any molecule with up to 50 amino acids chained together are referred to as peptides whereas molecules with more than 50 amino acids are called proteins. Proteins can be broken down into smaller chains of peptides through enzymatic digestion e.g. with Trypsin which cleaves the peptide chain at the carboxyl side of the amino acids Lysine or Arginine, except when either is followed by Proline [1]. The resulting peptide mixture can be characteristically analyzed by Mass Spectrometry (MS). MS of whole proteins is less sensitive than peptide MS and the mass of the intact protein by itself is insufficient for identification of their sequence [2].

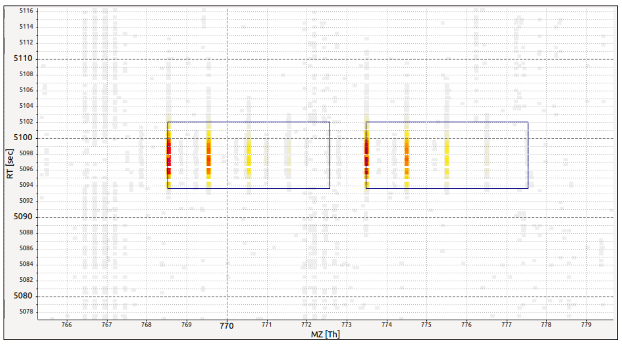
Liquid Chromatography (LC) is used to reduce complexity of the pepide mix before they are ionized by electrospray ionization and fed into a mass spectrometer for analysis. Mass spectrometers measure mass-to-charge ratio (*m/z*, also abbreviated as MZ in this thesis), measured in Thomson (Th) which is a ratio of mass in Daltons (Da) and charge of the ion. The electrospray ionization process generates ions with multiple charges such that the observed *m/z* value has to be multiplied by charge z to calculate the molecular weight of a particular peptide [3]. A mass spectrum is a recording of signal intensity of ions at each value of *m/z* scale, as shown in Figure 1a. Each peptide feature comprises of isotope clusters of peaks, and such peaks are separated by 1 Da which is caused by the presence of C^{13} isotope instead of C^{12} atom. Peptide fragments can be identified due to this predictable isotopic peak pattern, an example of which is shown in Figure 1c. The retention time (RT), also called elution time, measured in seconds, is calculated as the time taken by a solute to pass through the chromatography column. A 3D plot of observed intensities can represent a LC-MS experiment, as in Figure 1 where two peptide features are visible.

Tandem MS (LS-MS/MS) is the technique where a particular peptide ion is isolated and fragmented by collisions with an inert gas, and a mass spectrum of the fragments is obtained. By identifying the complete set of peptides in the spectra of LC-MS or LC-MS/MS experiments, parent proteins can be identified. However, all peptides are not detectable. This might occur due to many reasons such as incomplete digestion, small size, poor binding or elution from the type of LC column used, the limited detectable mass range of the mass spectrometer, bias toward detecting peptides with an intense MS signal in mixtures, the charge prior to ionization, and non-covalent interactions between peptides in the gas phase while in the mass spectrometer [5]. Moreover, there is no method available that can identify and quantify the components of a complex protein sample in a simple single-step operation [2]. MS spectra are stored for matching against protein sequence databases for possible peptide matches. This is achieved by correlating the observed spectra of individual peptides with the predicted MS/MS spectra of the amino acid sequences derived from protein databases. The other peptide identification method is the *de novo* sequencing where the amino acid sequence of peptide is derived from mass spectrometry and peptide fragmentation data without using databases.

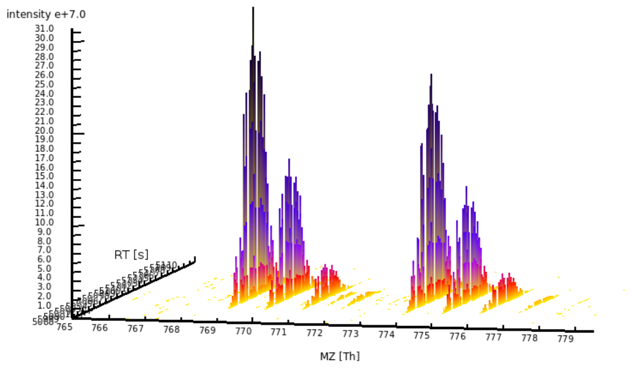
Peptide detection in 3D MS Spectra is a challenging task [6]. Higher intensity peaks profiles stand out in the spectra and can easily be quantified. However, the signal-to-noise ratio drops for regions with low-intensity peaks due to which the characteristic peak patterns are hard to identify. A graphical comparison of high and low intensity features is shown in Figure 2. Peptide features retain a gaussian-like elution profile along RT axis which is also affected due to noise and looses its distinct shape. At even lower intensities, the background noise dominates the m/z positions of isotopic peaks. However, the relative shifts along m/z axis between isotopic peaks remain preserved due to high accuracy and precision of modern mass spectrometers. These relative shifts provide much clearer evidence of peptide feature than the elution profile along RT axis. This project aimed at solving the peptide detection problem by treating it as a semantic image segmentation problem. A Convolutional Neural Network (CNN) was trained to learn the characteristic isotopic peak pattern of peptide features in processed images of MS spectra. The images were annotated with labels for valid peptides by the classical feature detection algorithm 'FeatureFinderMultiplex'.



(a) Intensity vs MZ plot showing a single spectrum read at RT = 5097.47s. Single mass traces are vertical lines showing the intensity at a particular MZ value. The traces are joined together through spline interpolation and shown here as gaussian-like curves.

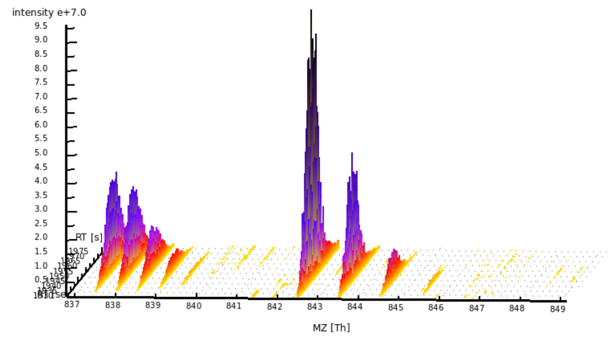


(b) RT vs MZ plot where color of the spectra represent intensities. Valid peptides are enclosed in rectangles and each peptide feature has five different isotope compositions here.

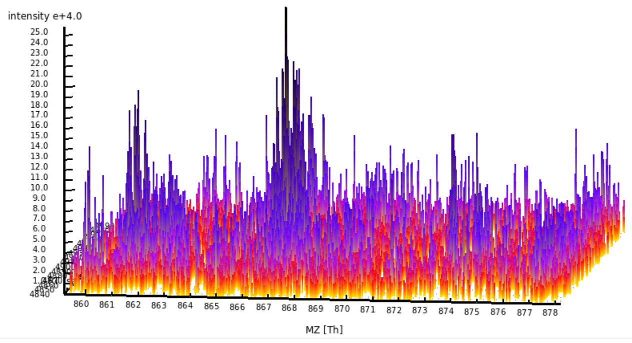


(c) Two peptide features shown in 3D. The five different isotope compositions have decresing intensities, the fifth composition is hardly visible in this graphic.

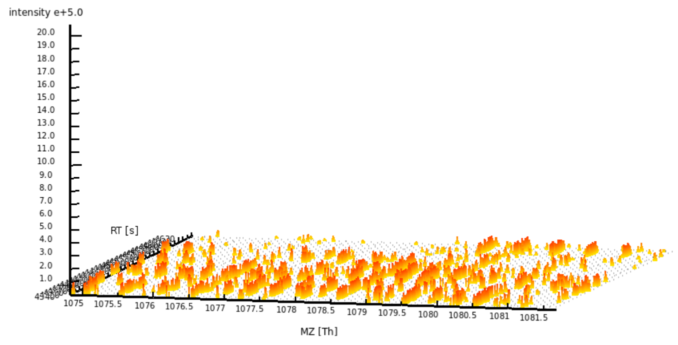
Figure 1: MS Spectra Plots for Peptide Feature Detection. These images are screenshots of TOPPView [4].



(a) High intensity features



(b) Low intensity features



(c) No feature / noise

Figure 2: Higher intensity peptide features, such as in (a), are easier to detect than the lower intensity ones as in (b). (c) represents a small space with even lower intensities where no peptides can be identified.

2 Background

2.1 Mass Spectrometry Data Analysis

LC-MS enables quantitation and identification of proteins and peptides using a wide range of experimental protocols, algorithms and statistical models to analyze the data. Many such tools used for quantitative proteomics are included in an open-source software platform OpenMS [7]. FeatureFinderMultiplex and MSSimulator, further described in subsections 2.1.1 and 2.1.2 respectively, are two OpenMS tools which were used for data generation during this project.

Furthermore, the library pyOpenMS [8], a Python interface for accessing data structures and algorithms in OpenMS core libraries, were used for data pre-processing and converting MS spectra to images for network training. TOPPView is another tool available in OpenMS and was used to visualize MS spectra and known peptide features [4]. The mzML file format is a open source file format for mass spectrometer output files and consists of spectra and some meta-data such as operational parameters, MS instrumentation, etc. [9]. Known or detected peptides features are saved as featureXML files which store a bounding box for the MS spectrum of every valid peptide when viewed in TOPPView, as seen in figure Figure 1b. All figures in this thesis representing a mass trace, spectrum or a bounding box are snapshots of the TOPPView tool.

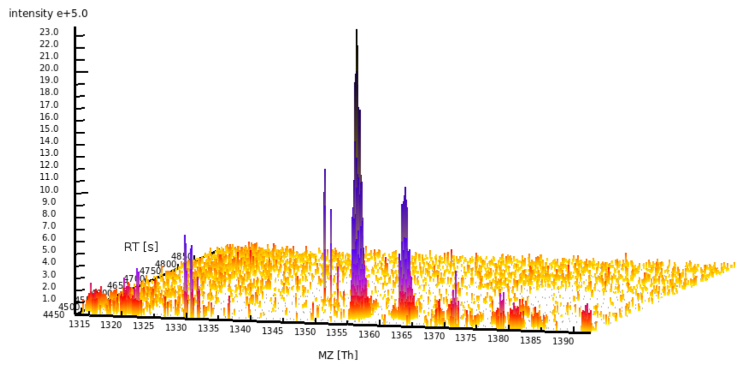
2.1.1 FeatureFinderMultiplex

Peptides of a proteomic samples can be tagged before being studied under the LC-MS experiment. The tagging involves adding a chemical label by dimethylation of primary amines. The dimethyl group gives a 'light' (+28 Da) or a 'heavy' (+36 Da) label. Using this technique, half of the proteomic sample can be labelled 'light' and the other half 'heavy' and mixed together. Consequently the LC-MS data of the mix should contain peptides appearing in pairs in each spectrum with an m/z offset depending on the charge they carry. FeatureFinderMultiplex (FFM) is a tool designed to detect such peptide features. It can also be used to detect single peptides, pairs, triplets etc. FFM does not rely on any information about the peptide sequence and performs the feature detection in three parts: filtering, clustering and linear fitting. Given an intensity threshold, MS spectra are filtered out based on their and their neighbours' intensity profiles, the m/z offset with neighbouring peaks and intensity ratios as compared to a theoretic model. Detected pair intensities are assigned to a cluster of similar pair intensities. Finally, relative amounts of differently labelled peptides can be determined using linear regression based on the linear correlation of intensity pairs.

