KU LEUVEN

FACULTY OF SCIENCE

Machine Learning Analysis of STM Images of Self-Assembling Monolayers

Schuyler Hanson

Thesis presented in
fulfillment of the requirements
for the degree of Master of Science
in Chemistry

Supervisor: Prof. Jeremy Harvey
Theoretical and Computational Chemistry Group

Co-supervisor: Prof. Steven de Feyter
Molecular Imaging and Photonics

Academic year 2019-2020

Table of Contents

Abstract

This work presents two exploratory machine learning analyses applied to STM chemical imaging problems. A denoising autoencoder was applied to noised simulated STM feature images and denoised positions of features were recovered. In a separate result, a fully convolutional network (FCN-8) implementation is presented for performing image segmentation on small datasets of scanning tunneling microscopy images of molecular self-assembly. Datasets of two molecular systems, PCDA, DBA coadsorption (10,12-pentacosadiynoic acid, octoxybenzodehydro[12]annulene) and CN-PDI (cyano-substituted perylene diimide), were constructed and input to the segmentation model as raw experimental images. This model allows for an end-to-end training of raw STM images into segmented class regions. Successful segmentation was demonstrated and while the results are promising, they are mainly qualitative at this stage, and primarily limited by availability of data.

Acknowledgements

I would like to thank Prof. Jeremy Harvey for his agreeing to supervise this project, as this was a novel direction for both of us. I would like to thank him for our many discussions and his encouraging of my desire to pursue directions in science I find interesting. Additionally I'd like to thank Prof. Steven De Feyter and members of the MIP group for help gathering data and for having supervised me during my previous work with them. Lastly, I'd like to thank my family and my friends for their love, support, and curious minds.

Chapter 1 Introduction

1.1 General introduction

This work presents a first foray into an application of deep learning-enabled computer vision to analysis of chemical microscopy images. Using a relatively simple deep learning model, we wished to demonstrate the feasibility of computer-learned recognition of features present in a dataset of STM images. This project began with the idea that automated domain labeling of microscopy images would be both possible and attractive for imaging chemists and sought to establish a first result/proof of concept for establishing interest in machine learning tools applied to this problem domain.

An interesting frontier of science is the emerging ability to characterize and control nanoscale processes, offering a new avenue of possibilities for both basic scientific understanding as well as for bottom-up materials development [1]. Scanning probe microscopy (SPM) has emerged as a central tool for probing this new world of nanoscale physics and chemistry, with a main component of SPM dedicated to digital acquisition of images of scanned regions of interest. Generally in SPM, a probe scans a nanometer-scale discrete 2D grid and records physical quantities as a response to local conditions, which reflects chemical/physical properties of interest at the sampling locations. As we have recently entered an era defined by a marked uptick in both data acquisition and processing capabilities, novel opportunities exist for extracting physical and chemical information from ever-widening datasets [2,3].

Scanning tunneling microscopy (STM) is an SPM technique that can be used for image acquisition, nanomanipulation, and spectroscopic measurements, with the ability to simultaneously measure at the nanoscale and perform controlled local nanostructure manipulation. The KU Leuven Molecular Imaging and Photonics group (MIP) uses ambient-condition STM to image multi-molecular assemblies at the liquid-solid interface. A primary interest of the MIP group is in using STM to understand non-covalent molecular self-assembly processes, specifically self-assembled monolayers of organic molecules, investigating the relationship between molecular structure and assembled pattern as well as dynamics and controlled manipulation at the probe-sample interface.

STM images are electrical current maps as a function of local electronic density of states and probe height with respect to surface, which reveal a variety of structural chemical information via contrast between features, visible as for instance distinct molecular/atomic domains/lattices, lattice defects, step edges, etc. A typical STM experiment involves the collection and hand-labeling of a set of images to be used to make statistical statements about chemical features present in the data. Typically within the MIP group, enough images are collected to account for all of the possible variation in behavior in a chemical dataset, which usually results in small to mid-sized datasets.

Machine learning (ML) is the empirical computational learning of functional approximations to various data modeling tasks. This functional approximation is used to learn a mapping between inputs and labeled data. In this way the machine learns a model of data such that predictions or decisions can be made without explicitly programmed instructions.

Typically machine learning problems are of the format (model, optimization algorithm, data) with the model being a feature extractor in the case of neural networks, and the optimization algorithm the chosen way to represent and minimize an empirical loss function, defined loosely here as the elementwise distance between a label vector and the machine-predicted vector approximating the label. With the recent resurgence of interest in machine learning, and particularly deep learning, a variety of scientifically-interesting tasks can be designed and successfully implemented with ever-increasing facility as this interest has resulted in user-friendly frameworks scientists have been quick to adopt [4,5]. Many chemistry problems are formattable as deep learning problems [6].The main interest of deep learning methods is the generality of problem they may be formatted to apply to, yielding impressive, often state-of-the-art results across many domains [4,7,8]. Famously, deep learning has provided ground-breaking advances in many computer vision tasks through the implementation of convolutional neural network-based architectures [9,10].

This thesis demonstrates a use of a machine learning model to automate labeling of molecular domains in experimental raw STM chemical image data. A fully convolutional network (FCN) was chosen to perform the computer vision task of image segmentation, wherein a dense mapping is constructed between a microscopy image and a corresponding image of labeled pixels, resulting in an approximated image of labeled pixels. Image processing and construction of the machine learning model was conducted using the Python programming language and its Keras library [5]. STM image datasets of self-assembling organic molecules were used. While demonstrating the ability of a trained computer to perform automated classification to some degree, insufficient data existed within the MIP group to construct interesting datasets for this task or characterize model performance. It is unknown how well this model could be trained to perform with a more appropriate dataset. Therefore while this result demonstrates applicability of machine learning methods to computer vision problems typical of microscopy or imaging science, broad, general applicability to automating microscopy image analysis workflows remains unexamined.

At this stage, a model was selected that is not fully compatible with the image analysis task for the bulk of the available data in the MIP group. The chosen model was selected because it corresponds to an architecture that underlies much of the recent progress in deep learning image segmentation [11].

1.2 Introduction to the application of deep learning to computer vision

Neural networks (NNs), a subfield of artificial intelligence and machine learning, are a now popular approach to pattern recognition problems. They have been demonstrated to perform very well on a diverse range of tasks, ranging from computer vision to speech recognition to image/waveform generation to other fields of inquiry [7,8,12]. NNs were invented in the 1940s-50s with the advent of interest and ability to begin mathematical and computational modeling of the human brain. The 1980s discovery of an algorithm for backpropagation, allowing for efficient optimization of multilayer networks [7], coupled with the discovery of the universal approximation theorem [13] of neural networks, stimulated interest in neural networks. The universal approximation theorem states that a single hidden-layer neural network containing a finite number of neurons can approximate any continuous function of an n-dimensional input variable when using a certain class of activation functions [14].

A 2012 result reinvigorated interest in deep learning, wherein a deep convolutional network was successfully used to obtain state-of-the-art results on the Large-Scale Visual Recognition Challenge (LSVRC), almost halving the prediction error from 28% to 16% [10]. This result used ReLu activation functions, data augmentation, a regularization technique called dropout – used to introduce sparsity in connections to prevent overfitting, and was trained on parallel instances of graphical processing units (GPUs), the general use of which has since become a mainstay for training networks with many parameters [7]. GPUs are not necessary for models with a smaller number of trainable parameters, as is the case in this work, having used an encoder network with frozen pre-trained weights.

One of the main fields of revolutionarily successful application of neural networks is in computer vision. Computer vision is a discipline concerned with extracting semantic information and scene understanding from digital images. Problems that often required unique featurizations of objects of interest and or other image processing, such as outputting pixels belonging to a certain pattern at multiple orientations, positions, and scales, are now able to have this featurization automated by use of convolutional neural networks (CNNs) [15,16] which learn the feature extraction process over iterative learning cycles, termed epochs. To illustrate this point, in traditional computer vision workflows, manually selected and parametrized feature extractors, such as blob or edge detectors or histogram of oriented gradients (HOG) feature detector are used to detect sharp/characteristic gradients in order to highlight a desired feature [17,18].

Feature selection has largely been replaced with automated learning of the relevant features in an image such that non-experts can often achieve significant results by merely designing an end-to-end mapping and selecting an appropriate model [19]. Deep learning can be used in an end-to-end manner, such as is done in this thesis, replacing the need for stacking of a complex series of manually-designed image processing and feature extraction modules. Perhaps most promising is the generality of tasks to which various CNN-based models may be applied to achieve state-of-the-art results, all while retaining the same basic idea of stacked convolutional

modules, with a relative minimum of pre- and post-processing techniques needed. A model architecture is now dependent on a task and corresponding high-level architectural ideas, and not on the data, and so may be retrained on a custom dataset dissimilar to any previous data, as is demonstrated in this work.

Chapter 2 Technical Background

2.1 Scanning Tunneling Microscopy and Self-Assembling Monolayers

2.1.1 Scanning Tunneling Microscopy

Scanning tunneling microscopy (STM) is a type of scanning probe microscopy that uses a scanning conductive probe that tunnels current across a gap between itself and a conductive surface when a bias voltage is applied to shift the relative Fermi energies of the two materials. This produces empty states in one material that filled states in the other can tunnel into, direction of tunneling depending on the sign of the bias [20]. This can be used to acquire a 2D map of LDOS variations for the section of surface under consideration . Commonly used for surface characterization of metals and semiconductors, this technique is also widely used for the characterization of molecular adsorbates [21].

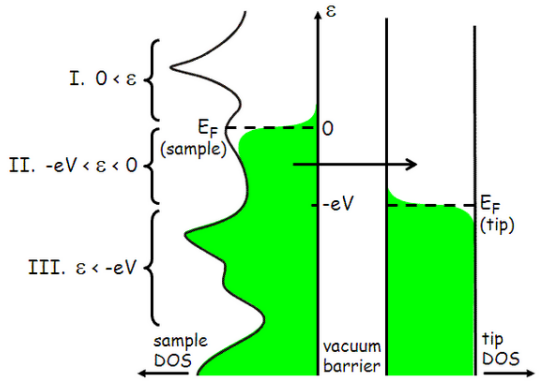

Fig. 2.1.1: Schematic overview of tunneling between electron states. Here electrons tunnel from sample to tip as the sample is negatively biased with respect to the tip.

There are two modes in STM imaging: constant-current with height adjustments and constant-height scanning, which records changes in current. In constant-current mode, feedback control is used to adjust tip height such as to obtain constant tunneling current. Constant-height mode involves scanning a vertically static tip over the surface and recording current changes due to varying LDOS and sample topography as the signal. Constant-current being frequently used for chemical imaging, these experiments provide topographic height maps of chemical features on the scale of angstroms.

It may be shown that tunneling current is strongly dependent on angstrom-scale perturbations in the z direction, and presents surface topography resolution of at least 0.1nm in

the z direction [22]. STM tunneling current in constant height mode maps constant LDOS contours [22]. Care must be nevertheless be taken in the interpretation of STM images for chemical information, as the resulting image could be a convolution of orbitals contributing to the tunneled LDOS signal [21].

2.1.2 Self-Assembling Monolayers

One of the main focuses of the MIP group is studies of small organic molecule self-assembly on various substrates, using STM to characterize nanoscale monolayered network structures. Self-assembly is the formation of large-scale ordered networks resulting from local functional group interactions. In organic molecules, these are often hydrogen bonding interactions between such groups as carboxylic acids and aliphatic van der Waals interactions between long alkyl chains with each other and or substrate, as in the case of graphene (HOPG) [23].

Interest in self-assembling monolayers largely stems from a desire for basic understanding of and control over the processes governing nanoscale intermolecular interaction and organized supermolecular network formation. Due to their ease of experimental access as well as the large amount of tunable features, organic molecule self-assembly is an ideal system for studying bottom-up approaches to patterned nanoscale materials [24]. Additionally, due to the high degree of organization of SAMs, they are inherently attractive from a functional materials point of view, for instance, in functionalizing surfaces for molecular electronics.

