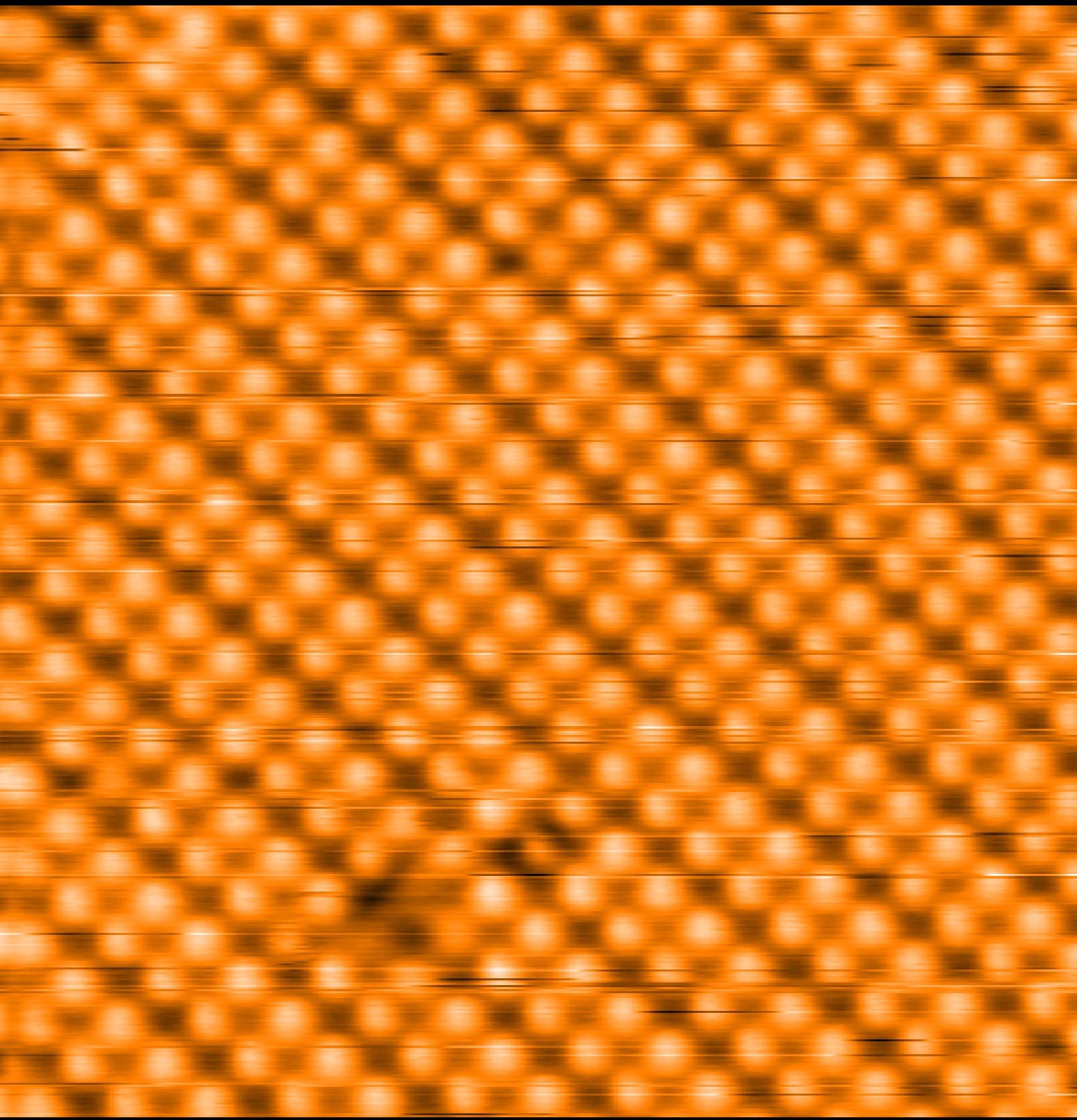
Using Neural Networks to Classify and Automate Tip Production in SPM Imaging

Oliver Gordon & Nicole Landon
School of Physics and Astronomy
University of Nottingham



Contents

1 Introduction

The study of Scanning Probe Microscopy (SPM) incorporates imaging techniques such as STM and AFM, amongst others. When an atomically sharp tip is brought to within a few nanometres of a sample, a distance dependent signal will be induced. For STM, this signal will be a quantum tunnelling current. For AFM, this signal will be related to inter-atomic forces. By rastering the tip backwards and forwards across a sample, an image will be produced over time. An analysis can then be drawn by investigating both qualitative and quantitative aspects of multiple images.