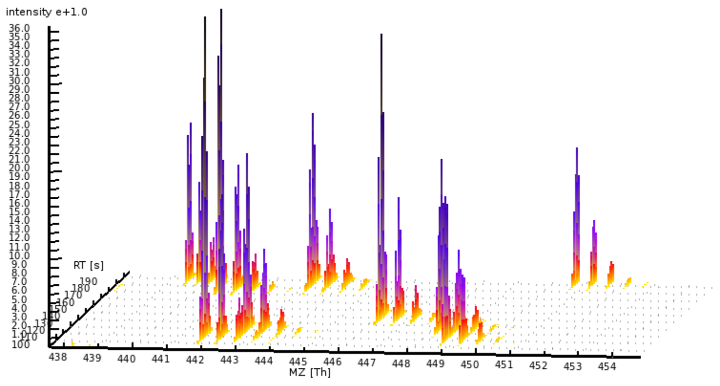
FFM was used to generate ground truth for experimental MS data i.e. for detecting peptides. The strictness of peptide detection was fine-tuned by parameters such as allowed m/z tolerance between peak patterns, typical retention time for a peptide in LC column, intensity cuttoff and peptide similarity for isotopic pattern. The parameters were fine-tuned manually to find maximum number of peptides and minimal false positives in the experimental data. Once set, the same parameter settings were used for all ground truth generation experiments so that the neural network could learn and predict the same peptide feature detection, especially for peptides occurring in labelled pairs. However, FFM has its shortcomings in the way that a general parameter setting would miss a number of peptide features in the experimental data. Moreover, low intensity peptides are hard to detect for any feature finder currently available.

2.1.2 MSSimulator

MSSimulator is a simulation software for LC-MS and LC-MS/MS experiments and is integrated into the OpenMS workflow engine 'The OpenMS Proteomics Pipeline (TOPP)' [10]. Given a list of proteins in FASTA format and a configuration file with parameter settings, MSSimulator can perform digestion of the proteins, retention-time prediction, ionization filtering and raw signal simulation for both MS and MS/MS experiments. It was used to generate peptide feature maps (featureXML file) such that every spectrum in the sampled MS spectra (mzML file) can be treated as a valid peptide. With the absence of low intensity mass trace fragments, the data obtained can be treated as noise-free. The data generated with this tool is referred to as simulated data from here on in this thesis. A graphical comparison between real and simulated MS spectra is shown in Figure 3.



(a) Real Spectra



(b) Simulated Spectra

Figure 3: A graphical comparison of experimental and simulated spectra. The real spectra contain unwanted irregular mass traces from protein fragments, whereas the simulated spectra does not contain any noise. The grey dots in (b) represent a grid for readability and should not be confused with simulated noise. Screenshots from OpenMS viewer TOPPView [4].

2.2 Neural Networks

2.2.1 Neuron: A single computational unit

Traditional machine learning algorithms rely mainly on hand-crafted feature extractors to interpret and learn from their data sets. Designing such feature extractors requires considerable efforts by experts. Deep neural networks bypass the need of manually designing a feature extractor. Instead, optimal features are learned jointly with the final classification task within the hidden layers of the network. The layers consist of computational units called neurons. Figure 4 shows a single neuron which has three inputs x_1, x_2 and x_3 and a single output. A neuron first computes an affine transformation of its inputs and applies a non-linear transformation called activation function to compute the output. The affine transformation is a weighted sum of the inputs:

$$z \ = \ \sum_{i=1}^{3} w_i x_i \ + \ b \tag{1}$$

where $\{w_1,\ w_2,\ w_3\} \in \mathbf{W}$ are the weights for the three inputs $x_1,\ x_2$, and x_3 respectively, and b is the bias. One of the most popular non-linear activation functions is the Rectified Linear Unit (ReLU):

$$f(x) = \max(0, x) \tag{2}$$

such that the output of this neuron is calculated as $y = f(z)$. In the past, other smoother non-linear activation functions such as $\tanh(x)$ or $1/(1 + \exp(-x))$ were widely used, but deep networks typically learn much faster with ReLU activation [11]. The weights and bias of the neuron are adjusted such that the error between true value and the calculated output is minimized.

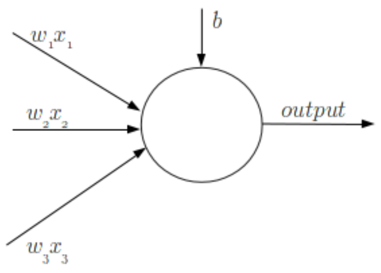


Figure 4: A basic neuron with three inputs and a single output.

An artificial neural network typically is constructed with successive layers of such neurons. Figure 5 shows a neural network with three hidden layers, a single neuron in the input layer and two neurons for outputs. In the hidden layers, the output of

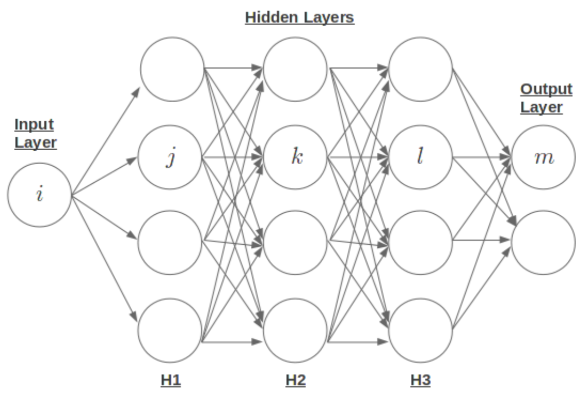


Figure 5: A fully connected neural network with three hidden layers.

each neuron is passed to all neurons in the successive layer, and each neuron takes as input the outputs from the neurons of the previous layer. Such an architecture is called a fully-connected network. The forward and backward passes are the essential computations of a network. Given the input for inference, the forward pass computes output and loss between ground truth and calculated output, whereas the backward pass computes the gradient of the loss w.r.t. the network weights. A complete forward and backward pass constitutes a single iteration. One possible loss or objective function is the average squared/absolute difference between the actual outputs T and the predicted outputs Y. Its gradient is computed using the back propagation algorithm described in section 2.2.2 The loss is minimized with Stochastic Gradient Descent (SGD) described in section 2.2.3.

2.2.2 Back Propagation Algorithm

The forward pass for the fully connected neural network shown in Figure 5 for input neuron i, through hidden neurons j, k and l and to output neuron m is given by

$$\begin{aligned}
z_j &= \sum_{i \,\epsilon\, \text{Input}} w_{ij} x_i + b_j \\
y_j &= f(z_j) \\
z_k &= \sum_{j \,\epsilon\, \text{H1}} w_{jk} y_j + b_k \\
y_k &= f(z_k) \\
z_l &= \sum_{k \,\epsilon\, \text{H2}} w_{kl} y_k + b_l \\
y_l &= f(z_l) \\
z_m &= \sum_{l \,\epsilon\, \text{H3}} w_{lm} y_l + b_m \\
y_m &= f(z_m)
\end{aligned}$$

The backward pass consists of computing the derivative of the final loss E w.r.t. output $\mathbf{y}$ and applying the chain rule to propagate it back from the output layer to the input layer. The following equations compute the backward pass through neurons

m, l, k and j:

$$\begin{aligned}
\frac{\delta E}{\delta y_m} &= \text{cost function derivative w.r.t. output } m \\
\frac{\delta E}{\delta z_m} &= \frac{\delta E}{\delta y_m}\frac{\delta y_m}{\delta z_m} \\
\frac{\delta E}{\delta y_l} &= \sum_{m\ \in\ \text{out}} w_{lm}\frac{\delta E}{\delta z_m} \\
\frac{\delta E}{\delta z_l} &= \frac{\delta E}{\delta y_l}\frac{\delta y_l}{\delta z_l} \\
\frac{\delta E}{\delta y_k} &= \sum_{l\ \in\ \text{H3}} w_{kl}\frac{\delta E}{\delta z_l} \\
\frac{\delta E}{\delta z_k} &= \frac{\delta E}{\delta y_k}\frac{\delta y_k}{\delta z_k} \\
\frac{\delta E}{\delta y_j} &= \sum_{k\ \in\ \text{H2}} w_{jk}\frac{\delta E}{\delta z_k} \\
\frac{\delta E}{\delta z_j} &= \frac{\delta E}{\delta y_j}\frac{\delta y_j}{\delta z_j}
\end{aligned}$$

2.2.3 Stochastic Gradient Descent Optimizer

The objective function $E(\mathbf{W})$ can be thought of as hilly landscape in the high-dimensional space of weights $\mathbf{W}$. An optimizer such as Stochastic Gradient Descent (SGD), adjusts the weights so as to lower the output error. SGD iterations are repeated for a small random set of training samples $\mathbf{x}$ hence giving it the name 'stochastic'. This process gives a noisy estimate of the average gradient of all training samples, but this method of estimation is quick in comparison to more elaborate optimization techniques [12]. For every iteration i of SGD, the weights $\mathbf{W}$ are updated by a linear combination of the current estimate of the negative gradient $\nabla_{\mathbf{y}} E_{\mathbf{x}}$ and the weight update for the previous iteration. SGD optimization can be controlled by adjusting learning rate α and momentum μ where α is the step length along the negative gradient and μ weights the influence of the previous update.