STM is commonly used to study molecular self-assembly in the MIP group. Formation of large-scale organized supermolecular networks is enabled by mid-strength substrate-adsorbate interactions: the adsorbate must outcompete the solvent molecule and have some affinity for the substrate, otherwise resolved images of chemical structure will be impossible to record, however, the interactions cannot be so strong as to prevent dynamic adsorption-desorption processes that allow for defect repair and the formation of large-scale ordered networks [51]. In fact, the requirement of having dynamically-changing surface structure that avoids defect-trapping and thus progresses to self-assembled networks is a key component of what makes ambient-condition STM a prime tool for the study of molecular self-assembly [51].

SAMs can have a wide variety of tunable parameters and features to select for. Chemical structure of adsorbate may be tuned as in for instance selecting the chain length of an alkyl moiety to select for stronger adsorption to a graphene substrate. Sterically-bulky groups can be added to a molecule to result in entirely new self-assembled patterns (fig. 2.1.2) [51]. Head and tail groups may be tuned to result in multilayer formations [24].

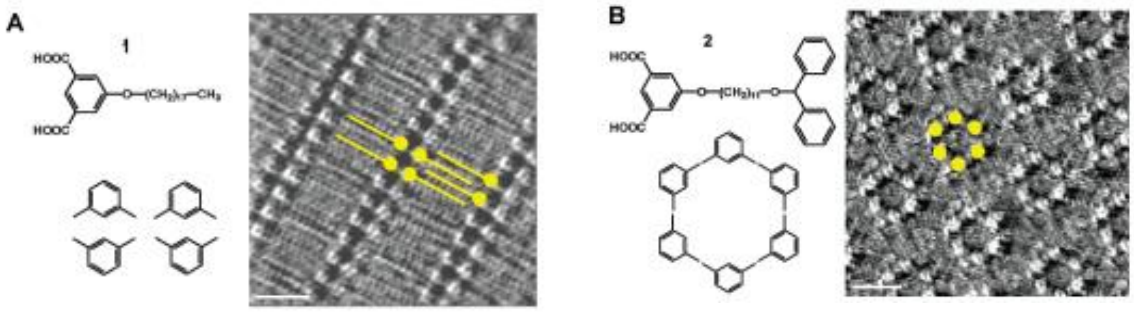

Fig. 2.1.2: Replacement of a long alkyl chain which self-assembles as a stacked lamellar structure with a benzhydryl group results in rosettes. The molecular structure schemas below the molecular structures demonstrate the respective carboxylate H-bonding geometries characteristic of these domain motifs.

2.1.3 Datasets used

The machine learning model was trained on datasets created from STM experiments on two different molecular systems.

The first was a small dataset resulting from one day of unusually high-quality acquisition in the MIP group's lab. This data was obtained in the larger framework of testing over various concentration ranges of two species on bare highly-oriented pyrolytic graphite (HOPG) to find suitable concentration ranges for testing for control over selective adsorption using nanocorrals as a continuation of work by Verstraete et al. [25]. The species are PCDA (10,12-pentacosadiynoic acid) and DBA-OC_8 (ether derivative of dehydrobenzoannulene). These are images of varying size scales. This bicomponent mixture was prepared via a 250:1 concentration ratio of PCDA:DBA with PCDA concentrations being 10^{-3}M. For the preliminary result demonstrated in this thesis, only 5 PCDA, DBA comixture images were used in training the end-to-end segmentation model. PCDA forms lamellar domains in 6 possible orientations as guided by van der Waals interaction between the long alkyl chains and HOPG lattice interstices and the symmetry axes of HOPG [25] DBA-OC_8 forms hexagonal networks.

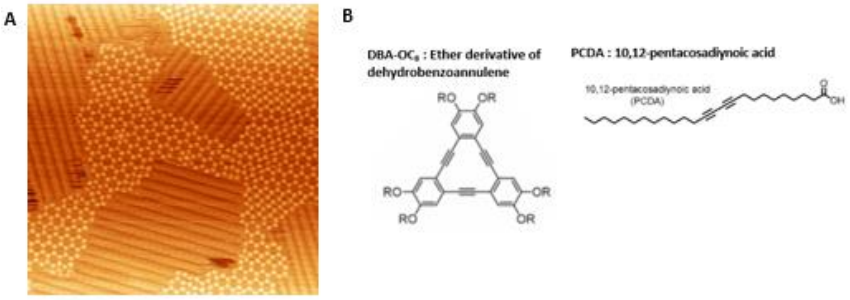

Fig. 2.1.3.A : a) 100x100 nm² image of DBA and various PCDA domains; b) DBA-OR, PCDA molecular structures

The second molecular system used was of CN-PDI (cyano-substituted perylene diimides). This dataset had 3 concentration ranges [1 mg/mL, 10X dilution, 50X dilution]. Only 1 mg/mL and its 50X dilution yielded self-assembly into distinctly oriented domains, therefore these were the only useable datasets for this molecular system. Two separate datasets were constructed and trained on, of 3 and 8 images respectively.

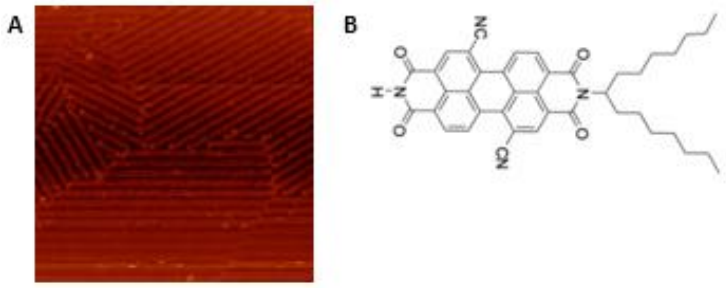

Fig. 2.1.3.B: 100x100 nm² CN-PDI 1mg/mL

In the case of multicomponent self-assembly, the De Feyter group's main focus is cases of coadsorption, with phase-separated systems being secondary in interest or often the result of failed experiments. Coadsorbed systems present cointegration of the components as in host-guest systems for instance, whereas phase separated systems present separate, non-overlapping, monocomponent molecular domains. However, phase-separated systems are ideal for image segmentation as with these, labeling of every pixel with bounding polygons is a feasible task. Whole-image labeling is possibly not directly feasible in the case of the rosettes in (fig. 2.1.2) for instance.Coadsoprtion would perhaps be better addressed by other computer vision approaches not included within the scope of our study. Therefore, the amount of appropriate data obtainable from the MIP group was quite limited.

In machine learning, the features learned depends on the data shown to the model. Therefore care must be taken to create balanced datasets representative of all the desired features one could think of both in terms of actual object features and recurrent image artefacts. Similarly, model performance is best characterized on a balanced test set.

2.2 Machine Learning

Machine learning is the empirical learning of a functional approximation to data. Generally speaking, a supervised machine learning model is selected and the parameters of this model are trained on a "training" subset of the data, and this model is used to generalize to a new previously unseen "testing" subset of the same data. Supervised machine learning, which

involves learning an input-output mapping on a labeled dataset, is currently the dominant paradigm for machine learning, with most applications using supervised models.

An overview of the basic artificial neural network architecture, the multilayer perceptron, will be given as pertains to the concepts drawn upon in the neural networks used. A brief discussion of convolutional neural networks follows.

2.2.1 Artificial Neural Networks

Artificial neural networks are a type of machine learning model consisting of a computational graph, consisting of neurons (nodes) and weights (edges), and an optimization algorithm. The computational graph is a set of layers of nodes with each layer being associated with an activation function acting upon the weighted output from the previous $k\text{-}1$ layer neurons connected to a j neuron in the next k layer (fig. 2.2.1.A). Activation functions serve to apply nonlinear transformations to an input scalar product between incoming $k\text{-}1$ neural activations and their weights of connection $w_{ij}^{(k-1)}$ for all the $i^{(k-1)}$ neurons connected to neuron $j^{(k)}$. A neuron's "activation" – or state – therefore refers to the output of a nonlinear function on incoming weighted neuron contents, which then becomes the numerical value of the $j^{(k)}$ neuron. Generally, all the activations for the next layer are updated in a feedforward pass through the network. A bias neuron $b^{(k)}$ in layer k is used as a thresholding agent for determining the value which the sum must exceed in order for the activation to survive propagation through the rest of the network. Otherwise the activation dies and with it the information it carried.

A deep neural network is a stacking of hidden layers, which usually outperforms single-layer/shallower networks [26] until 4 or so layers. Conversely, convolutional neural network performance (the main architectural component of the model used) generally improves with increasing network depth.

The activations in the $n\text{-}1$ hidden layer are then passed through a final activation function to a target output layer n, and this vector is used to compute the loss, or error of prediction, when compared to the corresponding training label vector of the input data. The basic network topology and concept of activation functions are well-summarized in the following image of a single hidden-layer, feedforward neural network (fig.2.2.1.A). This example NN is fully connected, in that all of the nodes between adjacent layers are connected such that $w_{ij} \neq 0 \; \forall \, i,j \in k,k\text{-}1$. The input layer for machine learning on images is often a flattened image vector, where each neuron in the input layer is a pixel intensity value.

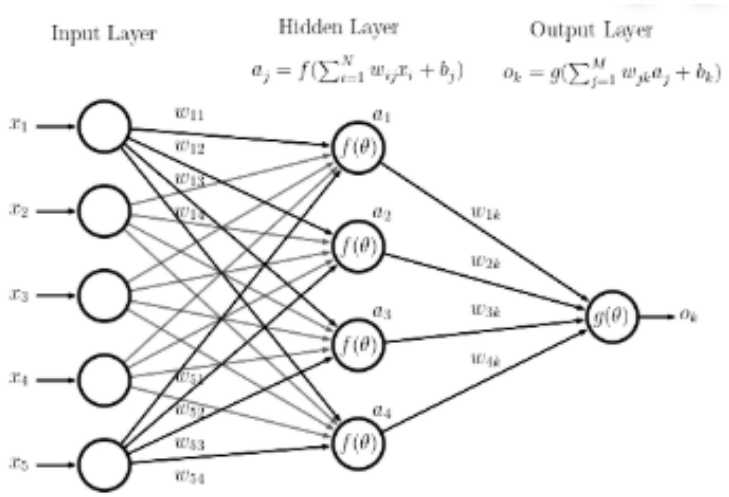

Fig. 2.2.1.A: Schema generically demonstrating information transfer through an ANN

Training epochs are discrete time-steps *t* over which the network is trained on a selection of input data [27]. The initial weights of the network are randomly chosen. This training corresponds to a feedforward pass through the network from an input to a predicted output, followed by the calculation of the loss function between predicted outputs and labels, and finally gradient descent using the current state of weights at time *t* via calculation of the direction of change of the weights that corresponds to minimization of the loss function [7]. The forward and backwards processes correspond to one epoch such that the state of the network is updated: network(t) → network(t+1).

Activation functions are used to introduce the possibility of learning nonlinearities in the network. This can be understood as nonlinear transformations of data manifolds such that data points are (eventually) mapped to output spaces where the data classes are linearly separable (Fig. 2.2.1.B).

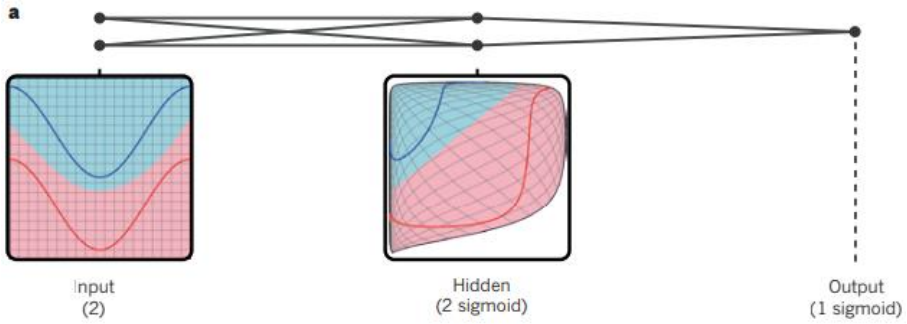

Fig. 2.2.1.B: Mapping input data to linearly separable space via activation functions where linearly-separable classification occurs in the output vector space

.