Despite the existence of thorough, detailed research into the analysis of SPM images, the manufacture of atomically sharp SPM tips is currently rudimentary and inelegant. As well as altering the tip apex by etching, annealing and crashing it into a surface, an SPM operator must also change various environment dependent scan parameters such as voltage, current and scan speed to optimize the system to demonstrate features of interest. The tip must also be given a sufficient 'settle time' to reach its lowest energy configuration before use. This trial and error approach to manufacturing tips often leads to probes that are often blunt and/or contain multiple apices. These tip defects result in images that can be blurred, stretched and/or self repeated, amongst other flaws.[1,2] Owing to the fragility of the tip, these flaws can also spontaneously appear whilst scanning a sample. Correcting probes is a tedious and time consuming process that would benefit greatly from automation.

The potential for a neural network to be used in the classification of images has long been recognised throughout a number of fields. Popular examples include medical imaging,[3] satellite imaging,[4] cell identification[5,6] and facial recognition.[7] There also exist many thorough investigations regarding the use of machine learning techniques to increase the accuracy of breast cancer diagnoses.[8–10] Given that it should only require a semi-skilled operator to determine apex flaws, there exists the potential to exploit machine learning processes in order to determine apex flaws.[11]

After discussing the issues with tip production and prior literature into the use of automated and semi-automated methods to correct images, we will discuss the current state of our work into creating a neural network to correct tip defects.

2 Literature Review

Although it is possible to manually deduce an apex flaw through observation of well known samples, these cannot be determined when analysing unknown samples. It is also computationally expensive to realistically model the chemical interactions between the tip apex and the surface. There is therefore a significant impetus to not only better the production of tips, but also to computationally analyse SPM images, rather than tips, to determine what apex flaws are present.

In limited situations it may, however, be possible to remove artifacts from SPM images and salvage images produced by tips with multiple apices. Because this avoids the need to reform the tip and re-image the sample, this would obviously save a great deal of time. It could also be of use when imaging rare samples or with expensive tips. In 2016 Yun-Feng Wang et al. were able to do this without knowledge of the precise tip shape, and also without loss of resolution. This was achieved by first using Fourier transforms and ensemble learning to roughly guess a blur function; such methods have been well researched outside of SPM. By de-convolving the observed image with the rough blur function, a rough corrected image was generated. From here, novel steps were applied to hone in on an improved blur function. However, this method was not able to reconstruct all 'double tip' images. The tip may therefore still need to be reformed for future use.

As discussed previously, one issue with flawed tip manufacturing is that tips with multiple apices present will produce incorrect, repeated signals that are then convolved with the correct signal, creating ghosting artifacts in SPM images. In 2014 Straton et al were able to unambiguously detect an SPM image produced by such a 'double tip' by exploiting the translational symmetry of the artifact produced.[2] Crystallographic Image Processing (CIP) techniques were also used, which employ knowledge of lattice periodicity and advanced fourier analysis. The semi-automated analysis program, CRISP, was used to detect the periodicity of such images and therefore provide a point of comparison. Both methods often yielded the same periodicities.

This area of interrogation is yet to be fully realised, however. This research is not based on real STM images, but instead on simulations. These simulations also made the oversimplification that there are no interactions between the multiple apexes of the flawed tip. Some suggested double tip symmetries are also questionable. Also, because CIP operates by determining the phase difference of the currents in reciprocal space at each apex, CIP can fail if this phase difference coincidentally happens to equal 0 for a particular image. Furthermore, although the detection of periodicities is thought to hold for multiple types of SPM imaging beyond STM, these are yet to be simulated.

Another major issue involving blunt tips regards the production of blurry images. Blurring is often a spatially invariant process, produced by a blur kernel which describes the convolution of multiple types of basic blurs, such as Gaussian and directional blurs.[12] Removing such blurs is therefore a time consuming process of iterative trial and error. This process can be routinely programmed with MATLAB and Support Vector Machines (SVMs). However, these often require upwards of one million support vectors to achieve satisfactory results of over 95% blur detection.[13]

As both of the above methods are achieved with SVMs or generic algorithms, they will be computationally slower and less accurate than ones involving deep learning. A Multivalued Neural Network (MVN), for example, can be used to remove the same blurs as before. This requires only about 1000 complex weights, yet can detect simple blurs with an accuracy of approximately 98%.