Algorithm 1 presents SGD as applied to the neural network optimization, where $\mathbf{x}$ represents a vector of inputs, $\mathbf{y}$ represents a vector of neuron outputs, $\mathbf{z}$ represent the outputs of the affine transformations per layer, i.e. the inputs to activation function f such that $\mathbf{y} = f(\mathbf{z})$, l represents the network layers such that $l \in \{0, ..., L\}$, $\mathbf{W}$

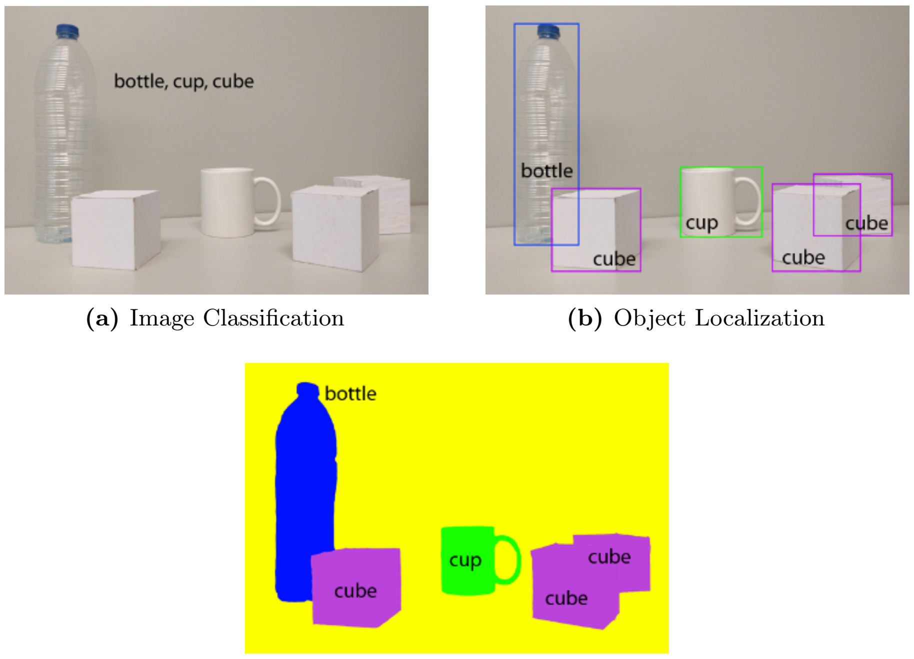
Algorithm 1 Stochastic Gradient Descent: Neural Network

```
foreach iteration $i \in \{1, ..., I\}$ do
    Create a mini batch of $m$ samples $\mathbf{x}_0 \dots \mathbf{x}_{m-1}$
    foreach sample $\mathbf{x}$ do
        $\mathbf{y}^{\mathbf{x},0} \leftarrow \mathbf{x}$                          ▷ Set activation for input layer
        foreach Layer $l \in \{1 \dots L-1\}$ do          ▷ Forward pass; computing outputs layer by layer
            $\mathbf{z}^{\mathbf{x},l} \leftarrow \mathbf{W}^l \mathbf{y}^{\mathbf{x},l-1} + \mathbf{b}^l$
            $\mathbf{y}^{\mathbf{x},l} \leftarrow f(\mathbf{z}^{\mathbf{x},l})$
        end for
        $\boldsymbol{\delta}^{\mathbf{x},L} \leftarrow \nabla_{\mathbf{y}} E_{\mathbf{x}} \odot f'(\mathbf{z}^{\mathbf{x},L})$     ▷ Compute error at the output layer by differentiating the cost function w.r.t. the output neurons where $\odot$ represents element-wise multiplication
        foreach Layer $l \in L-1, L-2 \dots 2$ do        ▷ Backpropagate error; at each hidden layer, compute the error derivative w.r.t. the output of the each neuron
            $\boldsymbol{\delta}^{\mathbf{x},l} \leftarrow ((\mathbf{W}^{l+1})^\top \boldsymbol{\delta}^{\mathbf{x},l+1}) \odot f'(\mathbf{z}^{\mathbf{x},l})$
        end for
    end for
    foreach $l \in L, L-1 \dots 2$ do                       ▷ Gradient descent
        $\mathbf{V}_i^l \leftarrow \frac{\mu}{m} \mathbf{V}_{i-1}^l - \frac{\alpha}{m} \sum_{\mathbf{x}} \boldsymbol{\delta}^{\mathbf{x},l} (\mathbf{y}^{\mathbf{x},l-1})^\top$
        $\mathbf{W}^l \leftarrow \mathbf{W}^l + \mathbf{V}_i^l$
    end for
end for
```

represents a weight matrix, $\boldsymbol{\delta}$ is the error derivative Jacobian matrix, $\nabla_{\mathbf{y}} E_{\mathbf{x}}$ is the Error gradient w.r.t. to outputs for the input vector $\mathbf{x}$, $\mathbf{V}_i$ is the weight adjustment for iteration i, α is the learning rate and μ is the momentum. Note that bias b discussed in section 2.2.1 can be treated as a weight for a corresponding vector of $\mathbf{1}$'s as input $\mathbf{x}$.

2.3 Convolutional Neural Network

Convolutional neural networks have become the de facto standard for visual recognition tasks. Three examples of such tasks are image classification, object localization and semantic segmentation as shown in Figure 6. Usually the first few stages of a CNN are composed of convolution and pooling layers, whereas we also employ a deconvolution layer in our network as well, described in more detail in section 4.3.



(a) Image Classification **(b)** Object Localization

(c) Semantic Segmentation

Figure 6: Examples of visual recognition tasks for CNNs. (a) Class labels are assigned for the whole image. (b) Bounding boxes are drawn for different objects found in the image. (c) Every pixel in the image is classified in order to recognize whole regions. Images from [13]

.

Convolution Layer

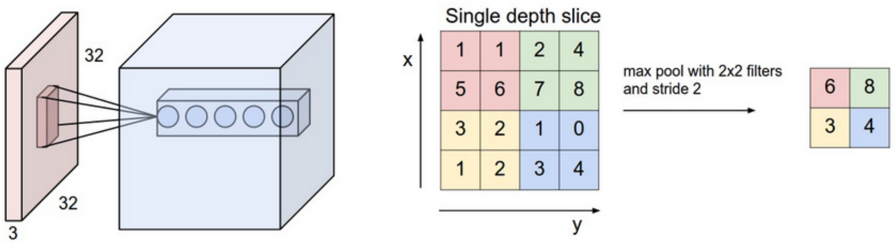
Each neuron in the convolution layer is connected to a small region of the input volume but extends through the full depth of the volume. The small region corresponds to the kernel size of the layer, see Figure 7a. During the forward pass, a kernel comprising of learnable weights is convolved across the width and height of the input volume and computes the dot product of the kernel weights and the input. This results in a 2D activation map for the kernel. Repeating the convolution operation and stacking the subsequent activation maps along the depth dimension produces the output volume of the layer. Discrete convolution operation is given by:

$$(f * g)(t) = \sum_{i=-\infty}^{\infty} f(i)g(t-i) \tag{3}$$

where f is the input signal and g is the kernel weight. The width and height of activation maps depend on the kernel size, kernel stride and input padding.

$$width_{out} = \frac{width_{in} - kernel_{width} + 2(padding_{width})}{stride_{width}} + 1$$

$$height_{out} = \frac{height_{in} - kernel_{height} + 2(padding_{height})}{stride_{width}} + 1$$



(a) A small region of the input volume in red is connected to volume of neurons in a convolutional layer.

(b) Max pooling operation for downsampling the input depth slice by a factor of 2

Figure 7: Convolution layer connectivity and Max Pooling operation. Images from [14]

.

Pooling Layer

The pooling layer also has a kernel which slides across the width and height of the input. The pooling kernel does not have any learnable weights, rather it combines its input region into a single neuron in the next layer. Figure 7b depicts a *max pooling* operation. The pooling layer operates independently on every depth slice of the input, but downsamples the spatial dimensions of the input. The downsampled width and height are given by:

$$width_{out} = \frac{width_{in} - kernel_{width}}{stride_{width}} + 1$$

$height_{out} \; = \; \frac{height_{in} \; - \; kernel_{height}}{stride_{height}} \; + \; 1$

Deconvolution Layer

Deconvolution layer is also known as upconvolution, transposed convolution or fractionally-strided convolution. This layer is used to project the input volume into higher-dimensional space. The discrete strided convolution is computed by:

$$(f \; \hat{*} \; g)(2t) = \sum_{i=-\infty}^{\infty} f(i)g(t-i) \tag{4}$$

$$(f \; \hat{*} \; g)(2t+1) = \sum_{i=-\infty}^{\infty} f(i)g(t-i+1) \tag{5}$$

where f is the input signal and g is kernel weight. The output height and width are given by:
$width_{out} \; = \; stride_{width} \times width_{in} \; + \; kernel_{width} \; - \; 2(padding_{width})$

$height_{out} \; = \; stride_{height} \times height_{in} \; + \; kernel_{height} \; - \; 2(padding_{height})$

3 Related Work

In recent years, artificial neural networks have been adapted to address the challenges in proteomics. Kim et al. propose a CNN based framework ‘DeepPep’ for protein inference [15]. Assembling peptides identified from tandem mass spectra into a list of proteins, referred to as protein inference, is a critical step in proteomics research. Given a peptide profile as a sequence, DeepPep predicts a list of scored proteins. Their training data comprises of peptide sequences, peptide probability returned by a mass spectroscopy database search and protein sequences from the respective organism. They define peptide probability as the probability that the peptide identified through a database search from the mass spectra is the correct one. Binary strings are generated for proteins containing ones where a specific peptide matches the protein sequence and zeros everywhere else. A CNN is trained to predict the peptide probability, and all peptide probabilities are predicted with and without the presence of each protein. A score based on the difference in peptide probability between being present or absent is assigned to each protein. DeepPep’s CNN comprises of four sequential convolution layers where a pooling layer is applied between each convolution layer and a fully connected layer that computes the output. The framework is interesting in the sense that it uses a CNN for to learn sequence-level location information in one dimensional binary strings. However, their main focus was protein inference and the method relies on peptide probabilities whereas we aimed at detecting valid peptides without taking any database search into account.

Other notable uses of artificial neural networks in MS based proteomics are briefly described here. Shinoda et al. use amino acid sequences as input to a fully connected 16-4-1 neural network to predict retention time in LC-MS data in [16]. For a protein undergoing a LC-MS/MS experiment, only a portion of its peptides are detected. This might occur due to many reasons such as incomplete proteolytic digestion, small size, poor binding or elution from the type of LC column used, limited detectable mass range of the mass spectrometer, bias toward detecting peptides with an intense MS signal in mixtures, phenomenon of "ion suppression", charge prior to ionization, and non-covalent interactions between peptides in the gas phase while in the mass spectrometer [5]. Sanders et al. [5] designed a 3-layer neural network for predicting peptide detectability as observable or unobservable, where as it is treated as a regression problem by Li et al. in [17] and Tang et al. in [18]. In [19], Zhou et al. identify 35 factors which might influence the behaviour of fragmented peptides in a MS/MS spectrum and train a Bayesian Neural Network to predict spectra intensities. They also proposed that the identified factors can be used for protein identification and quantification but did not present any framework.