A popular activation function used is ReLu (rectified linear unit), which kills negative activations and returns positive activations. ReLu has been found to outperform other activation functions in deep neural networks, and is the default activation function between

layers in many modern architectures [15]. ReLu has many advantages including sparsity of representation in that it's able to output a true zero value and its computational simplicity when compared to other activation functions involving more complex calculation such as sigmoid or hyperbolic tangent functions]. Sparsity increases the robustness of the model against small changes in input, and sparse representations are more likely to be linearly separable and force the model to learn a more efficient representation of the input data [28]. It has been shown that ReLu activation functions are particularly appealing for deep learning models, where the number of hidden layers is 3 or more [28,14].

$$ReLu(x) = max(0, x) = \begin{cases} x \; if \; x \geqslant 0 \\ 0 \; if \; x < 0 \end{cases}$$

Another important activation function used in this work is the softmax function, which maps a non-normalized vector to a normalized classification probability distribution (fig. 2.2.1.D). Softmax and ReLu are used in different contexts. ReLu allows for efficient training of hidden layer units whereas softmax acts on the *k-1* inputs the output layer receives.

$$softmax(\mathbf{x})_i = R^K \to R^K : x \mapsto \frac{exp(x_i)}{\sum_j^K exp(x_j)} \; for \; x_i \in \mathbf{x} = (x_1, \ldots, x_K)^T$$

Above shown is the mapping to the *i*th component of the output vector. The softmax normalizes a *K*-dimensional vector into the range *[0,1]*, thus forming a probability distribution over the *K* components: $\Sigma_j^K P(\mathbf{Y}=y_j \mid \mathbf{X}=\mathbf{x}) = 1$. Setting the dimensions of the vector to *K* dimensions that feeds to the softmax function for a *K*-dimensional classification problem results in a probability distribution over the classes. An important note here is that this scheme is enabled by the so-called "one-hot" vector encoding of the labels. An illustration of this scheme is as follows for the example of encoding numerical labels from 0-9 as would be used to generate labels for image-level (i.e. mapping from an input image to a single classification label) classification of handwritten digits (fig. 2.2.1.C). As can be seen, this similarly generates a probability distribution over classes, and the probability function is *1* for $P(\mathbf{Y}=y_{true} \mid \mathbf{X}=\mathbf{x})$ and *0* elsewhere.

Y=y	*one hot(y)*
1	[0,1,0,0,0,0,0,0,0,0]
9	[0,0,0,0,0,0,0,0,0,1]
0	[1,0,0,0,0,0,0,0,0,0]

Fig. 2.2.1.C: One-hot encoding transformation scheme

The optimization algorithm typically is a variant of the backpropagation algorithm. Backpropagation allows efficient computation of the gradient of the loss function with respect to connection weights by computing this gradient in a backwards manner, which results in more efficient computation by using chain differentiation. In practice, the details of the optimization algorithm did not have to be understood to obtain the preliminary successful results of this thesis, as the popular Adam optimization algorithm [29] was used and shown to be sufficient for the purposes of this work. However, a discussion of loss functions is essential, as in designing a neural network for a problem, one must choose the correct loss function corresponding to the type of problem at hand. A common loss function for regression is least-squares error (LSE), where the loss is minimized for a parametrization of f that returns values closest to the values of the true labels y_i.

$$L(w(t)) = \frac{1}{2}\sum_i (y_i - f(x_i; w))^2$$

For classification problems, categorical cross-entropy loss is a commonly used loss function that measures the prediction error for multiclass classification problems. A softmax output layer and categorical cross entropy loss are commonly used in tandem for multiclass classification. This is the scheme used in this FCN implementation.

A good definition of cross entropy loss is given by Wikipedia [50]: "*the cross entropy between two probability distributions* p *and* q *over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set if a coding scheme used for the set is optimized for an estimated probability distribution* q*, rather than the true distribution* p." A classification machine learning algorithm predicts a probability distribution of classes for a given input data, $q(\boldsymbol{y}_{est} \mid \boldsymbol{x})$ and this is compared with a model's output probability distribution of labels $p(\boldsymbol{y}_{true} \mid \boldsymbol{x})$. It can be seen that an estimated probability distribution equal to that of the true distribution causes $H(p,q) = 0$. The y_i components are individual classes within the probability distribution vector of classes $\boldsymbol{y}$.

$$H(p(y \mid x), q(y \mid x)) = -\sum_{y_i \in y} p(y_i) log(q(y_i))$$

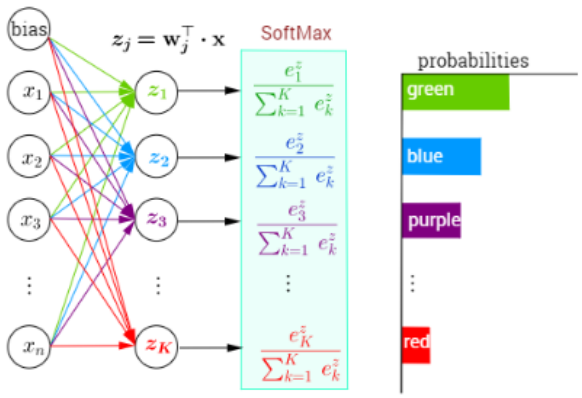

Fig. 2.2.1.D: Schema illustrating the n-1 layer feeding to the n output layer normalizing its activations to a probability distribution vector over classes, which becomes the output layer

In application, the model returns a predicted probability distribution with softmax and the categorical cross entropy loss is calculated by comparing this predicted distribution with the true probability distribution for a given input. Higher variance between the two distrubitions leads to increasingly large values of the loss function.

2.2.2 Convolutional Neural Networks (CNNs)

The architecture of CNNs greatly improves prediction efficiency and accuracy on 2D data containing local patterns/features [15,19]. For example, a core feature of CNNs as will be seen, involves the model learning features on a 2D image by sliding learnable filter kernels over the image. This outputs large activations to hidden layer neurons connected to regions of the network that contain the same feature.

The CNN architecture underlies the extensive progress of deep learning in computer vision. The core instrument in deep CNNs is the use of stacked learned convolutional filters, which results in learned feature maps corresponding to the features that correspond to target outputs. In the convolution of an image with a filter, a kernel (2D image filter of size $n \times m$ *pixels*, $n,m < N,M$) is passed sequentially over an input image of size N,M pixels, and an averaged scalar product is computed for all the pixels within the receptive field (i.e. the target region actively convolved by a filter at any given step of the convolutional process). The convolution of a kernel with an input receptive field is illustrated (Fig. 2.2.2.A). Filter kernel parameters are adjusted manually in traditional computer vision/digital signal processing so as to achieve the desired effect: blurring, edge detection, corner detection, etc [30]. In CNNs, the values of the filter parameters are learned during training and in this case correpsond to feature detectors. A heirarchy of features is learned, analogously to how the human brain processes images, resulting in differential detection of low-, mid-, and high-level featues depending on depth of the network [19]. Deeper networks have the ability to learn multi-stage, complex representations of features, which generally gives them an advantage over shallower networks [26].

Convolutions typically are packaged in modules, with a typical CNN module comprising a convolution layer k with $dim(kernel \in \{kernels\}) < dim(input)\ \forall\ kernels \in k$ on the input vector followed by ReLu and max pooling. Training occurs once the computational graph of convolutional modules has been assembled. The number of filters per convolutional module is human-specified (known as channel depth), as well as other parameters related to the convolutional process, such as sampling frequency and size of filters.

Additional to feature detection, a core characteristic of CNNs is a subsampling of the original image. Each layer returns a set of smaller-dimensional representations of the the set of its input receptive fields. Subsampling is native to the convolutions but also occurs in the use of the max-pooling operation, as will be subsequently discussed.

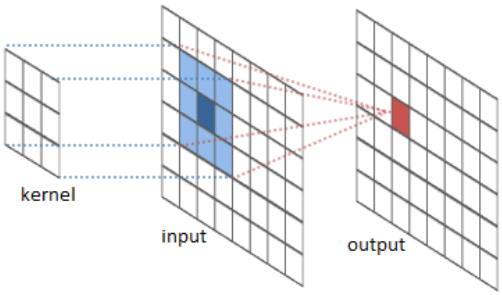

Fig. 2.2.2.A : Illustration of convolution operation: a 3x3 kernel passed over a 3x3 input region (receptive field) resulting in a scalar product mapping to a single pixel in the output filter map

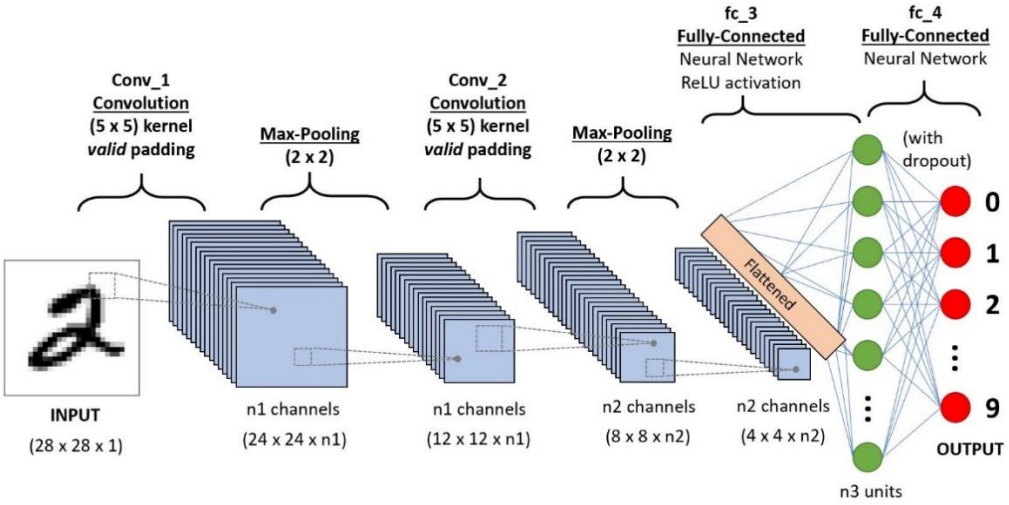

Fig. 2.2.2.B: Example CNN architecture illustrating downsampling via convolutions followed by two fully connected layers to the output classes for the problem of classifying handwritten digits data

The basic CNN implementation architecture is in the form of image-level classification networks (Fig.2.2.2.B) [31]. In these, hidden layers extract and subsample features down to a specified final feature map size. Then these final feature maps are unrolled and concatenated into a 1D vector, which is fully-connected to the output layer *n* of class labels. Therefore every neuron in the *n-1* layer has a weight and thus a vote. Certain weight patterns are learned on this fully-connected layer to correspond to maximal probability votes in the output classes.

The stacking of learned feature maps is a process of subsampling the original image space, which results in the learning of increasingly global features. This can be understood as a result of the sparsity of the connections in CNNs (fig. 2.2.2.C), where the inputs to any *k*th hidden layer neuron are only local groupings of neurons (i.e. receptive field in layer *k-1*) [15]. Therefore, as one progresses in hidden layers, one learns increasingly global features as higher-level feature representations are mapped to output representations represented by whatever activations in these deeper layers that correspond to minimized loss. The number of learned representations (channel depth) in a given layer is a user-controlled parameter in selecting the

number of filter kernels. The number of representations to learn can be used as a bottleneck to constrain the model at a certain point.

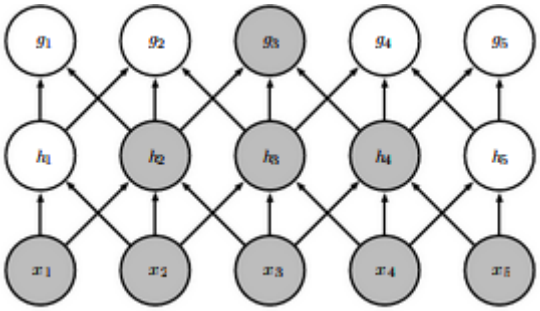

Fig. 2.2.2.C: Sparsity of connection are a key advantage of CNNs over fully connected networks. This has consequence of only groupings of local pixels being passed to subsequent layers' neurons.

Training efficiency gains in CNNs come from the weight sharing and sparse connectivity that occurs in convolution of a filter over all the receptive fields of an input image [15]. As can be seen in (Fig. 2.2.2.C), neurons of adjacent layers are only locally connected, as opposed to fully connected. Fig. 2.2.2.C depicts a single channel, where each activation in the 2nd, 3rd layers results from a receptive field of size 3 neurons. There three edges incoming from a receptive field will always have the same weights of connection, as the same filter is convolved over the entire input image vector. Generally, the kth module's output filter maps are treated as image vectors to be convolved with N filters in layer $k+1$.