However, this accuracy is only observed when reconstructing an image with one specific type of blur, and not with the combination of blurs inevitably seen in a typical STM image.[1] Although this could potentially be resolved by increasing the number of support vectors or complex weights, this increases the complexity of the system, slowing it down. This technology would therefore have to be adapted specifically for use with SPM imaging techniques and with no guarantee of results.

However, there also exists the issue of determining if the artefacts thought to be present are actually just features of unknown materials being scanned. In order to verify the state of the tip, a well understood surface such as Si(100) could be used as a reference to optimise a tip before using it to image materials with structure that is yet to be confirmed/determined. This process has the potential to be automated through machine learning techniques.

One such approach could involve recurrent neural networks. For example, a neural network invariant to both rotation, translation and scale could be trained to recognise image features which result from flaws, such as a tip crash, and automatically react and retest a tip. For this to occur, many images of the reference surface should be categorised in order to train a neural network to reach the desired image quality. This is our intention.

Besides the time savings, an automated response to tip formation and testing may produce better tips. This is as human brains have been shown to often observe structure in random distributions. As a result, there will be disagreement and bias amongst individuals when classifying images.

3 Neural Networks

When creating a neural network, both MATLAB and the keras[14] module in python provide simple frameworks for you to work with. Because this is such a new area of research, you will find that the best guides and tutorials will change on a yearly basis. MATLAB was popular a few years ago, but keras is now king. We will therefore only discuss the specifics of keras. However, we have been in constant contact with a computer scientist at the University of Newcastle, Ding, and he believes he can make something good in MATLAB. This is absolutely worth following through.

When training with keras, it is important to keep the backend consistent between your training and classification scripts. You will want to use either tensorflow or theano. Tensorflow is newer, so is preferable. Also bear in mind that the keras and theano documentation is outdated in places. These backends cannot be imported like regular python modules, and may cause you substantial grief. On Windows, everything can be easily installed at once with WinPython.[15] On Linux, it's helpful to make a virtual environment and install everything with anaconda. Tutorials for installing keras on linux are often included with general keras guides.[14,16–18] Linux executes python code far faster, so is preferable. If you can't access a computer with Linux, some Linux distributions can be natively installed on the (as of now in preview) Windows 10 Autumn Creators Update. In either case, make sure GPU support is enabled as training on the CPU is orders of magnitude slower than on GPU.

A neural network operates by trying to weight equations that map an input (in our case, image data) to several outputs (in our case, a combination of 6 categories).[19] It does this by creating interconnected neurons between transformations (also known as layers) of different depths and sizes. Every input is connected to every output by a network of nodes which are connected by neurons. Every node is therefore also connected to every other node. There is a training set of data that trains the model, and a testing set of different data that tests how correctly the model has learnt from the training set. The network guesses weights and parameters for the nodes, sees how correct its answers are compared to the training set, and then attempts to make changes that improve its ability to make correct predictions.[20] The network's ability to do this is then finally verified with the testing data.

One of three important metrics when training a neural network is the number of epochs the network is taught for. This is the number of times the entire dataset has been processed. Increasing the number of epochs increases the training time. Training will be orders of magnitude faster when using a computer with a fast GPU.

A simple dataset, such as mnist, may achieve good results within 30-50 epochs. More complex image data, such as cifar10,[21] could require 100+ epochs to fulfill a networks' potential. For complex images like ours, it is not unusual to see no progress within 20-30 epochs, (although in this case you may want to look into different initializations).

The other two metrics to be concerned with are accuracy and loss. Accuracy is how often a correct prediction is made. Loss is a summation of the errors made during training. Your aim is to maximise accuracy. The machine's aim is to minimise loss. One consequence of this conflict is that you cannot set a large number of epochs and achieve the highest possible accuracy and lowest possible loss for an individual network at the final epoch. If you do this, you will almost certainly discover that your network has overfit.

Overfitting, simply put, is when a model learns about random noise and detail in the training dataset that is not present in the testing dataset. This can therefore be seen as the testing accuracy dropping well below the training accuracy. It will also be seen as the accuracy decreasing, even though the loss is also decreasing itself. You may be able to solve it in several ways. You could reduce the complexity of your network, add more repeated images,[22] add noise layers,[23] add dropout,[24] reduce momentum or learning rates. Alongside other options,,[25] you will also want to implement early stopping to detect when overfitting begins.[26]

Underfitting, on the other hand, is when the overall accuracy is low, loss is high, and the network doesn't seem to achieve anything. This is obvious to determine, and you may spend a lot of time in this state. You may find that leaving to train for a while may help, however.