Another notable use of CNN on MS spectra is DeepNovo, a deep neural network model for de novo peptide sequencing proposed by Tran et al. in [20]. De novo sequencing reconstructs the amino acid sequence of a peptide given an MS/MS spectrum and the peptide mass. The model is based on a CNN and recurrent neural network (RNN). RNNs process an input sequence one element at a time, maintaining in their hidden units a 'state vector' that implicitly contains information about the history of all the past elements of the sequence. The CNN in DeepNovo learns the features of mass spectra (intensity vs. MZ plots in this case) whereas the RNN tries to predict the next amino acid in the sequence. The model also uses local dynamic programming to make sure that the mass of predicted amino acid sequence is not greater than the provided peptide mass. Finally, there is no work currently available, as per our knowledge, which has used MS Spectra and formulated peptide detection problem as a visual recognition task.

4 Approach

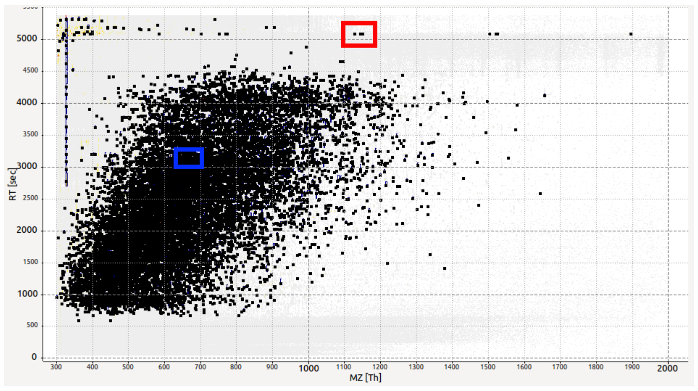
4.1 Data Insight

Experimental data consisted of mass spectra for *Escherichia coli* samples. Three different MS spectra files were used during this project, two of which had a dimethyl label and one was unlabelled. One of the labelled spectra was used to render test images while the other two were used for training images. The unlabelled spectra is shown in full in Figure 8a where black labels represent peptide features detected by FFM. As can be seen, the spectra are not evenly populated. A zoomed-in dense region and a sparse region are shown in Figure 8b and Figure 8c respectively. Many peptides have overlapping spectra along both RT and MZ axis. The overlapping peptides do not pose a significant challenge for FFM as they can be detected due to high signal-to-noise ratio. High intensity spectra are also more common in dense regions, though there is no correlation between the two. Note that there are regular intervals along RT axis. This is an inherent artifact for experimental data gathered using 'Thermo Q Exactive' mass spectrometer. We dealt with it by repeating the intensity values for every spectrum along RT axis in the preprocessing steps further discussed in the following section.

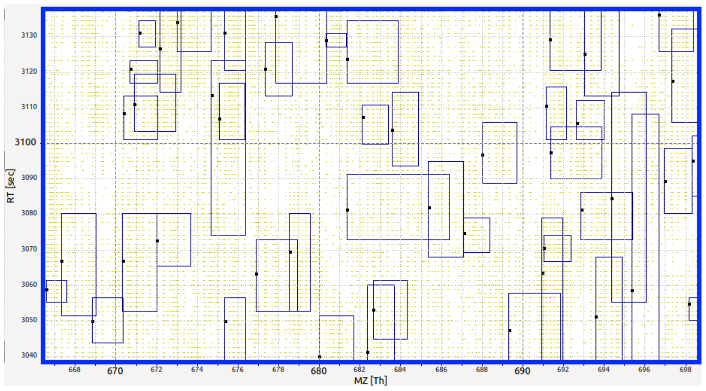
MSSimulator was used to generate simulated spectra, as in Figure 9a, for 100 random *E. coli* samples, where the number of samples were chosen by comparing dense regions of resulting simulated data to that of experimental data. Since both real and simulated spectra were to be rendered into grey scale images, having dense regions with visually comparable overlapping spectra was important. Figure 8b and Figure 9b both show 90 s across 30 Th regions of MS spectar and can be compared in this regard. Another important characteristic of the simulated data is the absence of any background noise. In this way, the simulated data only helps the network to learn overlapping peptide features.

The simulation spectra for 100 samples lies in ranges $[0, 2000]$ along RT axis and

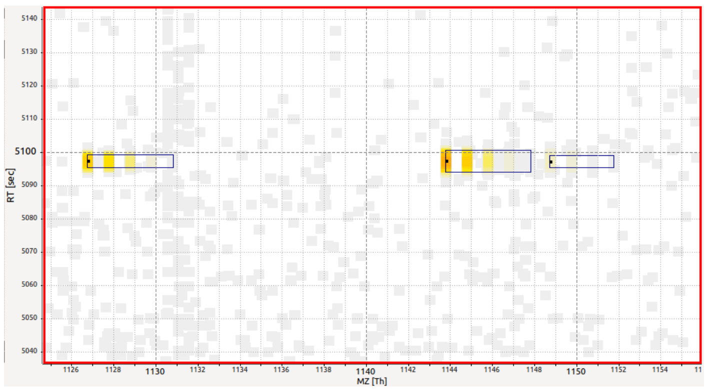
$[200, 1800]$ along MZ axis, whereas the experimental spectra span $[60, 5400]$ and $[50, 6000]$ along RT and MZ axis respectively. Since, both spectra are subsampled, with overlap, to generate images of size 3000×5500 pixels (size discussed in more detail in the following section), a total of 12 simulated spectra were generated, each from a different set of 100 protein samples so that the total number of rendered images are equal for both real and simulated datasets.



(a) Experimental Data

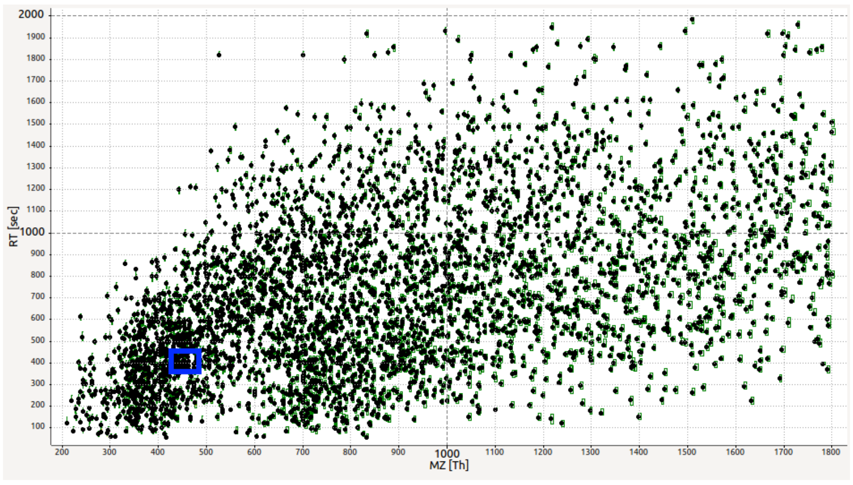


(b) Dense Region

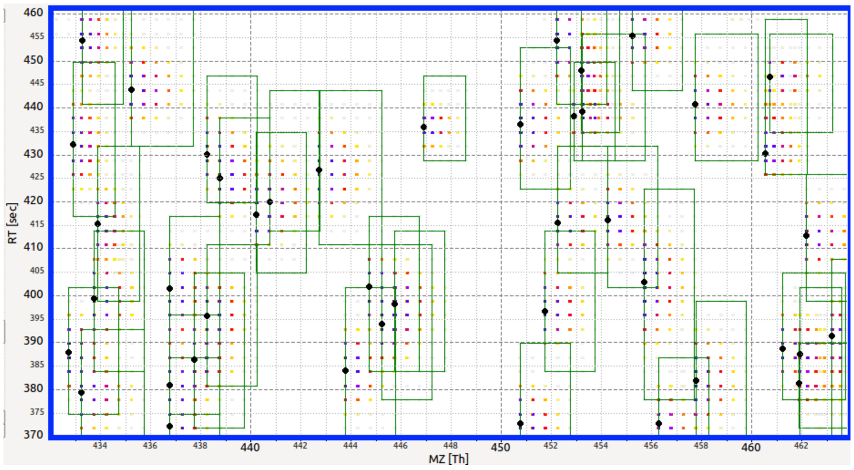


(c) Sparse Region

Figure 8: Experimental MS Spectra. Panels (b) and (c) depict 90 s across 30 Th on the MS Spectra grid. (a) Peptides detected by FFM are marked by black squares. Spectra density decreases towards higher MZ and RT values. Higher intensity spectra are also more common in dense regions. (b) Overlapping spectra for valid peptides is visible in the dense region. (c) Signal-to-noise ratio drops for valid peptides in low-intensity regions.



(a) Simulated Data for 100 *E. coli* samples



(b) Dense region in simulated Data

Figure 9: Simulated data generated using MSSimulator tool in OpenMS [7]

4.2 Data Preprocessing

In order to feed data to the CNN, 3D MS spectra are converted to 2D grey-scale images via preprocessing steps. The first challenge is determining a rendering resolution since the MS spectra does not lie on a “native” grid with fixed resolution. As it can be observed in Figure 8b and Figure 9b, MS spectra can have overlaps along both MZ and RT axes. Distinguishing individual mass traces is possible. This requires sampling with at least 0.001 Th and 0.03s step size along MZ and RT axis respectively, as determined by observation of overlaps in experimental data. Such a high rendering resolution would also require efficient interpolation techniques which maintain the characteristic features of individual spectra. Moreover, sampling of MS spectra differs from one brand of mass spectrometer manufacturer to another. High rendering resolution also requires a CNN with a field of view that is big enough to capture at least one typical peptide feature spectra. A high resolution is thus difficult to work with mainly due to limitation of RAM, consumed during pre- and post-processing steps, GPU memory, consumed during CNN training and prediction, and long training time for the CNN to go through the whole spectra iamges dataset. Keeping in mind the trade-off between distinguishing individual mass traces and computational resources available, a rendering resolution of 0.027 Th/px along the MZ-axis and 0.116 s/px along the RT axis was fixed. Some of the overlapping spectra with similar MZ range and MZ offset between isotopic peaks may not be distinguishable anymore at this resolution, but it is assumed that their elution profiles along RT axis can still guide the neural network training process. The CNN’s field of view required for this resolution is discussed in more detail in the following section.