As mentioned above, CNN modules are often composed of Conv → ReLu → Pool operations. Pooling is a subsequent subsampling operation that results in invariance to small translations in inputs [15,19]. Max pooling is a type of pooling that takes as input subfields of a filter map and returns the maximal activation of that subfield, as a summary statistic for a local, connected grouping of pixels. As described in Goodfellow et al. [15], combining various feature maps - each returned by different filters - as inputs to a max pooling function can result in a network becoming invariant to different types of transformations that can occur.

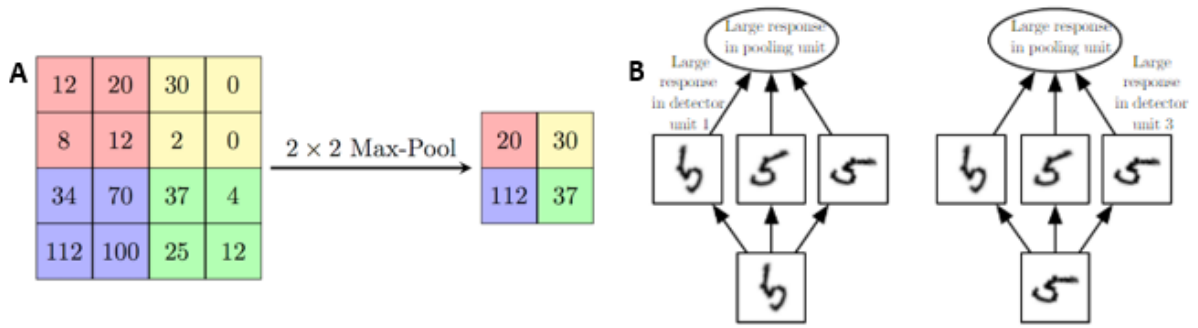

Fig. 2.2.2.D: A) Maxpooling operation example; B) Example of pooling used to learn rotational invariance downstream of different filters returning different filter maps depending on object orientation

The widespread success of CNNs in computer vision mostly results from the flexibility with which the basic features and advantages of convolutional layers may be generally applied to a wide variety of novel task-specific architectures for various computer vision tasks. This work is one such example. Here, I implement a fully convolutional network (FCN) for image segmentation, which is the task of pixelwise classification. As will be discussed later, the FCN architecture is the use of a pre-trained downsampling network made of the convolutional layers described above combined with an upsampling deconvolutional network.

2.2.3 Image Segmentation Task

Imagine a camera installed on a street corner programmed to record frames every 15 seconds. Perhaps comparing two images spaced 15 seconds apart, we see two different sets of cars yet the same person walking on the side, albeit occupying a different position now and thus different pixels, and say perhaps there's a cloud now so the general image brightness distribution is now modified. Some objects remain more or less positionally invariant: lampposts, trees, etc. Image segmentation is the task of automatically labeling each of these pixels as belonging to a certain class, often in a probabilistic context, as there's noise and other sources of variation across images. In the case of the model used in this work, the output of the segmentation model is an image where each pixel is labeled as belonging to a class of maximum probability from an output pixelwise probability distribution.

Image segmentation was chosen as it seemed like a good place to start for a first use of CNNs, which was ultimately the goal of this project. Naively, also this is what naturally comes to mind when faced with an image and asking what kinds of classification tasks could be performed on it, as it makes sense that one would like for every pixel in the image to be labeled. In practice, the labeling of each pixel is a tedious task and not suited to image datasets requiring hand-labeling of a large number of separate object instances.

Other approaches to image segmentation include thresholding (e.g. conversion of greyscale image to binary image based on a threshold value), region growing, region split and merge, and segmentation based on machine learning approaches such as k-means clustering or weakly supervised learning [18]. An important distinction, again here, is the necessity of careful parameter tuning versus the ability to construct an explicitly semantic correspondence between various object instances and their corresponding class labels in the case of neural networks. Thresholding and methods using this basic concept, for instance rely on similarity between neighboring pixel values, but artefacts such as shadows can easily throw this off and in any case the segmented regions do not correspond to any explicit semantic information. Therefore the choice of classical segmentation algorithm is highly dependent on the specific dataset, and often for reasons that do not correspond to semantic features. In a well-tuned deep learning model trained on an adequate dataset, deep learning has been shown to outperform traditional computer vision approaches [18,34].

Probabilistic graphical models are another approach to image segmentation that exploit complex contextual dependencies between labels to assign class probabilities to a pixel based on learned relationships to a neighborhood of pixels [32]. The FCN and other CNN architectures have been used in conjunction with conditional random fields (CRFs) to result in performance improvements in fine-grained segmentation accuracies [33,34]. An example novel application of these combined methods is in Ziatdinov et al. (2017) [35]. Investigating STM images of bowl-shaped sumanene molecules, they combined positional correlations between molecular classes (bowl up/down, rotational) as a Markov random field, and CNN features of rotational states to establish correlation values of rotational states with neighboring bowl up/down conformations and propose a reaction path for bowl inversion and its effects on neighboring molecules. While medical [36] and engineering [37,38] imaging communities seem eager to adopt machine learning methods for automatic image recognition tasks, there seems to be a relative scarcity of research done on chemical image deep learning image applications such as performed by Ziatdinov et al. in an application of FCN for chemical identification of features on scanning tunneling electron microscopy images [39].

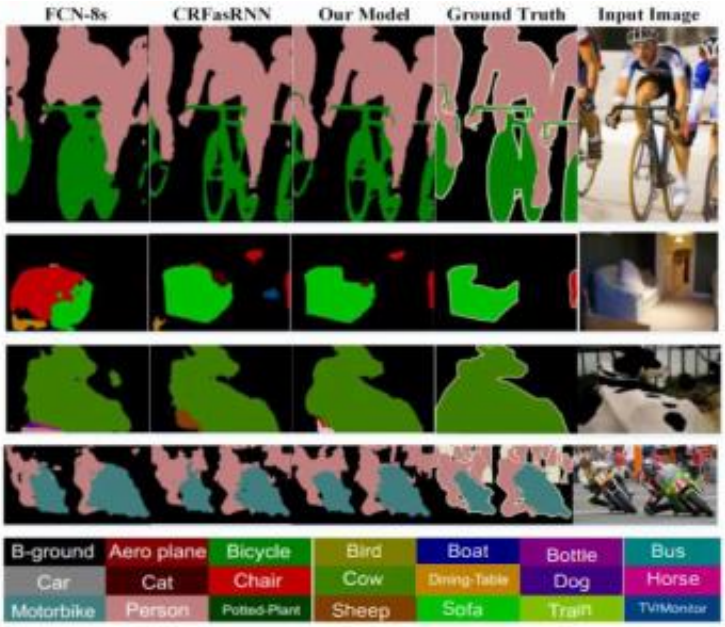

Fig. 2.2.3: FCN-CRF model performance compared to FCN-8 and other architectures

2.2.4 Autoencoders

A class of useful examples of relatively simple neural networks are autoencoders. These are single- or multi-layer neural networks that are trained to encode lower-dimensional representations of data and then map to reconstructions of the data [40]. This is achieved by providing original/denoised data as outputs. Autoencoders can provide high-fidelity reconstructions of noised data. The hidden layer(s) receive inputs which are then passed through a representational bottleneck in the form of hidden layer(s) with less units than the input layer, which forces the network to learn a compressed, more efficient feature representation. This is then passed to an output layer with an appropriate activation function for image reconstruction. Autoencoders can be used as a feature extraction step before training on deep CNNs, or other separate models.

As an example application is an autoencoder trained to learn a compressed representation of plasma emission x-ray spectral data. An 8-hidden-unit autoencoder was used to learn an 8-dimensional compressed representation that matches the output of a physics model [50,4].

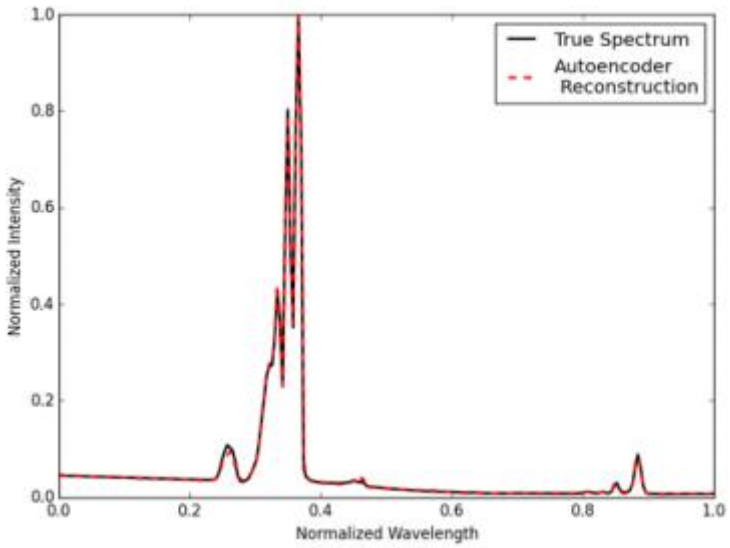

Fig. 2.2.4: Autoencoder learning principal, low-dimensional features of x-ray emission data.

Chapter 3 Implementation Outline and Details

3.1 Creating Dataset for Input into Model

In the supervised learning paradigm, a dataset of pairs $\{(x_i,y_i)\}$ of training points xi and corresponding label points/annotations y_i is used to train the model. Operating iteratively over batches of pairs (taking advantage of ability to compute in parallel to a certain degree), the model adjusts its weights via an optimization algorithm. Again, this is done by improving feedforward network predictions over discrete training steps on these batches of data. This generally results in improving parametrizations that return increasingly accurate generated predictions.

In the FCN model applications, we used raw, untransformed experimental STM image data. One of the advantages of neural networks, as previously described, is the ability to use raw data as is, without the need for complex manual preprocessing/featurization. It's not always the case that one may simply use raw images. Model performance on image data can often be improved by augmenting the dataset [10], which means creating new images (as perceived by the computer) using the images already available, by applying rescalings, rotations, croppings, etc. It is however generally the case that neural networks generally rely on less pre and post-processing of data, and present in any case an avenue for raw input to output mapping design. In this case, the model was trainable on unmodified training images (x_i). This is interesting as this provides the opportunity to see how well model performance directly generalizes to various cases one might encounter in STM such as differences in scan area size, probe drift, and random regional brightness artefacts.

While a comprehensive study of the effect of variability across images on was not able to be undertaken at this stage, the datasets trained upon did reflect some of this variability.

To generate a dataset of label images corresponding to each training image, we first used an online annotator [41] which allowed hand-drawing polygons around regions of interest in the STM images. Downloading these annotations, we wrote a python program that converted a file containing lists of vertices of these polygons into segmentation masks, meaning a set of distinct numerically-featurized pixelwise class values (for example PCDA=1, DBA=2, Noise=3), which serve as segmentated labels for training. For each polygon, every pixel within that polygon was mapped to the class-specific value when provided with a class label for that polygon. Since the source of this labeling is manual, there arises the case when pixels are not captured by any hand-drawn polygon. If left as is this would be interpreted as a distinct (numerical) class label and the network thus trained to map these pixels to a class that doesn't exist. It's hard to estimate the

magnitude of error this would introduce, but to be sure, unlabeled pixels were iterated over and randomly set to the first nonzero neighboring pixel value in a concentric searched ring. Since the scale of feature resolution is likely much larger than the size of these unlabeled areas, incorrect assignment of neighboring pixel values probably introduces negligible error in the infrequent cases this occurs.

Running this program converting lists of vertices into segmentation masks, transposed versions of the original raw images were returned. As a result, the training library was constructed from transposes of the original data.