You want to achieve a balance between overfitting and underfitting. For this reason it is a good idea to set up a large number of epochs and record the progress of the network to determine how it improves or overfits.[27] If you discover it has not overfit, then it almost certainly will have underfit. If you immediately overfit or cannot eventually overfit a simple, basic model, then something is wrong.

When learning about the setup of keras, it may be helpful to practise on a simple network such as the well established mnist data set. This aims to detect the numbers 1-9.[14] Most tutorials will cover the training of the mnist data set, as you can achieve high accuracy in a relatively short period of time with a simple network.

There are also different loss and activations functions that will vary depending on the setup of the network. When training in keras, we will primarily be using "categorical_crossentropy" as the loss function if one category is to be selected, or "binary_crossentropy" for multiple. You may get good results from either, though.

We also chose "sigmoid" as our activation function as opposed to "softmax," which makes all classification scores add up to 1 and is therefore better for mutually exclusive classifications. We also do not use "relu" as intermediate activation functions as this is for "softmax" and will cause some categories to not be classified.

There are a multitude of optimisation functions you can use, and these will affect both the rate at which you can train at and the maximum final accuracy of the network.[28] We have attempted with both "adam" and "sgd," both with and without Nesterov momentum enabled.

Before even beginning to make a network, however, I feel it useful to stress that a neural network does not possess any level of common sense. It simply tries to build a series of mathematical equations and transformations that map one set of numbers to another. It will probably see a straight line on the right edge of a box as completely different to one on the left side. It will probably see a straight line rotated 10 degrees as completely different to one that is unrotated. It will probably see a slightly warped straight line as completely different to an unwarped one. These particular problems can be overcome by repeating images with various transformations,[29] and should be done in all cases.

To complicate matters further, just because you see a high accuracy does not automatically mean the network is good. The network may try to "cheat" by learning the order of inputs and/or the probability of an image being a category. You should be mindful of this and always create a full classification matrix to test.

It should also be kept in mind that creating a good network is not a perfect process. A premade network that gives 70% accuracy for detecting dogs and cats may only be 15% accurate for our purposes. You will have to optimise various network architectures, epoch numbers, normalisation methods, types of layers, numbers of layers, orders of layers, depths of layers, activation functions, loss functions, optimisation functions, training rates, decay rates, momentums, resolutions, backends and dataset formats (to name but a few) in order to create the best network possible. You may even find that you will have to create and optimise one network for each category and then combine the scores at the end and still may not be very successful. There is even research in using neural networks to setup neural networks.

4 Acquiring & Manipulating Data

4.1 Acquiring Images

To start, we first acquired a set of 6167 images of the H passivated Si(100) plane at variable temperature. These images were taken between 2014-2017. The images have a range of tip qualities, and in some cases contain defects and step edges. Most are 256x256 or 512x512, but some are unusual sizes from when a scan was not run to completion.

These images are all saved in the proprietary .z_mtrx file format, along with a .mtrx index file. However, the SPIW toolbox[30] was developed to allow these to be read into MATLAB.

After installing V0.3.1 (the latest available at the time of writing) of the toolbox as instructed in the readme file, we found that we had to change one programming line. In the file SPIW_open.m, we changed XtraceSpecOpen at lines 69 and 76 to XtraceImOpen, and were able to continue without further difficulties.

One caveat of SPM imaging is that certain features, such as impurities, must be "masked" out, and that contrast often needs to be readjusted to obtain a high quality image. Images are often also "flattened." If this is not performed, then the surface of the sample often appears as a gradient. This sometimes makes features harder to distinguish. There are multiple algorithms for doing this, most of which are available with the SPIW toolbox.

We opted to use the Im_Flatten_X function from the SPIW toolbox, as opposed to the better looking Im_Flatten_XY in our script. This was as the X flattening is done in real time by the capture software, but not XY. Using XY flattened images to train the network would therefore cause issues as what it learnt about what mapped dimers to dimers, etc, would no longer apply.

We did not apply any other processing to the images. Although it would be easier to classify images that have manually edited masks, levels of flattening and contrast, this cannot be automated in real time, and so could not be used to train a network with. In future, it could therefore be a good idea to investigate what would happen if people were shown postprocessed images, and the machine trained with the unedited versions. It would also be interesting to see if object detection and other networks could be used to automatically mask out defects and alter contrast accordingly.