In the first step of preprocessing, the spectra are read from the mzML file and spline interpolated such that a single spectrum has continuous values along the MZ-axis. Without interpolation, single spectra might have missing values along the MZ-axis. The effect of interpolation on a single mass spectrum is shown in Figure 10. Secondly, the missing values along RT-axis are filled using the last known value for every mass spectrum, this ensures that there are no breaks along the RT-axis. The effect of repeated values on a rendered image of simulated data is illustrated in Figure 12. In the next step we cropped the spectra into tiles of shape 3000$\times$5500px with the fixed rendering resolution already mentioned above. The tiles overlap to make sure the spectra cropped at boundaries in one tile are included in the next tile. Smaller images fit easily into the RAM for pre- and post-processing. Figure 11a

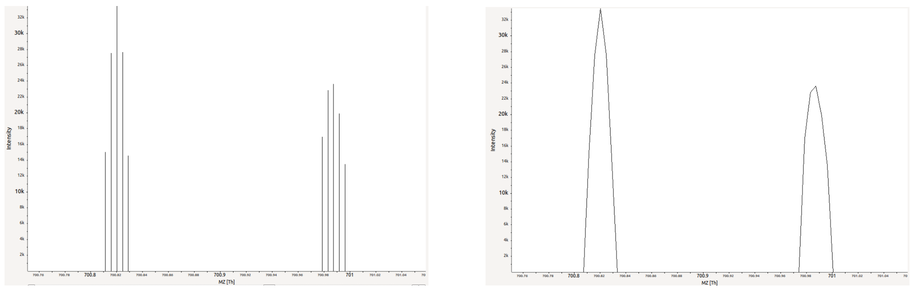

(a) (b)

Figure 10: Mass trace values measured by mass spectrometer, as in (a), are interpolated so that a single profile has continues values, as in (b).

shows an image created using the ImageCreator tool in OpenMS where the missing values along the RT-axis are visible (note that these images are included only for demonstration and were not used for network training), where as Figure 13a shows the corresponding final grey-scale normalized image for the same spectra. Figure 11b shows valid peptides, as detected by FFM, enclosed in bounding boxes. Each detected peptide is marked with a 'label' at the highest peak of the *mono-isotopic peak profile*, i.e. the cluster of peaks with the lowest MZ value. A single grey-scale 2D image can

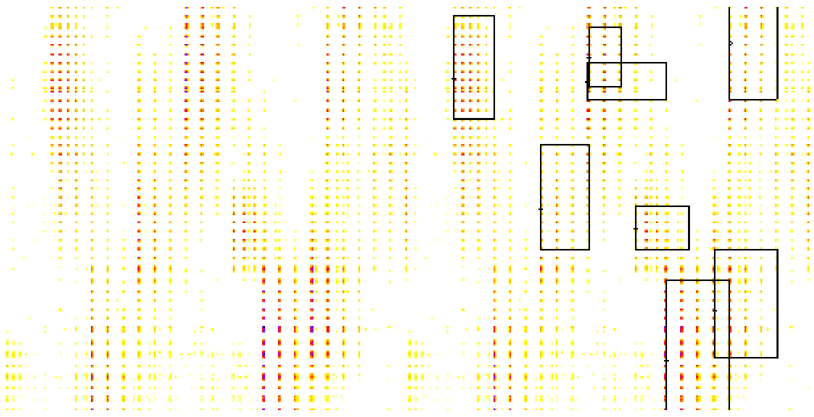

(a) Spectra **(b)** Features

Figure 11: 2D view of spectra and peptides features on RT vs MZ axis. (a) MS spectra where intensities are encoded by color. (b) Bounding boxes annotated around valid peptides as detected by FFM. Both images generated with OpenMS's ImageCreator tool [7].

be represented by a matrix D. For each of the sub-sampled images, log-intensity values are normalized such that $D(\mathbf{x}) \in [0, 1]$ for every pixel $\mathbf{x} \in \Omega$, where $\Omega \subset \mathbb{Z}^2$ is the image domain.

$$D(\mathbf{x}) = \begin{cases} \frac{\log(I(g(\mathbf{x}))}{\max(\log I)} & I(g(\mathbf{x})) > 1 \\ 0 & \text{otherwise} \end{cases}$$

$g : \mathbf{x} \mapsto (\mathrm{mz}, \mathrm{rt})$ maps image pixels to the corresponding MZ and RT values in MS spectrum. We reformulate the detection problem as semantic segmentation. For this the ground truth peptide position is presented to the network as disk of 1's with a radius 6 pixels. In this way, all pixels $\mathbf{x}$ in the corresponding ground truth matrix L have a label $L(\mathbf{x}) \in \{0, 1\}$, as shown in Figure 13b.

$$L(\mathbf{x}) = \begin{cases} 1 & \mathbf{x}' \text{ is peak of mono-isotope profile } \wedge \|\mathbf{x} - \mathbf{x}'\| \leq 6 \\ 0 & \text{otherwise} \end{cases}$$

Every pixel in the ground truth label image $L(\mathbf{x})$ is also given a binary weight in matrix W such that $W(\mathbf{x}) \in \{0, 1\}$. In this way the dilated labels have a weight of 1 but are surrounded by a ring of $0's$ with outer radius 12 px, as depicted in Figure 13d. This weighting scheme allows the network to cope with inaccurate peptide label positions in the ground truth.

$$W(\mathbf{x}) = \begin{cases} 0 & L(\mathbf{x}') = 0 \wedge 6 < \|\mathbf{x} - \mathbf{x}'\| \leq 12 \\ 1 & \text{otherwise} \end{cases}$$

Finally, a 'convex hull' image is generated with a corresponding label matrix L_{CH}. A convex hull can be thought of as a vertical shadow of the mono-isotope profile of a known peptide in the MS spectra. A sample convex hull's image is shown in Figure 13c. Both label and convex hull images are treated as ground truth for training the network.

$$L_{\mathrm{CH}}(\mathbf{x}) = \begin{cases} 1 & D(\mathbf{x}) \in \text{mono-isotope peptide profile} \\ 0 & \text{otherwise} \end{cases}$$

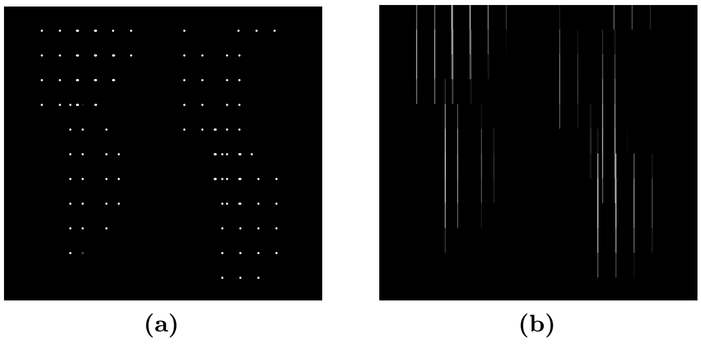
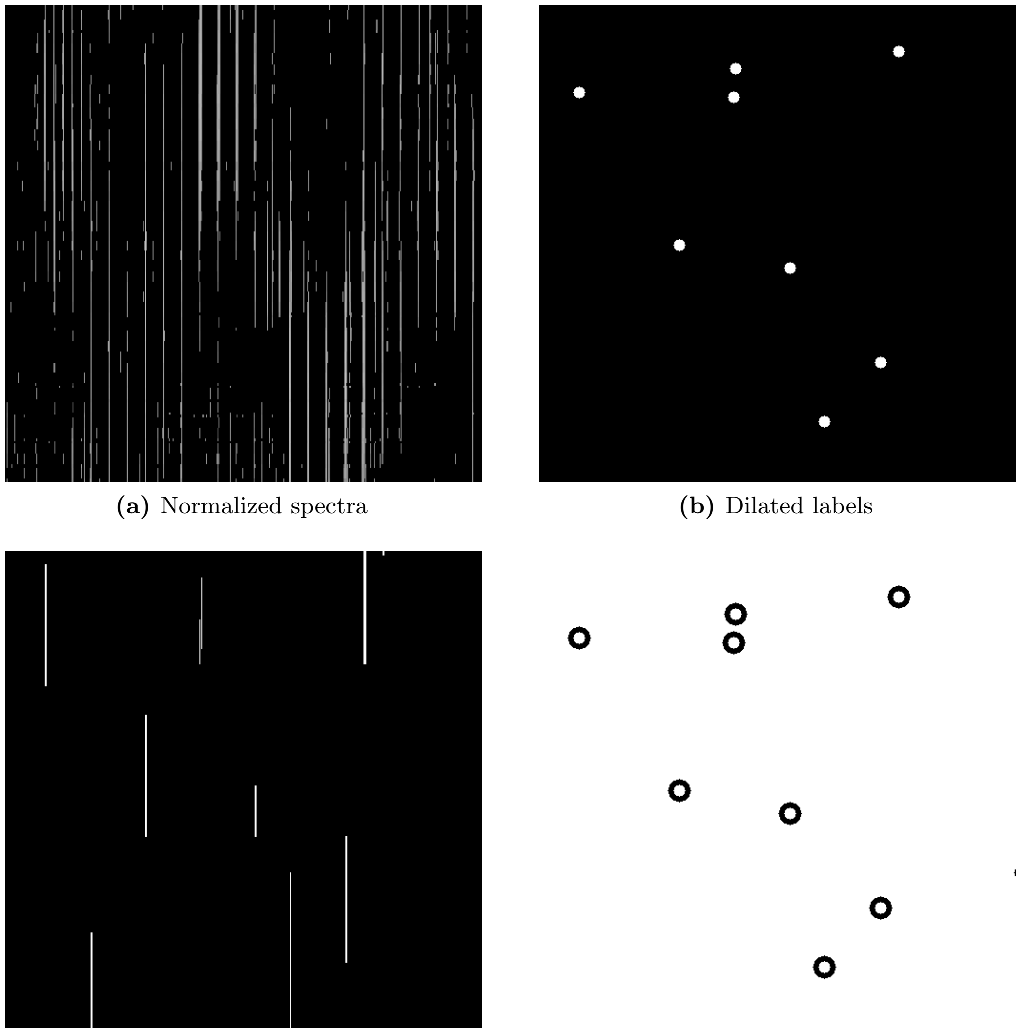


Figure 12: Rendered image of simulated data (a) without intensity values repetition along RT axis and (b) with repetition.

4.3 CNN Architecture

CNNs are deep, feed forward networks that are easier to train and that generalize much better than networks with full connectivity between adjacent layers. U-Net is such a CNN which comprises of a contracting path to capture context in the training images and a symmetric expanding path that enables precise localization [21]. The downward contraction path comprises of successive convolution and max-pooling operations, where as the upward expansion path has deconvolution and convolution operations. All convolution operations use isotropic 3×3 kernels with a stride of 1×1, except for the last operation that uses a 1×1 kernel to generate the two output channels. Each max-pooling layer has a kernel size of 2×2 and a stride of 2×2, which halves width and height of the input image i.e. it performs downsampling by a factor of 2. Each deconvolution operation is followed by a crop and concatenate operation which combines the convolution feature maps from the downward path with the feature maps of the upward path, as can be seen in Figure 14. This operation helps to restore the effect of pooling layers which discards the context location information in order to align class maps for pixel-wise prediction. Only valid part of each convolution output, i.e. the feature map only contains the pixels for which the full context is available in the input image. Cropping before concatenating is necessary due to the loss of pixels at the border after every convolution layer. Each deconvolution operation (up-convolution) has a kernel size of $2{\times}2$ and a stride of $2{\times}2$, which results in a doubled width and height of the input. All convolution and deconvolution layers are followed by a ReLU layer. In total, the network has 23 convolution layers, 5 max-pooling layers, 5 deconvolution layers and 27 ReLU activation layers.