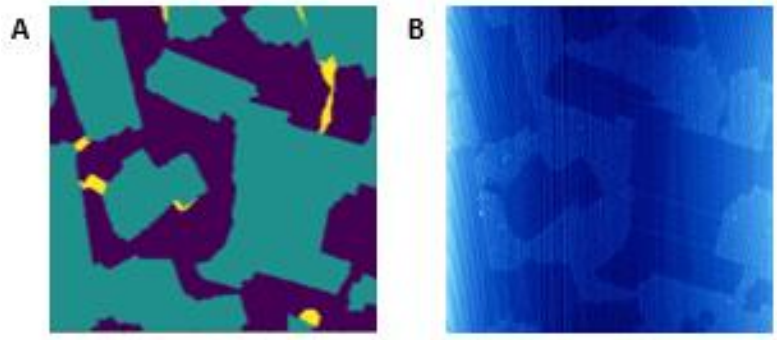

Fig. 3.1: A) Annotation image with distinct class-specific pixel masks B) Corresponding STM image

3.2 Fully Convolutional Network

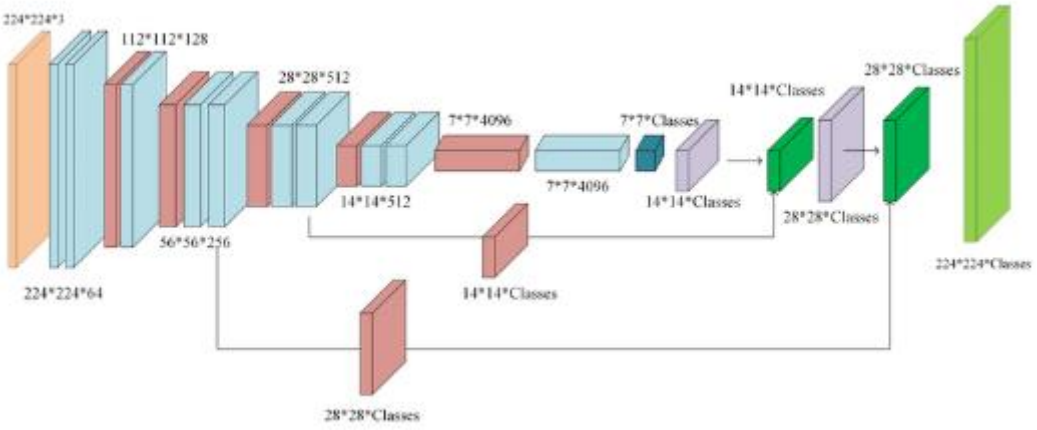

Fig. 3.2.A: FCN-8 architecture

(N.B. for explanation of image/tensor dimension notation used in this section, c.f. appendix)

The architectural concepts underlying much of the recent approaches to image segmentation networks may be found in fully convolutional networks. An FCN implementation

(FCN-8) was chosen as it's a relatively simple network architecture that performs end-to-end image segmentation. The publication reporting its invention (2015) [42] presented several architectural components that recur as a dominant theme in deep learning segmentation networks, as well as a performance benchmark on a computer vision image dataset both in terms of speed and accuracy. The main features of the FCN model are the fact that it's fully-convolutional, has an encoder-decoder architecture, the outputs are structured as segmentations, and the skip connections introduced to augment prediction accuracy on upsampled segmentation outputs. CNN-based segmentation has the issue that image-level classification CNNs are not natively appropriate for segmentation outputs. The FCN publication presents the removal of the fully-connected classification layers of a downsampling CNN, and appended to the outputs of the downsampling path, an upsampling convolutional path for spatial reconstruction of segmented features.

The FCN architecture consists of an encoding network and a decoder network. The encoder network functions to learn a compressed representational space of salient features and the decoder network reconstructs the image. The encoder network is often a pretrained deep CNN whose weights are downloaded and used as constants. Transfer learning, i.e. use of a pretrained network, is common in problems where data availability is limited [43]. The pretrained CNN is often a very deep network trained over many class instances and thus very likely to contain some degree of useful representational information in its learnable activations that can be generalizeable to be used on new image data. We used vgg16, a deep CNN trained on 20,000 classes for the pretrained encoder [44,45]. The vgg16 model has 5 convolutional blocks of increasingly downsampled spatial dimension (each module outputting a 2X spatial downsampling) and expanding channel depth (figs. 3.2.A, 3.2.C, 3.2.D). This can be understood as decreasing "where" information and increasing "what" information as one goes deeper in the network, and the channel depth, or number of learned features, grows. The original vgg16 network comprises several fully-connected layers which are removed in an FCN implementation and replaced with the upsampling path.

The decoder network consists of learned upsampling operations that convert smaller 2D feature maps into larger 2D feature maps via transposed convolutions. Convolution may be seen as a linear operator acting on a flattened input image patch vector $K\,x\,v \rightarrow v'$ where K is a sparse matrix of zeros for pixels outside of the receptive field [46].

To illustrate, consider a kernel of size (4,16) acting on a vector of size (16,1) (i.e. a flattened 4x4 image patch) to return an output vector of size (4,1):

$(4,16)\,x\,(16,1) \rightarrow (4,1)$

Therefore, the transpose of this matrix acting on the subsampled vector v' will map to the dimension of basis of v:

$Dim(K^T) = (16,4)$

K^T x v' → v : (16,4) x (4,1) → (16,1)

The weights of the convolutional transposes are similarly learnable, thus providing a more attractive alternative and performance boost with respect to previous upsampling approaches such as bilinear interpolation [42].

Skip connections are introduced in this paper as a way to connect feature maps from upsampling and downsampling paths. Generally speaking, skip connections combine outputs from two non-neighboring layers, thereby combining representations from different parts in the network. In the case of FCN, skip layers are used to combine learned feature representations of similar sizes. As can be seen in the illustrations of the FCN architecture (figs. 3.2.A, 3.2.C) or equivalently, in the Keras summary of the model used in this work (fig.3.2.D), skip layers combine (by simply adding (pixelwise) the values of the activations) the outputs of each upsampling convolution with a max pooling output in the encoder network of similar dimension. In this way, the concatenation of features from the downsampling path with output activation maps of the upsampling path allows the placement of features to be learned during upsampling. This essentially places downsampled feature representations of local pixel groupings onto the input to an upsampling operation, such that the positional location of these features is learned.

The FCN publication presents the outputs of different stages of upsampling and it can be seen that the granularity of the predictions increases along increasing depth of the upsampling path (fig. 3.2.B) [42].

The final layer of an FCN implementation is an upsampling to the original image size, which in the case of this thesis were images

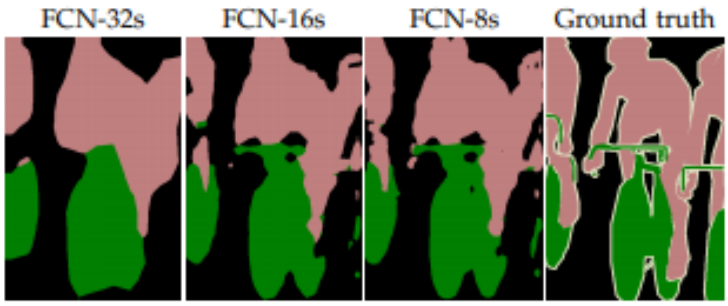

Fig. 3.2.B: FCN-32s: Upsampling to image-sized prediction map directly from the last downsampled layer (downsampled 32X with respect to input image); FCN-16s: Final predictions obtained from the combined maps of a 2X upsampled output of the downsampling path, concatenated with an output of similar size (pool 4); FCN-8s: Output of FCN-16s combined with downsampled filter maps of similar size (pool 3 outputs)

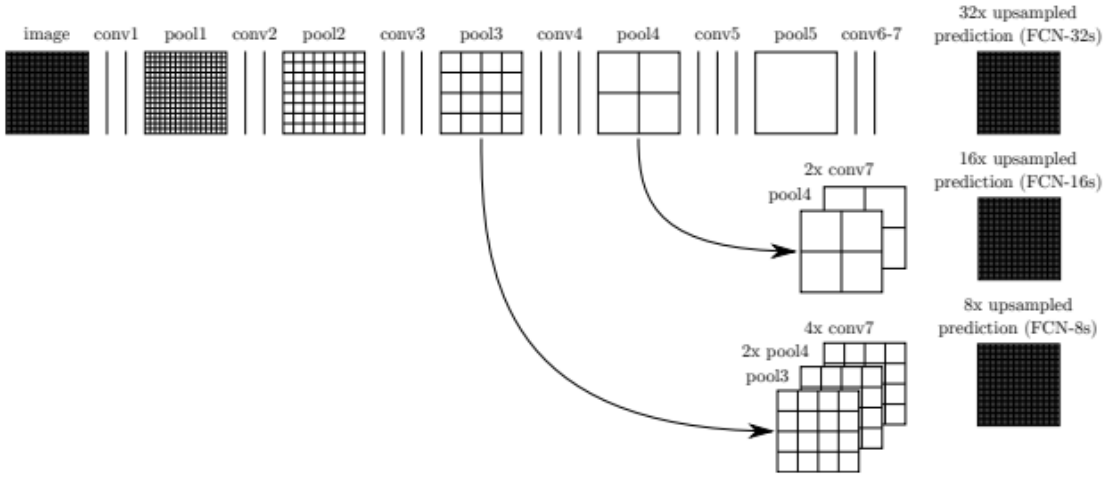

Fig. 3.2.C: FCN architecture highlighting the skip connection upsampling process. Shown to the right are the prediction maps resulting from application of a final upsampling on the corresponding would-be output layer.

The last layer of the encoder network is a custom 1-dimensional convolution appended to vgg16 and used to change the number of filters from the final number of vgg16 filter maps to the number of classes to force a representational bottleneck wherein all of the 512 channels representing the various filters must be fed into $N_{classes}$ 2D channels representative of the classes present in the labels. Therefore, for example, we end up with 3 filter maps of dimension (7,7) at the end of the downsampling path for the network used on the PCDA,DBA dataset and the reported layer output is (?,7,7,3), the first dimension of this tensor referring to batch size as will be explained later.

In the case of the FCN, a softmax vector of classes is returned for each pixel and therefore cross-entropy loss is calculated pixelwise, as each pixel's softmax distribution is compared to the corresponding probability distribution of the pixel's label for that input data, and these pixelwise loss functions sum to a total loss function.

Various accuracy metrics for semantic segmentation exist, each with its own motivation and use cases [49]. A basic metric is a global pixelwise accuracy score reporting the percentage of correctly classified pixels.

$$\frac{\sum correct\ pixels}{N_{total\ pixels}} = \frac{\sum_{ij}(i,j)}{N*M}$$

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792	input_1[0][0]
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Conv2D)	(None, 112, 112, 128	73856	block1_pool[0][0]
block2_conv2 (Conv2D)	(None, 112, 112, 128	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][0]
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][0]
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808	block5_conv2[0][0]
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0	block5_conv3[0][0]
fcn1 (Conv2D)	(None, 7, 7, 3)	1539	block5_pool[0][0]
fcn2 (Conv2DTranspose)	(None, 14, 14, 512)	25088	fcn1[0][0]
add_1 (Add)	(None, 14, 14, 512)	0	fcn2[0][0] block4_pool[0][0]
fcn4 (Conv2DTranspose)	(None, 28, 28, 256)	2097408	add_1[0][0]
add_2 (Add)	(None, 28, 28, 256)	0	fcn4[0][0] block3_pool[0][0]
fcn6 (Conv2DTranspose)	(None, 224, 224, 3)	196611	add_2[0][0]
lambda_2 (Lambda)	(None, 50176, 3)	0	fcn6[0][0]

Fig. 3.2.D: FCN model used for segmentation of PCDA, DBA dataset

3.3 Programmatic aspects

This network was written in Python language using the Keras [5] library. Using Keras' Functional API allowed for modular graph construction, which was necessary for the addition of skip layers and the concatenation of the pretrained encoder with the decoder that I wrote.

Keras, like the underlying lower-level library it's built on, TensorFlow, is based on the premise of specifying a graph structure and then performing computations on inputs once all tensor shapes are specified and found to interact well with all of the other layers. Therefore, as seen in the model summary, the model expects inputs of shape (?,224,224,3) or (None,224,224,3) with the first dimension representing batchsize, which is a specifiable parameter and thus not a priori known to the model. The expected input tensor shape is defined as Keras' built-in vgg16 function.

The decoder network was largely inspired from two implementations found online [47,48]. As stated before, filter hyperparameters such as stride, padding, and channel depth for each layer were copied from these previous implementations that were proven to give the desired output segmentations. Otherwise the arithmetic of the convolutions would have to be worked out, adding unneeded complexity to the problem without any tangible return on time invested. Therefore, in this implementation, the result is that each convolutional layer results in a down/upsampling by a factor of 2. It's still unknown to me what inspires various tweaks to architectures that appear in deep learning literature as these results are mainly just stated and not explained, so it seems likely that these are based on intuition and insight gained from experience rather than anything that can be explicitly readily cast as accessible rules or best practices for non-experts. Most of the programming work in buidling the FCN consisted of translating the found implementations into a working Keras implementation, with various minor programmatic details to resolve.

Chapter 4 Results

4.1 Denoising autoencoder on STM-like image features

As a first approach to machine learning applied to chemical image features, a dataset was designed to simulate the kinds of features present in STM images, and a single-layer autoencoder was trained on these images.

A mapping was designed and trained such that for an unseen input image of an STM-like feature, the network outputs an image with the region of the input image corresponding to the feature recognized by the computer. In order to train the network to learn the desired mapping, a set of binary-valued label images was automatically generated for each corresponding image datapoint, with feature $= 255$, noise $= 0$. This was accomplished by generating "label" images first and then casting a unique, randomly generated Gaussian noise overlay on each image, i.e. elementwise-adding an array of noise to the label array. The positions of the features were similarly randomized within a restricted subgrid of the image (to avoid boundary problems).

A dataset was constructed of (x_i,y_i) pairs of x_i data images and y_i corresponding label images (fig. 4.1.A). This dataset was further divided into train and test sets, where the train set $\{(x_i,y_i)\}_{i \in train}$ was used to construct the desired approximation function f_{network} and the test set $\{(x_i,y_i)\}_{i \in test}$ was used to construct a data subset unseen by the trained model upon which model performance is evaluated. The training data images consisted of randomly-placed features with a noise overlay, simulating acquisition noise as is often encountered in measurement/digital data acquisition instrumentation.

A sigmoid activation function was used in the hidden layer followed by a softmax with cross-entropy loss in one experiment, and in a subsequent experiment, sigmoid applied on the output layer and cross-entropy. Both returned unexpected behavior for cross-entropy as this didn't result in a mapping to the discrete class variables but rather seemingly a mapping to $R \in [0,1]$. This does not make sense and should be reconciled with the rest of the ideas in this work. In both cases, the predicted pixel values outside of the feature region were on the order of 10^{-6} but I expect differences between the two models inside the feature region if one model performs better than the other. Using a sigmoid activation on the output layer and mean squared error loss essentially returned the original images, with random noise outside of the feature region and noised features in correct positions.

Additionally, pixel values were transformed to z-scores, which are centered with respect to mean and scaled to unit variance. Before applying this normalization procedure, the neural network was completely unsuccessful.

This was a fully-connected network with one hidden layer comprising 32 units. The input layer size is that of a flattened image vector, in this case 50*50=2500. The output layer similarly comprises 2500 neurons.

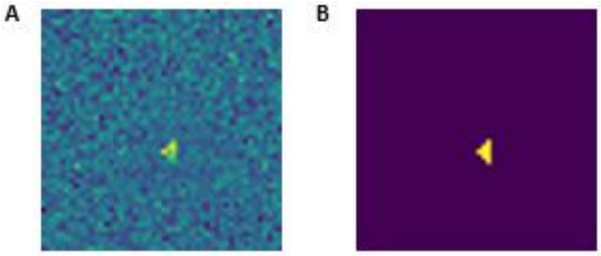

Fig. 4.1.A: a) Noised training image: feature randomly placed with noise overlay; b) Corresponding label image

The demonstrated result was acquired on a dataset of 10^4 greyscale images in a training dataset of dimension (9500,50,50,1), with 9500 training pairs and 500 test pairs. Training on smaller datasets did not work, which seems to indicate that this model is probably performing quite poorly with respect to what would be possible with an improved architecture since so many datapoints were required for what initially appears to be a relatively simple problem. A cursory examination of the results revealed low numerical output values for all pixels, this especially being the case outside of the feature regions enough to markedly distinguish the two. The low prediction accuracies perhaps result from the noise overlay convolution with the feature. Since this result was not the main focus of the thesis, model performance metrics were not collected and further experiments on this model were not carried out. While no statistics were collected on this model, it can be seen that a decent correspondence exists between the true label positions and the predicted label positions.

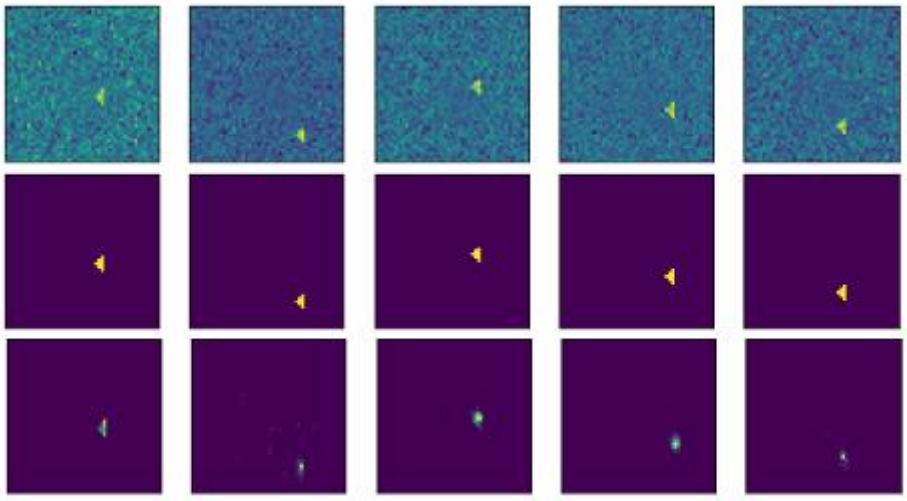

Fig. 4.1.B: Top row) Noised test images; Middle row) Corresponding label images; Bottom row) Predictions returned by the model on the noised test images

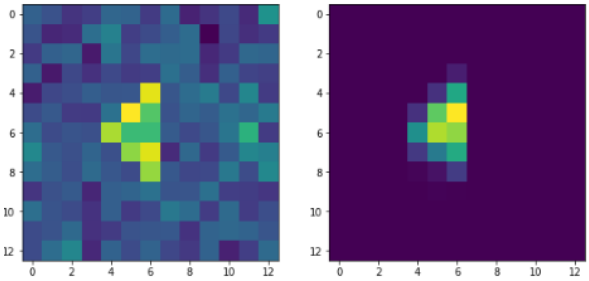

Fig. 4.1.C: Closeups of same region in noised test image and corresponding prediction image

Denoising autoencoder problems are an excellent starting point to develop intuitions for neural network applications to image analysis as both the image and network experimental parameters are readily tunable and relatively easily interpretable. Deep networks are often relatively uninterpretable, in terms of both complexity of the network and the complexity of the input data.

To conclude, this model performed denoising of an input image to distinguish a feature from noise, with the outputted result being in fact a segmented image corresponding to a denoised label image. In this case, this task could likely be readily accomplished by an experienced scientific programmer using classical algorithms. However, this illustrates an important and powerful concept and capability of neural networks: the ability to perform correctly structured tasks by specifying the desired input-output mapping and using an appropriate network.

4.2 FCN Segmentation of STM Images of Self-Assembled Monolayers

Initial result with street scenes data

The FCN was first trained on a dataset of street scene images. These images were resized to (224, 224, 3). A computer with Intel Core i7-7600U processor and 8.00 GB RAM was unable to train batches larger than ~ 5 images and more than 10 or so images in a training session. Therefore due to machine limitations, coarser results were obtained than probably possible on the street scenes dataset (371 images total). The street scenes data present an interesting feature of every 5 images being a timeseries of a specific camera view, with some such subsets involving different objects in different contexts. One subset of 5 images in this dataset is therefore generally not representative of the entire dataset. Structuring of a dataset could perhaps be a useful paradigm for learning interesting relationships between images of a same camera view (shadows/random brightness variations, partial occlusions, etc).

A street scenes dataset was chosen as we desired to establish that the FCN implementation worked. The rationale for this was both the limited size of the chemical datasets as well as the possible lack of overlap of vgg16 features (classes = buildings, cars, etc.) with those present in STM images. However, we suspected that this would be successful as the feature space of the molecules is relatively low-dimensional and ordinary.

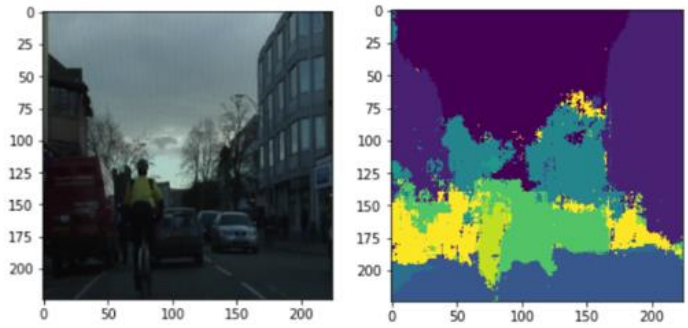

Fig. 4.2.A: FCN segmentation on small dataset of street scene

FCN result on PCDA, DBA dataset

Using a dataset comprising 5 training pairs, the ability of an FCN to classify molecular domains was successfully established. The training images were of varying sizes (fig.4.2.B) as well as brightness distributions both between images and between same feature classes within a

same image. Therefore, based on an extremely small number of training patterns, the model was able to generalize well to novel data.

This model was trained on 30 epochs, using a batch size of 2 images per epoch. Successful performance typically was 30-50 epochs with smaller number of epochs resulting in undertrained, coarser results. Training over this number usually caused the loss function to fluctuate above its neighborhood of minimum values and worse quality images were consequently predicted.

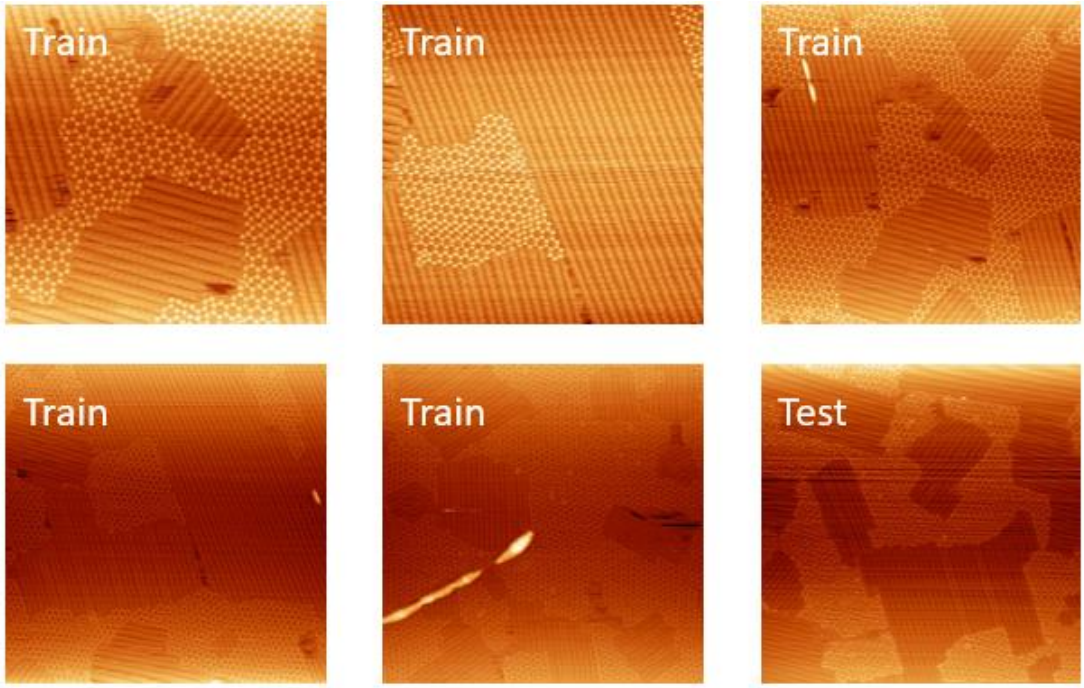

Fig. 4.2.B: The 5 training images used to train the network below and one test image. Note that the images used in the training set are merely transposes of the above dataset with a different colormap used by Python's matplotlib library. Image sizes [nm²]: (Train 1: 100x100), (Train 2: 120x120), (Train 3: 150x150), (Train 4: 200x200), (Train 5: 200x200), (Test, 200x200)