(a) Normalized spectra

(b) Dilated labels

(c) Convex Hulls

(d) Label weights

Figure 13: Images used for network training. In all panels: y-axis: RT, x-axis: MZ. (a) Rendered spectrum of experimental data. Higher intensity peaks are shown in white and background as black. (b) Dilated labels used as ground truth for valid peptide. (c) Convex hull for the mono-isotope peak profile is represented by 1's and the background by 0's. (f) Pixel weights corresponding to (b). Each label has weight 1 and is surrounded by a ring of 0's.

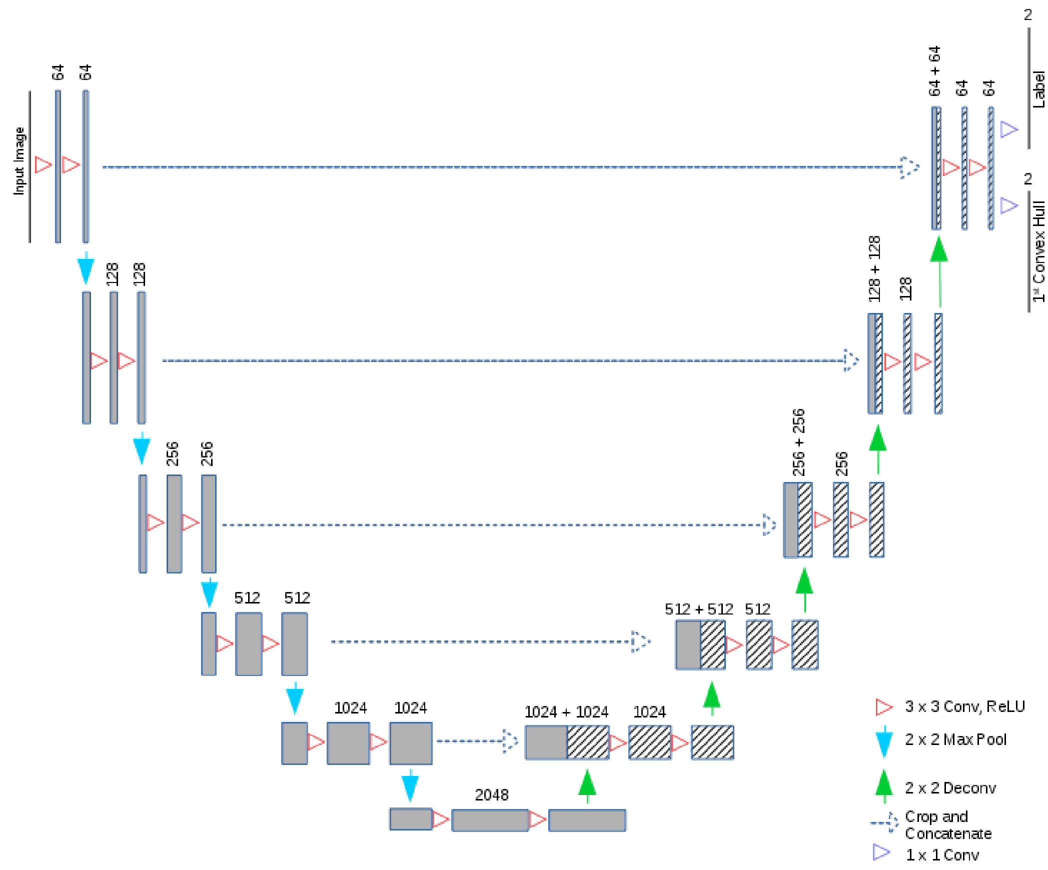


Figure 14: The neural network architecture referred to as U-Net. Solid grey rectangles represent convolution feature maps in the contraction (encoding) path of the network, and rectangles with hashing lines represent feature maps in the expansion (decoding) path. Two final convolution operations compute the outputs for Label and Convex Hull.

The proposed network differs from the original U-Net by Ronneberger et al. in a couple of ways. Our network simultaneously solves two semantic segmentation tasks for peptide peak position detection and convex hull segmentation. Data augmentation as proposed by Ronneberger et al. is only partially applicable to MS spectra. Especially elastic deformations are not used in our configuration of the network because it would introduce anomalies in the MS spectra. The strategy of doubling the number of output channels for every successive level of resolution after convolution layers and halving them after deconvolution layers was however maintained. The original U-net has four levels of resolution (or stages) whereas we employ five levels. This was crucial to increase the field of view or *receptive field* of the network.

The receptive field of a single neuron is the region of its input space which directly influences the output of the neuron. A neuron in convolution, deconvolution or max-pooling layer has a receptive field equal to the layer's kernel size, where the extent of connectivity along the depth axis is always equal to the depth of the input volume. (Note that *depth* here refers to the third dimension of input volume and should not be confused with the network's depth.) In this regard, a neuron in the first convolution layer has a 3×3 view of the input image. A neuron on the second convolution layer has a 3×3 view of the first convolution layer, and hence a 5×5 view of the input image. A neuron on the first pooling layer has a 2×2 view of the second convolution layer and a 10×10 view of the input image. In this way, the receptive field of the output layer can be calculated, which defines the receptive field of the network. With the additional level of resolution, the receptive field of the network is 376×376 pixels, or equivalently 44 s across 10 Th on the MS spectra grid. A receptive field covering 10 Th on MZ-axis is enough to capture multiple peptide features, as can be seen in Figure 8b and Figure 9b, whereas as 44 s on the RT-axis matches one of the parameter setting for FFM; typical retention time = 40 s. This parameter does not define an upper bound on RT values for FFM peptide detection, but an RT value for peptides of interest in general. However, having another level of resolution also requires more GPU memory for computation which has to be considered before training the network.

4.4 Network Training

Keeping into consideration the reduction in width and height after convolution layers and downsampling by a factor of 2^5 in total, the minimum possible input image size for the network is 380×380 pixels. This size would result in a convolution feature map of size 4×4×2048 at lowest feature resolution. Bigger input tile size was preferred for training over larger batch size, so a batch size of 1 was used for every iteration and the input tile size for the network was adapted according to use all available GPU memory; 700×700 for nVidia TITAN with 6GB memory and 1052×1052 for nVidia TITAN X with 12GB memory. The input tile size should not be confused with subsampled tiles, the former refers to input size for a single iteration of CNN and the latter refers to the image size that was rendered from spectra. The stochastic gradient descent implementation of Caffe [22] was used to train the network with learning rate $\alpha = 0.01$ and momentum $\mu = 0.9$.

As Ronneberger et al., we used the cross-entropy of the pixel-wise softmax over the final output channels as objective function for training both 'Label' and 'Convex Hull' outputs . Softmax and cross-entropy are given in equations 6 and 7 respectively in the following section.

4.5 Loss Function

The objective function used for training the neural network was the cross entropy between the pixel wise softmax score and the one-hot encoded ground truth labels. For each pixel, softmax maps the network outputs to values between zero and one such that they sum to one over all output channels: $\mathbf{p} : \mathbb{R}^C \mapsto [0,1]^C$ where C is the number of output channels. The softmax is defined as:

$$p_k(\mathbf{x}) := \frac{e^{a_k(\mathbf{x})}}{\sum_{j=1}^{C} e^{a_j(\mathbf{x})}} \tag{6}$$

where $a_k(\mathbf{x})$ denotes the activation of output channel k at pixel position $\mathbf{x} \in \Omega$. C is the number of classes and $p_k(\mathbf{x})$ is the approximated maximum-function, i.e. $p_k(\mathbf{x}) \approx 1$ for the maximum activation $a_k(\mathbf{x})$ and $p_k(\mathbf{x}) \approx 0$ for all other k. Taking the softmax of the network output highlights the largest values in the output and suppresses the values which are significantly lower than the maximum value, for every channel. In this way, the output is transformed to obtain a discrete probability density function which can be compared to the true pdf via cross-entropy which measures the similarity between the two pdf's.

Cross entropy computes the loss between the pixel score and its ground truth i.e. at each position the deviation of $p_{l(\mathbf{x})}(\mathbf{x})$ from equation 6 using

$$E = \sum_{\mathbf{x} \in \Omega} W(\mathbf{x}) \log \left(p_{l(\mathbf{x})}(\mathbf{x}) \right) \tag{7}$$

where $l \in \{L, L_{\mathrm{CH}}\}$ are the true labels and W are the per-pixel weights as introduced before. In the final network configuration, the number of output channels is $C = 2$; for background and foreground, where only the foreground channel is used in the post processing step to generate peptide feature maps.

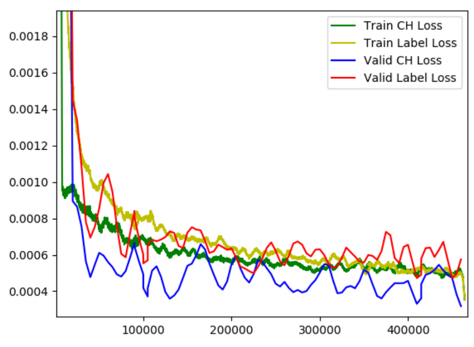
5 Experiments and Results

5.1 Experiment details

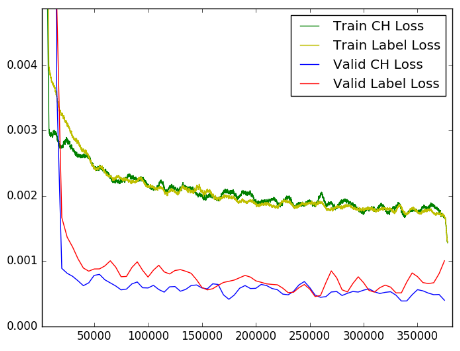
Three compositions of training data are possible: simulated data only, mixed real and simulated data, real data only. The neural network was trained for four experiments:

- Experiment 1: Network trained and validated on simulated data only
- Experiment 2: Network trained on mixed real and simulated data, validated on simulated data.
- Experiment 3: Network trained on mixed real and simulated data, validated on real data
- Experiment 4: Network trained and validated on real data only

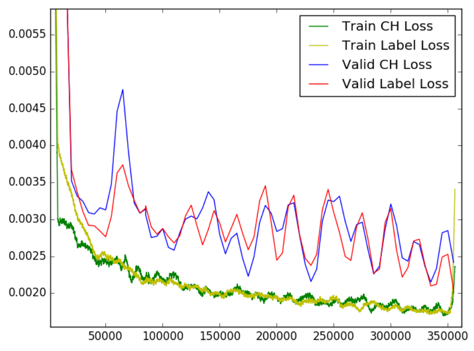
Except for the differences in composition of data, all experiments were conducted with the same network configuration. For every experimental spectra (.mzML) file, three simulated spectra files were used such that the resulting composition of mixed data has a 50 : 50 ratio. Training and validation losses for the four experiments are shown in Figure 15, where validation loss was calculated after every 5000^{th} iteration on a random set of 100 images. Experiment 2 and 3 are trained on the same dataset but their loss curves are not identical. This is because of the random order of reading images for every iteration. One interesting observation about CH loss is that it leads the learning process for all experiments except the last one. It should be noted that the loss curves are smoothed by convolution for 2000 iterations for training loss and 5 iterations for validation loss. Due to this reason, the sudden rise or fall of loss curves towards the last iterations have to be ignored.