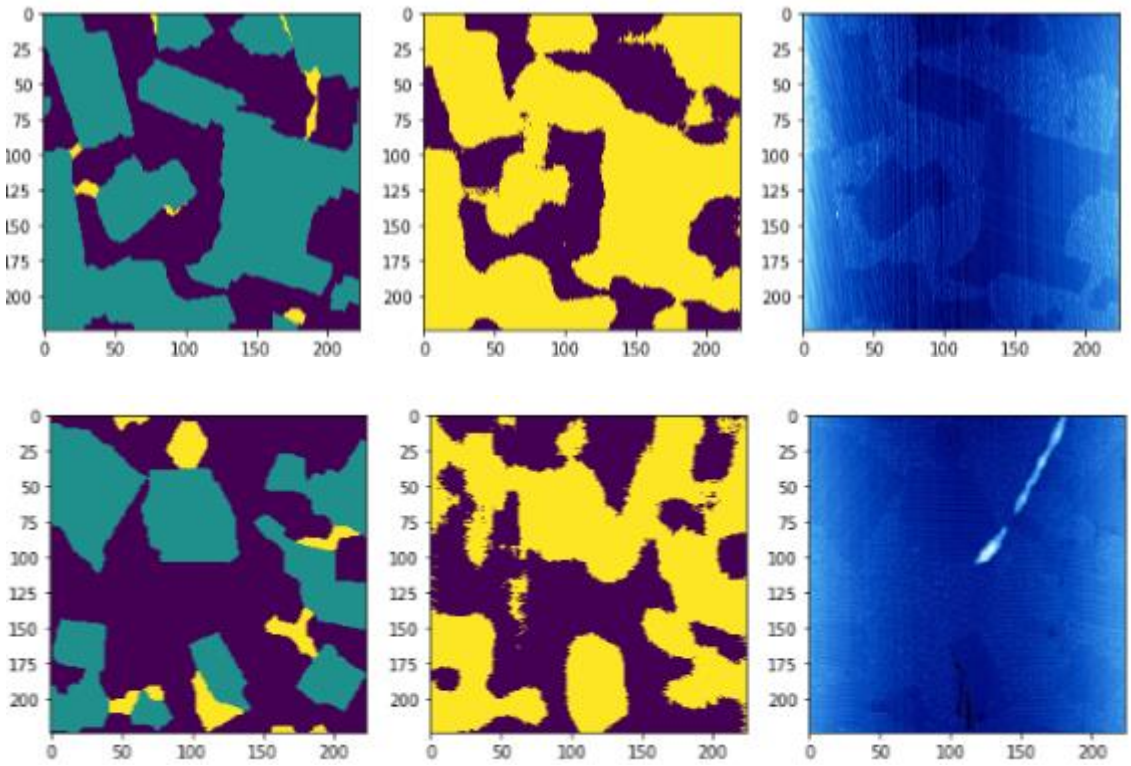

Fig. 4.2.C : From left to right: i) corresponding hand-generated annotations included here for the reader's visual facility in seeing distinct domain patterns ii) model predictions for iii) unseen STM images presented to the trained model

The annotation images contain 3 distinct classes: PCDA (green), DBA (dark blue), Noise (yellow), each having its own distinct mapping to numerical pixel value. Yet interestingly, the model returns only two predicted classes, yellow (PCDA) and blue (DBA). This may be because PCDA and noise are actually somewhat similar, as would be recognized by an STM experimenter familiar with this molecule. In this dataset, PCDA is easily confounded with noise. It will be noticed that Noise and PCDA often occur adjacently (fig. 4.2.C). From having recorded this data myself, I suspect that some of the noise adjacent to PCDA domains is probably in fact PCDA but not registered as such by the STM, perhaps due to dynamical adsorption/desorption at domain interfaces.

A small dataset was used to establish results, as data generation was still relatively time-intensive as hand-labeling is a fundamental bottleneck. This was the case for both PCDA,DBA and CN-PDI datasets. Although ~ 15-20 useable images each would have been extractable in any case, which is not expected to result in huge performance increases. However, the size of this dataset limited the characterization of both model performance and generalizability across images. A core feature of STM image data is the relatively small size of usable data typical of equipment resolution ability. Ambient condition STM often requires a lot of time to collect a dataset of high-resolution images, fundamentally limiting the applicability of deep learning to this field. Probe stability is also highly dependent on the molecular/materials system itself.

FCN classification on CN-PDI images

Having established the applicability of the FCN approach to the previous chemical system of DBA, PCDA coadsorption, images of CN-PDI self-assembly were tested. CN-PDI forms linear domains on HOPG with 6 possible orientations along the 3 symmetry axes of HOPG. In principle we have 7 total classes, 6 being the possible orientations of self-assembly and the 7th being noise/gaps between CN domains. However it was suspected that noise was negligible in these images and that adding a noise class could unnecessarily confuse the model and lower prediction accuracies.

An initial training on 3 CN-PDI images, all of which were of images of different regions, yielded 84.03% pixel classification accuracy (fig. 4.2.D). Transforming the R,G,B values each to a z-score as was successfully done for the autoencoder yielded a decrease in prediction accuracy at 65.71% on the same image.

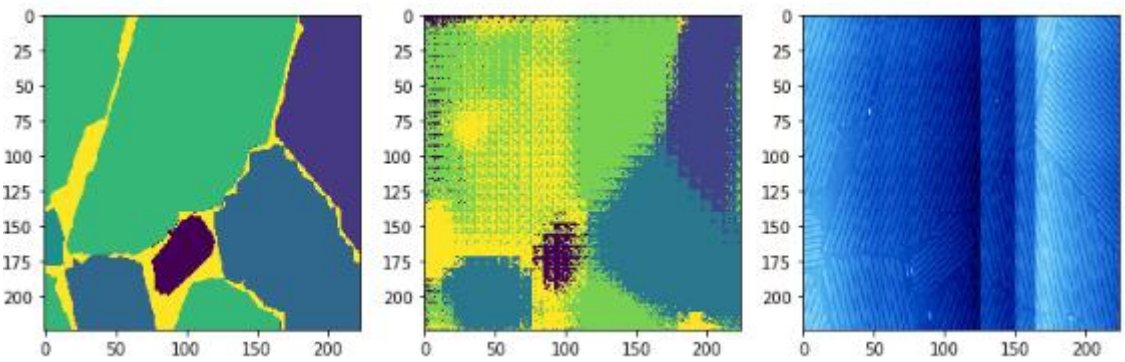

Fig. 4.2.D: FCN classification on CN-PDI using 3 training images

It was demonstrated in the PCDA, DBA segmentation that the FCN model could be trained to be rotationally invariant. PCDA also self-assembles in 6 directions according to the underlying HOPG symmetry. When all of these distinct rotational domain types were presented under a same class label to the FCN model, it successfully classified all rotational instances of PCDA as belonging to the same class. Interestingly, the above result on CN-PDI suggests that the FCN model can be trained to distinguish between different orientations of a same molecular domain when each domain is provided with its own class label, using exactly the same model architecture.

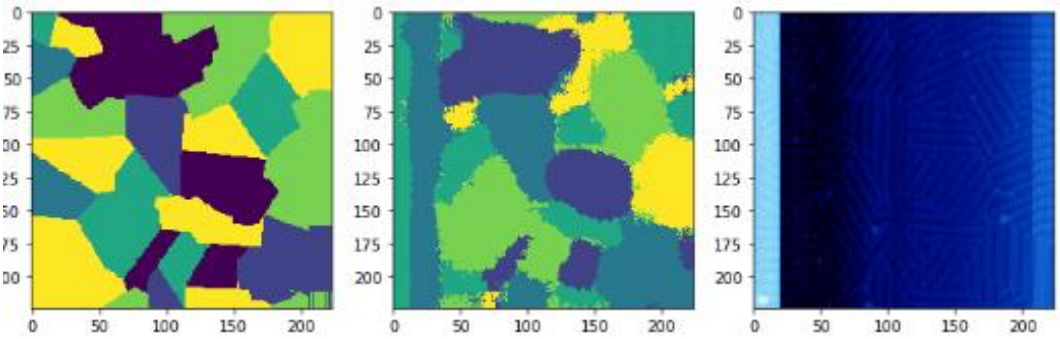

Fig. 4.2.E: Predicting on an image presenting many small domains and contrast artefacts

A new dataset was constructed using eight CN-PDI images from an experiment of 1 mg/mL solvent. The above training session yielded 55.49% pixelwise classification accuracy. As seen in (fig. 4.2.E) the positional placement of the distinct domains is more or less correct but the semantic labeling is most often incorrect with a seemingly correlated confusion between classes. This is interesting as the first dataset of three CN-PDI images successfully distinguished between rotational classes. Seemingly the datasets are similar in size and surface molecular composition, with the second eight-image dataset containing some instances of smaller domain size.

However, it should be noted that while the feature recognition stage was not successful here (and interestingly seemingly for a non-random reason), the placement of features as well as geometry of domain boundaries was learned correctly.

Chapter 5 Conclusion

The goal of this work was to define an image recognition problem on STM images of self-assembling monolayers, and research and implement a machine learning program capable of recognizing features in experimental STM images. Neural networks were decided upon because of their current relevance as a fast-growing field, and their wide-ranging ability of application to pattern recognition tasks.To gain familiarity with neural networks in an interpretable context, a first result of a denoising autoencoder was built. Semantic segmentation was decided upon as the feature recognition task for the STM data, and a fully convolutional network was chosen as the model architecture with the library Keras chosen for implementation.

Herein we demonstrate successful pattern recognition on very small datasets of STM images. We used an FCN-8 model that was able to discriminate between various types of features present in STM images of self-assembled monolayers as well place these features in a reconstructed segmented image. This work demonstrates successful proof of concept of applicability of CNN-based models to perform image recognition tasks for STM images. We constructed and trained an autoencoder for the task of image denoising of STM-like simulated images of noised features. We then constructed and trained an FCN model on two raw experimental STM image datasets of self-assembling monolayers that were constructed using a labeling program written for this purpose. From a preliminary investigation, it was found that for bicomponent PCDA, DBA self-assembly, the FCN model predicted both classes and positions with consistent accuracy. However, the CN-PDI system gave contradictory results requiring further investigation of the origin of this seeming discrepency of being able to at once recognize and not recognize a feature. Globally, these results are encouraging as the FCN model trained quickly on very limited data to still yield effective feature recognition and segmentation.

The possible future directions of this work are probably numerous, as both results are already scientifically interesting problems or ameanable to being developed into interesting work. Almost to even be able to see the interesting possible directions, one must begin by obtaining better data and better model characterizations to know what to do next. By far the most important thing is to characterize FCN model/denoising autoencoder performances on chemical image datasets adequately in order to be able to move further with either one. The main area of immediate improvement would be in assembling a larger dataset, which in this case, using these image datasets, would involve preprocessing in the form of image augmentation and or other pixel value normalization techniques. If this fails to yield significant improvements in prediction accuracy, then it it would be perhaps interesting to use a finer-grained, more performant segmentation algorithm that makes better use of the information contained in the data such as for instance CRFs used in tandem with segmentation networks. However, of course an appropriate model would have to be researched, as there exist other more recent segmentation networks we are not currently familiar with.

Machine learning, pattern recognition, and neural networks are very deep, complex, and fascinating fields – and heavily interrelated - that require a lot of experience, a wide range of knowledge, and domain expertise to be able to truly understand and be able to use effectively.

It is very exciting to live in a time where computational abilities are becoming more and more accessible as they become more powerful, penetrating, and intellectually dazzling.

Appendix

i) Image dimension notation: Images are read by python's numpy library as multidimensional numerical tensors with each entry being an intensity/brightness value. Most scientific python libraries use numpy arrays for numerical data manipulation. I use the numpy convention for recording image dimensions as tensors. A 256x256 greyscale image would be recorded as (256,256) or equivalently (256,256,1) with the third dimension reflecting how many 256 rows x 256 column arrays are stacked depthwise. Image tensors to be used in machine learning have a fourth dimension, that of dataset/batch size. (10,256,256,3) indicates there are 10 RGB images of size 256x256; the last "channel" often referring to number of colors composing an image, or in the case of deep learning, the number of filters a particular convolutional layer contains. Similarly, (1,384,384) is equivalent to one greyscale 2D image of dimensions 384x384 for example. Below are two rows (each row having 256 columns) of a color (256,256,3) image. Each pixel has 3 values for the R,G,B channels:

```
array([[[0.49019608, 0.81568627, 1.        ],
        [0.56862745, 0.83921569, 0.99215686],
        [0.50588235, 0.82352941, 0.99215686],
        ...,
        [0.51372549, 0.78823529, 0.92156863],
        [0.45490196, 0.78431373, 0.96470588],
        [0.40392157, 0.76470588, 0.95686275]],

       [[0.40784314, 0.76078431, 0.96862745],
        [0.49411765, 0.79607843, 0.96862745],
        [0.44705882, 0.79215686, 0.98039216],
        ...,
        [0.50588235, 0.81176471, 0.96470588],
        [0.45490196, 0.79215686, 0.98431373],
        [0.39607843, 0.74901961, 0.95294118]],
```

ii) All the code written for this project can be seen at https://github.com/schuylerhanson

References and works used

[1] Iqbal, P., Preece, J., Mendes, P. (2012). *Nanotechnology: The "Top-Down" and "Bottom-Up" Approaches.*

[2] Clarke, P., Coverney P., Heavens, A., et al. (2016) *Big Data in the physical sciences: challenges and opportunities, Alan Turing Institute*

[3] Rodrigues Jr., J., Florea, L., de Oliveira, M., et al. (2019) *A survey on big data and machine learning for chemistry* https://arxiv.org/ftp/arxiv/papers/1904/1904.10370.pdf

[4] Spears, B., et al., (2018). *"Deep learning: A guide for practitioners in the physical sciences", Physics of Plasmas, Volume 25, 8*

[5] Chollet, F., *Keras* https://keras.io/

[6] Mater, A., Coote, M. (2019) *Deep Learning in Chemistry, Journal of Chemical Information and Modeling, Volume 59, 6, 2545-2559*

[7] LeCun, Y., Bengio, Y., Hinton, G. (2015). *Deep learning. Nature, 521(7553), 436–444*

[8] Deng, L. (2014) *Deep Learning: Methods and Applications, Foundations and Trends in Signal Processing, Vol. 7, 3-4, 197-387*

[9] Ronneberger, O., Fischer, P., Brox, T. (2015) *U-Net: Convolutional Networks for Biomedical Image Segmentation. MICCAI*

[10] Krizhevsky, A., Sutskever, I. Hinton, G., (2012) *ImageNet classification with deep convolutional neural networks. In Proc. Advances in Neural Information Processing Systems 25, 1090–1098*

[11] Liu, X., Zhidong, D., Yuhan Y. (2018) *Recent progress in semantic image segmentation, Artificial Intelligence Review, 6*

[12] Nassif, A., Shahin, I., Imtinan, A., et al. (2019) *Speech Recognition Using Deep Neural Networks: A Systematic Review, IEEE Access*

[13] Hornik, K., Stinchcombe, M., White, H. (1988) *Multilayer Feedforward Networks are Universal Approximators, Neural Networks, Volume 2, 359-366*

[14] Lu, Z., Pu, H., Wang, F. et al. (2017) *The Expressive Power of Neural Networks: A View from the Width. Neural Information Processing Systems, 6231-6239*

[15] Goodfellow, I., Bengio, Y., Courville, A. *(2016), Deep Learning. MIT Press*

[16] Xu, Y., Xiao, T., Jiaxing, Z., et al. (2014) *Scale-Invariant Convolutional Neural Networks.* https://arxiv.org/pdf/1411.6369.pdf

[17] O'Mahony, N., Campbell S., Carvalho A. et al., (2019*). Deep Learning vs. Traditional Computer Vision.* In K.Arai, S.Kapoor (Eds.): *CVC 2019, AISC 943, 128-144*

[18] Yuheng, S., Hao, Y. (2017). *Image Segmentation Algorithms Overview. arXiv:1707.02051v1.*

[19] Khan, A., Anabia, S., Zahoora, U. (2019) A Survey of the Recent Architectures of Deep Convolutional Networks, https://arxiv.org/ftp/arxiv/papers/1901/1901.06032.pdf

[20] Chen, J. (1994) *Introduction to Scanning Tunneling Microscopy. 2nd Edition*

[21] Weigelt, S., (2007) *Self-assembly, dynamics, and reactions of organic molecules on metal surfaces: A scanning tunneling microscopy study. PhD Thesis*

[22] Tersoff, J., Hamann, D. (1984) *Theory of the scanning tunneling microscope, Physical Review B. Volume 31, 2, 805-813*

[23] Elemans, J., Lei, S., De Feyter, S. (2009). *Molecular and Supramolecular Networks on Surfaces: From Two-Dimensional Crystal Engineering to Reactivity. Angewandte Chemie International Edition, 48(40), 7298–7332.*

[24] Ulman, A. (1996). *Chem. ReV., 96, 1533-1554*

[25] Verstraete L., Greenwood J., Hirsch, B., De Feyter, S. (2016) *Self-Assembly under Confinement: Nanocorrals for Understanding Fundamentals of 2D Crystallization. ACS Nano, 2016, 10, 12, 10706-10715*

[26] Kaparthy, A., *Standford University cs231 notes* http://cs231n.github.io/neural-networks-1/

[27] Keras documentation https://keras.io/getting-started/faq/#what-does-sample-batch-epoch-mean

[28] Glorot, X., Bordes, A., Bengio, Y. (2011) *Deep Sparse Rectifier Neural Networks. Proceedings of 14th International Conference on Artificial Intelligence and Statistics, PMLR 15, 315-323*

[29] Kingma, D., Ba, J. (2014) *Adam: A Method for Stochastic Optimization. International Conference on Learning Representations*

[30] Szeliski, R. (2010) *Computer Vision: Algorithms and Applications. Springer.*

[31] Rawat, W., Wang, Z. (2017). *Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. Neural Computation, Volume 29 (9), 2352-2449*

[32] Chen, S-C., Zhao, T., Gordon, G., et al. (2006). *A Novel Graphical Model Approach to Segmenting Cell Images. 2006 Symposium on Computational Intelligence and Bioinformatics and Computational Biology.*

[33] Zhou, H., Zhang, J., Lei, J., Li, S., Tu., D. (2016). *Image semantic segmentation based on FCN-CRF model. 2016 International Conference on Image, Vision and Computing.*

[34] Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., et al. (2017) *A Review on Deep Learning Techniques Applied to Semantic Segmentation* https://arxiv.org/pdf/1704.06857.pdf

[35] Ziatdinov, M., Maksov, A., Kalinin, S. (2017) *Learning surface molecular structures via machine vision. npj Computational Materials, 3, 31*

[36] Fourcade, A., Khonsari, R. (2019). *Deep learning in medical image analysis: A third eye for doctors. Journal of Stomatology, Oral and Maxillofacial Surgery, Volume 120, Issue 4, 279-288*

[37] Huang, B., Zhenghao, L., Jiangyu, L. (2018) *An artificial intelligence atomic force microscope enabled by machine learning. Nanoscale, I45*

[38] Yang, S., Berndl, M., Ando D.M., et al. (2018) *Assessing microscope image focus quality with deep learning. BMC Bioinformatics 19:77*

[39] Ziatdinov, M, Dyck, O., Maksov, A., et al. (2017) *Deep Learning of Atomically Resolved Scanning Transmission Electron Microscopy Images: Chemical Identification and Tracking Local Transformations. ACS Nano 11, 12, 12742-12752*

[40] Dong, D. Liao, G., Hongwei, L., et al. (2018). *A Review of the Autoencoder and Its Variants: A Comparative Perspective from Target Recognition in Synthetic-Aperature Radar Images. IEEE Geoscience and Remote Sensing, Volume 6, Issue 3*

[41] http://www.robots.ox.ac.uk/~vgg/software/via/via_demo.html

[42] Long, J., Shelhamer, E., Darrel T. (2015) *Fully Convolutional Networks for Semantic Segmentation. CVPR* https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf

[43] Kaparthy, A. *Stanford University cs231 notes* http://cs231n.github.io/transfer-learning/

[44] Simonyan, K., Zisserman, A. (2014) *"Very Deep Convolutional Networks for Large-Scale Image Recognition", CVPR* https://arxiv.org/abs/1409.1556

[45] *Keras vgg16 Implementation* https://keras.io/applications/#vgg16

[46] Dumoulin, V., Francesco, V. (2018) *A guide to convolution arithmetic for deep learning* https://arxiv.org/pdf/1603.07285.pdf

[47] https://nanonets.com/blog/how-to-do-semantic-segmentation-using-deep-learning/

[48] https://fairyonice.github.io/Learn-about-Fully-Convolutional-Networks-for-semantic-segmentation.html

[49] Csurka, G., Larlus, D., Perronnin, F. (2013) *What is a good evaluation measure for semantic segmentation?. BMVC*

[50] https://en.wikipedia.org/wiki/Cross_entropy

---

[51] De Feyter, S., & De Schryver, F. (2005). *Self-Assembly at the Liquid/Solid Interface: STM Reveals. The Journal of Physical Chemistry B, 109(10), 4290–4302*

[52] Ciricosta, O., Scott, H., Durey, P., et al. (2017). *Simultaneous diagnosis of radial profiles and mix in nif ignition-scale implosions via x-ray spectroscopy. Physics of Plasmas, 24(11):112703*

Images used in figures

- Fig. 2.1.1: http://hoffman.physics.harvard.edu/research/STMtechnical.php
- Fig. 2.1.2: [X] De Feyter, S., & De Schryver, F. (2005). *Self-Assembly at the Liquid/Solid Interface: STM Reveals. The Journal of Physical Chemistry B, 109(10), 4290–4302*
- Fig. 2.1.3.A: Own figure
- Fig. 2.1.3.B: Own figure
- Fig. 2.2.1.A: https://www.astroml.org/book_figures/chapter9/fig_neural_network.html
- Fig. 2.2.1.B: [7] LeCun, Y., Bengio, Y., Hinton, G. (2015). *Deep learning. Nature, 521(7553), 436–444*
- Fig. 2.2.1.C: Own figure
- Fig. 2.2.1.D: https://rpubs.com/leexiner/bigdata-final-project
- Fig. 2.2.2.A: https://forum.smartcitizen.me/t/about-filters-and-convolution-signal-processing-for-audio-part-iv/945
- Fig. 2.2.2.B: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- Fig. 2.2.2.C: [15] Goodfellow, I., Bengio, Y., Hinton, G. (2016). *Deep Learning. MIT Press.*
- Fig. 2.2.2.D.A: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling
- Fig. 2.2.2.D.B: [15] Goodfellow, I., Bengio, Y., Hinton, G. (2016). *Deep Learning. MIT Press.*
- Fig. 2.2.3: [33] Zhou, H., Zhang, J., Lei, J., Li, S., Tu, D. (2016). *Image semantic segmentation based on FCN-CRF model. 2016 International Conference on Image, Vision, and Computing.*
- Fig. 2.2.4: [4] Spears, B., et al., (2018). *"Deep learning: A guide for practitioners in the physical sciences", Physics of Plasmas, Volume 25, 8*
- Fig. 3.1: Own figure
- Fig. 3.2.A: Sankaranarayanan, P., Saber, E., Schwartzkopf, W. (2018) *Supervised Classification of Multisensor Remotely Sensed Images Using a Deep Learning Framework. Remote Sensing, 10, 9*
- Fig. 3.2.B: [42] Long, J., Shelhamer, E., Darrel, T. (2015). *Fully Convolutional Networks for Semantic Segmentation. CVPR*
- Fig. 3.2.C: [42] Long, J., Shelhamer, E., Darrel, T. (2015). *Fully Convolutional Networks for Semantic Segmentation. CVPR*
- Fig. 3.2.D: Own figure
- Fig. 4.1.A: Own figure
- Fig. 4.1.B: Own figure
- Fig. 4.1.C: Own figure
- Fig. 4.2.A: Own figure
- Fig. 4.2.B: Own figure
- Fig. 4.2.C: Own figure
- Fig. 4.2.D: Own figure

- Fig. 4.2.E: Own figure

DIVISION OF QUANTUM CHEMISTRY AND PHYSICAL CHEMISTRY
Celestijnenlaan 200 F b2404 Chem &Tech
3001 LEUVEN, BELGIË
tel. +32 16 37 21 98
jermy.harvey@kuleuven.be
https://chem.kuleuven.be/en/research/qcpc/tcc