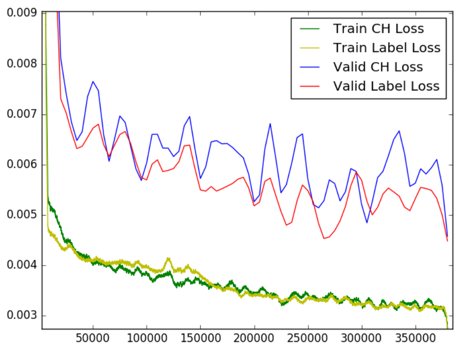
(a) Training and validation on simulated data



(b) Training on real and simulated data, validation on simulated data



(c) Training on real and simulated data, validation on real data



(d) Training and validation on real data

Figure 15: Loss plots for training U-Net on different compositions of training data. In all panels: y-axis: Cross entropy of softmax score, x-axis: No. of iterations.

5.2 Evaluation Metrics

The network's predictions for labels are evaluated as classifications. Every label $l \in L$ can be treated as a connected component, whereas the network's predictions need to be processed before evaluation. The foreground channel is binarized for softmax threshold > 0.5 and dilated, as shown in Figure 16. The dilation is necessary to fill any gaps within a single label and to connect any two parts of a single label. To avoid combining two different labels into a single label, the binary dilation was performed with a structuring element of square connectivity equal to one for a single iteration only. A connected component in dilated detections is assigned to a connected component in L if the squared distance between their centers of mass is within an allowed tolerance of 10 pixels, which is equivalent to 0.27 Th and 1.16 s for MZ and RT-axis respectively. Increasing the allowed tolerance could give rise to multiple nearest neighbour assignments. The assignment strategy is still susceptible to multiple assignments but it is assumed that two labels lying within the allowed tolerance level are different spectra from similar ionized molecules.

Moreover, the post processing of predicted labels follows the same procedure of finding connected components and mapping them to RT and MZ values in order to generate feature maps (.featureXML file).

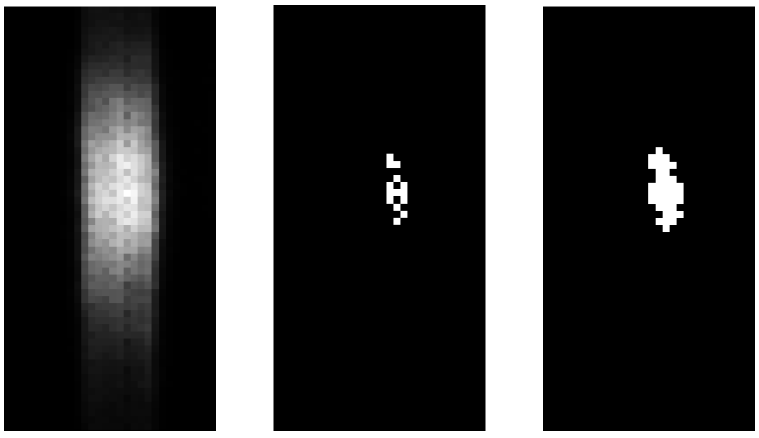

(a) Softmax label **(b)** Binarized label **(c)** Dilated label

Figure 16: Processing predicted labels with softmax score close to threshold 0.5 (a) Softmax label as predicted by the neural network. (b) Binarized label for softmax threshold > 0.5, where two connected components are visible in this example. (c) Dilated label with filled gaps and combined connected components.

In terms of classification, a ground truth label correctly predicted by the network is a *True Positive*, a label missed in the prediction is a *False Negative* and a predicted label with no corresponding ground truth label is a *False Positive.* The evaluation metrics *Precision*, *Recall F-measure* are given by:

$$\text{Precision} = \frac{|\text{True Positives}|}{|\text{True Positives}| + |\text{False Positives}|} \tag{8}$$

$$\text{Recall} = \frac{|\text{True Positives}|}{|\text{True Positives}| + |\text{False Negatives}|} \tag{9}$$

$$\text{F-measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10}$$

Precision is thus a measure of the efficiency of the neural network's predictions whereas recall measures the relevancy of those predictions. F-measure is the harmonic mean of precision and recall, where the best value 1 represents perfect predictions.

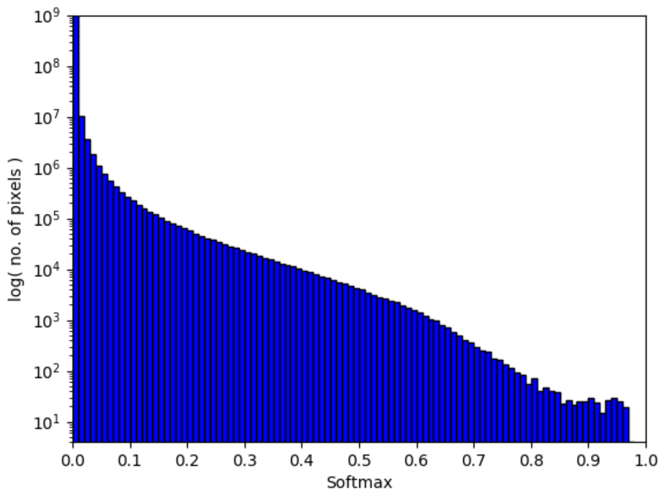
5.3 Quantitative Results

Table 1 shows the quantitative results of predictions for the different experiments.

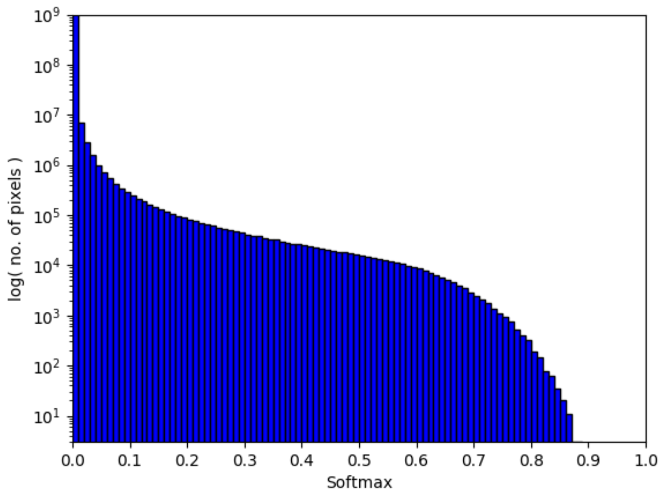
Table 1: Evaluation of Predicted Labels

Exp	Training Data	Validation Data	Results				
			Training Set		Test Set		
			Precision	Recall	Precision	Recall	F-measure
1	Sim	Sim	92%	91%	92%	92%	0.92
2	Real + Sim	Sim	94%	88%	94%	89%	0.91
3	Real + Sim	Real	55%	7%	54%	4%	0.07
4	Real	Real	53%	50%	50%	10%	0.17

Simulated data is easier to learn for the neural network due to a very high signal-to-noise ratio for both pure and mixed data composition. This is evident in the prediction results for Exp. 1 and 2, as well as in loss plots in Figure 15a and Figure 15b during training. The mixed composition fails to improve the score for real data predictions. However, the softmax distribution in Figure 17 shows that the predicted labels in real data have higher softmax scores when trained on a mixed data composition. The results are slightly better for pure real dataset training, but a recall of 10% is not sufficient to match the peptide detection ability of FFM. The reason for such a low



(a) Softmax distribution of predicted labels in real data for a model trained on both real and simulated data.



(b) Softmax distribution of predicted labels in real data for a model trained on real data only.

Figure 17: Softmax distributions of labels predicted using two different models on the same experimental data. Training on a mixed composition of real and simulated data results in higher softmax scores as compared to training on real data only.

recall and a precision of around 50% on real data is that the U-net performs better than FFM in low-intensity sparse regions of spectra. This is discussed qualitatively in the following section.

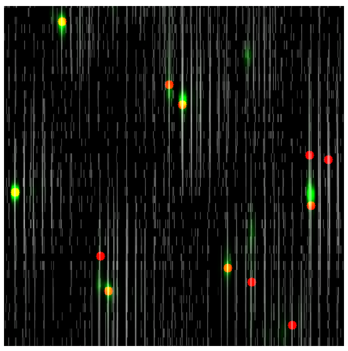
5.4 Qualitative Results

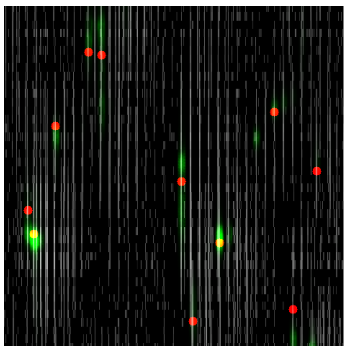
Qualitative results are shown in Figure 18, where figures (a), (d) and (c) correspond to predictions from experiment 3 which used mixed real and simulated data for training, and (b), (d) and (f) to experiment 4 which used only real data for training. Vertical grey lines represent rendered spectra, red dots represent ground truth labels and predictions are represented by green dots. The intensity of green labels represent softmax scores. As mentioned before, a softmax threshold > 0.5 was used for predictions, thus some of the visible green labels do not qualify as valid predictions. All panels in Figure 18 show 900×900 pixels or 100s across 24 Th equivalently.

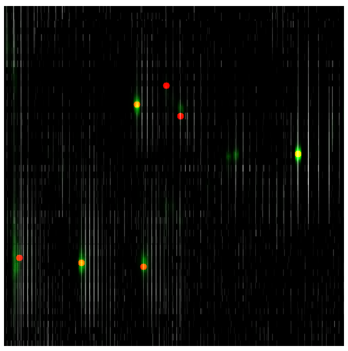
For both networks, there are few true positives in dense regions. Two ground truth labels lying close by (center right in Figure 18a and top left in Figure 18b) are also hard to predict for both networks, though model from experiment 3 identifies one of the two such labels in top center region in Figure 18a. The networks also do not produce any false positives with certainty, i.e. softmax > 0.5, in such dense regions.

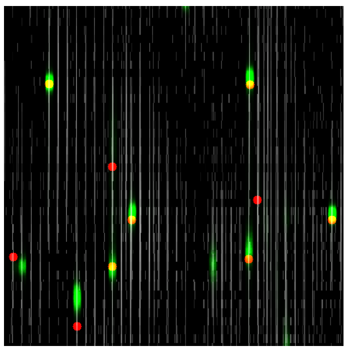
The number of true positives are higher and number of false negatives is lower in the medium density regions as shown in Figure 18c and Figure 18d. Model from experiment 3 misses two very low intensity peptides in top center region of Figure 18c. A false positive is visible in bottom center region of Figure 18d whereas two high intensity peptides are missed by model from experiment 4.

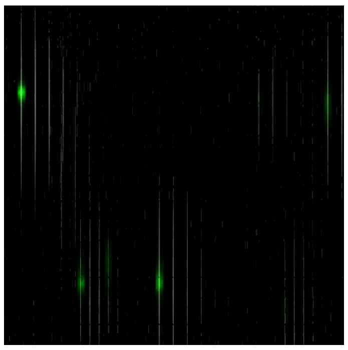
The most interesting predictions by both models occur in the sparse regions of spectra where signal-to-noise ratio is low. This is due to two reasons. First, FFM fails to annotate ground truth labels due to low signal-to-noise ratio. Secondly, from a proteomic point of view, less abundant peptides occur in these regions. As Figure 18e and Figure 18f show, the number of false positives is high for both models. Moreover, all available ground truth labels are correctly predicted by model from experiment 4 (and generally also with model from experiment 3 but such a example is not shown in the corresponding figure). It can be deduced that predictions in sparse regions and absence of ground truth labels are the main reason for a precision around 50%


(a) Exp 3 predictions in dense region


(b) Exp 4 predictions in dense region


(c) Exp 3 predictions in medium density region


(d) Exp 4 predictions in medium density region


(e) Exp 3 predictions in sparse region

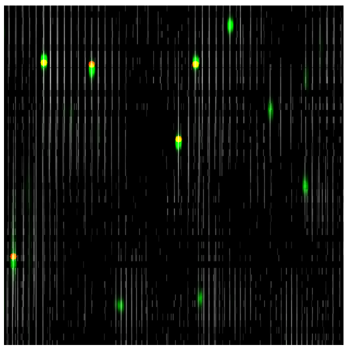

(f) Exp 4 predictions in sparse region

Figure 18: Qualitative results on real data. In all panels grey lines represent rendered spectra which are annotated with ground truth labels in red and predictions in green. The intensity of green labels represent the softmax scores. Figures (a), (c) and (e) correspond to results of network trained on mixed real and simulated data (Exp 3), and (b), (d) and (f) correspond to network trained on real data only (Exp 4).

for both experiment 3 and 4 in table 1.

However, the predictions for low-intensity regions have to be analyzed manually for correctness. MS spectra corresponding to the false positives are viewed in TOPPView [4], which also allows the measurement of m/z offset between any two isotopic peaks. If the product of m/z offset and charge z on peptide equals 1 Da, the false positive is a valid peptide.

5.5 Future Improvements

One of the reasons that the U-net performs well in detecting low-intensity peptides is that each rendered image was normalized separately. Normalizing all images with the maximum intensity observed in spectra should improve the neural network's peptide detection in dense regions, though its effect in detecting low-intensity peptides is not foreseeable. Moreover, we performed semantic segmentation on spectra images to find labels for peptides. As a future improvement, this problem can be formulated as a localization problem where a CNN learns to draw a bounding box for each detected peptide. This was one of the approaches we used in the early stages of this project but it failed to produce any significant results mainly because of a very high rendering resolution used at the time.

The receptive field of the network covered 44 s on RT axis whereas FFM produced labels for peptides with longer retention times as well. Configuring U-net for a bigger receptive field should also result is more true positives. MS spectra gathered using other brands of mass spectrometer should also be utilized in datasets so that a trained neural network is more robust to variance in data. However, data preprocessing techniques used in this project might have to be adapted for the other datasets. Finally, an ensemble of peptide detection algorithms can be used to annotate MS Spectra with ground truth labels so that the neural network learns from the strengths of all algorithms.

6 Conclusion

We reformulated peptide feature detection in MS spectra as a visual recognition problem. Deciding on a rendering resolution for spectra images was a crucial task and took the most effort and time during this project. The rendering resolution also dictates the preprocessing steps, especially the interpolation technique employed since raw spectra are too sparse to work with in this problem setting. Designing a CNN with a receptive field big enough to capture typical peptides is also an important task. Available computational resources should also be considered for these design choices.

The neural network outperforms the feature detection algorithm 'FeatureFinder-Multiplex' in regions with low signal-to-noise ratio, but performs with a 10% recall. One of the possible reasons is that each rendered image was normalized separately i.e. without the influence of the highest intensity peak of the spectra, due to which the low-intensity peptides appeared augmented to the neural network. FFM could not annotate valid peptides in the low-intensity regions, thus the network predicted a lot of false positives. Using the same normalization scale for all the rendered images should help in matching CNN's peptide feature detection performance with that of FFM. However, the low-intensity peptides are more interesting from the proteomic point of view since less abundant peptides are generally lost in such regions due to background noise. Less abundant peptides are also hard to detect for most of the algorithms currently available for peptide detection.

7 Acknowledgments

I would like to thank

- Prof. Dr. Thomas Brox and Prof. Dr. Oliver Schilling for their valuable support and accepting me as a student when I had little experience in deep learning and proteomics
- Dr. Thorsten Falk and Dr. Lars Nilse for guiding me at every stage of this project
- Dr. Hannes Röst for help with python wrappers in pyOpenMS
- Mohsin Ali for proof reading this thesis
- Last but not the least, My family and friends for their support all along

Bibliography

[1] J. V. Olsen, S.-E. Ong, and M. Mann, "Trypsin cleaves exclusively c-terminal to arginine and lysine residues," Molecular & Cellular Proteomics, vol. 3, no. 6, pp. 608–614, 2004.

[2] R. Aebersold and M. Mann, "Mass spectrometry-based proteomics," Nature, vol. 422, no. 6928, p. 198, 2003.

[3] H. Steen and M. Mann, "The abc's (and xyz's) of peptide sequencing," Nature reviews Molecular cell biology, vol. 5, no. 9, p. 699, 2004.

[4] M. Sturm and O. Kohlbacher, "Toppview: an open-source viewer for mass spectrometry data," Journal of proteome research, vol. 8, no. 7, pp. 3760–3763, 2009.

[5] W. S. Sanders, S. M. Bridges, F. M. McCarthy, B. Nanduri, and S. C. Burgess, "Prediction of peptides observable by mass spectrometry applied at the experimental set level," BMC bioinformatics, vol. 8, no. 7, p. S23, 2007.

[6] L. Nilse, F. C. Sigloch, M. L. Biniossek, and O. Schilling, "Toward improved peptide feature detection in quantitative proteomics using stable isotope labeling," PROTEOMICS-Clinical Applications, vol. 9, no. 7-8, pp. 706–714, 2015.

[7] H. L. Röst, T. Sachsenberg, S. Aiche, C. Bielow, H. Weisser, F. Aicheler, S. Andreotti, H.-C. Ehrlich, P. Gutenbrunner, E. Kenar, et al., "Openms: a flexible open-source software platform for mass spectrometry data analysis," Nature methods, vol. 13, no. 9, pp. 741–748, 2016.

[8] H. L. Röst, U. Schmitt, R. Aebersold, and L. Malmström, "pyopenms: A python-based interface to the openms mass-spectrometry algorithm library," Proteomics, vol. 14, no. 1, pp. 74–77, 2014.

[9] L. Martens, M. Chambers, M. Sturm, D. Kessner, F. Levander, J. Shofstahl, W. H. Tang, A. Römpp, S. Neumann, A. D. Pizarro, et al., "mzml—a community

standard for mass spectrometry data," Molecular & Cellular Proteomics, vol. 10, no. 1, pp. R110–000133, 2011.

[10] O. Kohlbacher, K. Reinert, C. Gröpl, E. Lange, N. Pfeifer, O. Schulz-Trieglaff, and M. Sturm, "Topp—the openms proteomics pipeline," Bioinformatics, vol. 23, no. 2, pp. e191–e197, 2007.

[11] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 315–323, 2011.

[12] O. Bousquet and L. Bottou, "The tradeoffs of large scale learning," in Advances in neural information processing systems, pp. 161–168, 2008.

[13] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, "A review on deep learning techniques applied to semantic segmentation," arXiv preprint arXiv:1704.06857, 2017.

[14] A. Karpathy, J. Johnson, F. Li, and S. Yeung, "Cs231n: Convolutional neural networks for visual recognition." `http://cs231n.stanford.edu`. Accessed: 2018-01-15.

[15] M. Kim, A. Eetemadi, and I. Tagkopoulos, "Deeppep: Deep proteome inference from peptide profiles," PLoS computational biology, vol. 13, no. 9, p. e1005661, 2017.

[16] K. Shinoda, M. Sugimoto, N. Yachie, N. Sugiyama, T. Masuda, M. Robert, T. Soga, and M. Tomita, "Prediction of liquid chromatographic retention times of peptides generated by protease digestion of the escherichia coli proteome using artificial neural networks," Journal of proteome research, vol. 5, no. 12, pp. 3312–3317, 2006.

[17] Y. F. Li, R. J. Arnold, H. Tang, and P. Radivojac, "The importance of peptide detectability for protein identification, quantification, and experiment design in ms/ms proteomics," Journal of proteome research, vol. 9, no. 12, pp. 6288–6297, 2010.

[18] H. Tang, R. J. Arnold, P. Alves, Z. Xun, D. E. Clemmer, M. V. Novotny, J. P. Reilly, and P. Radivojac, "A computational approach toward label-free protein quantification using predicted peptide detectability," Bioinformatics, vol. 22, no. 14, pp. e481–e488, 2006.

[19] C. Zhou, L. D. Bowler, and J. Feng, “A machine learning approach to explore the spectra intensity pattern of peptides using tandem mass spectrometry data,” BMC bioinformatics, vol. 9, no. 1, p. 325, 2008.

[20] N. H. Tran, X. Zhang, L. Xin, B. Shan, and M. Li, “De novo peptide sequencing by deep learning,” Proceedings of the National Academy of Sciences, vol. 114, no. 31, pp. 8247–8252, 2017.

[21] O. Ronneberger, P.Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in Medical Image Computing and Computer-Assisted Intervention (MICCAI), vol. 9351 of LNCS, pp. 234–241, Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[22] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” arXiv preprint arXiv:1408.5093, 2014.