After flattening, we then converted the images to the more conventional .png format, and saved them in a single folder. The file that does all of this is called mtrx2png.m.

4.2 Zooniverse & Image Categories

To train a neural network, it must first be given a series of correct outputs to learn from. Because many images are required and because previously people have been seen to disagree when classifying, we planned to ask multiple people to classify images, and take a mean result to train with.

Originally, our plan was to distribute a MATLAB GUI, alongside a folder containing all of the images, to various students and staff. This would then allow them to randomly view and classify the images into different folders for each category the neural net was to classify into, before combining all the folders together. The file Classification_GUI.m was written to do this. However, this would have been difficult to scale up to many people, and would have been difficult to analyse. This program also cannot allow images to be placed into multiple categories, or for

individual classifications to be easily deleted.

An online system, called Zooniverse, allows for images to be randomly seen and classified without asking individuals to send and recieve files and data. It also allows for tutorials, continuously evolving data sets, and to keep a record of each classification made. In future, it would be a good idea to contact Zooniverse to have this project listed officially; such projects often have thousands of scientifically inclined volunteers classifying information, and would therefore provide very good mean results.

After changing to Zooniverse, our basic page[31] was set up. We chose the following six categories to classify images into:

- Asymmetries
- Individual Atoms
- Dimers
- Rows
- Tip Change
- Bad/Blurry

Our definitions for each category are roughly outlined by Møller et al.,[32] and alongside edge cases are explained in the included tutorial. Step edges and surface defects were ignored. We chose to not use a pyramidal hierarchy wherein seeing one category would mean selecting everything below, as there are edge cases between categories that would not be uniquely defined otherwise.

We also allowed multiple categories to be selected for an image, as sometimes tip changes would be present in an otherwise pristine image, and at other times there would be insufficient resolution to observe one feature, but more resolution than the category below.

For the bad/blurry category, we asked this be chosen based on the time taken to make a decision. This is as features can often be manually made out with a change of contrast, but a network could not do this. As such, if a feature could be readily distinguished, then the feature was selected and bad/blurry was not. If a feature could be distinguished with some observation, then both the feature and bad/blurry was selected. If a feature could still not be seen, only bad/blurry was selected.

In future, it would be helpful to allow for sliders for each feature, instead of a binary checkbox (although this would mean that the equation for agreement score, as discussed below, would need to be changed to a standard deviation of the range of answers). It would also be good to have different subcategories in the bad/blurry category, such as noise, tip crash, etc.

We also have no way to label images in which a tip change has caused a change of viewable resolution on either side of the change. This may also be the case for a step edge.

Zooniverse allows classification data to exported twice a day, and emails a .csv file containing JSON strings of all classification events and answers.

4.3 Manipulation & Analysis of Zooniverse Data

We chose to take a mean score of the classification results for each image. This was because if we trained the system with every classification event and the system was scaled up to hundreds of individuals classifying even more images and image repeats were taken into account, then there would be several million images produced. The resulting image sets would then take months to train. Because we also intended to allow SPM operators to train new images during normal research, the resultant learning would be minimal.

We therefore created the MATLAB code Big_Data_Helper.m. This deconstructs the .csv file into a structure array, before analysing the JSON data contained within and outputting to a simpler excel file.

There are three worksheets in this excel file, and on each row there is:

- Mean Score
  - · Image name
  - · Number of times the image was classified
  - · A mean classification score, $\bar{x}$, from 0-1 for each category. 0 is always unselected, 1 is always selected.
- Weighted Score
  - · Image name
  - · Number of times the image was classified
  - · A mean score for each category as a relative percentage of the other categories
  - · As explained below, we deprecated this
- Agreement Score
  - · Image name
  - · Number of times the image was classified
  - · Agreement value from 0-1 for each category, $2|\bar{x} - 0.5|$
  - · Average agreement value from 0-1 for each image

4.4 Generating Training Images

A neural network, such as one created in keras or MATLAB, takes two inputs when training. One contains multiple images, while the other contains the correct classification of those images.

The images are not supplied as native file formats such as .png, but instead as 3D matrices, in which the x and y axis contain pixel brightnesses from 0-255, and each layer of z contains a new image. For RGB images, it is required to provide three matrices for each colour channel. However, we use greyscale images on one single channel as the colour of the images is not important. It is also beneficial to the speed and accuracy of learning algorithms to be given these inputs from 0-1. The pixel brightnesses are therefore divided by 255.

For our usage, there is a complication when providing the correct classifications for the images. In certain situations, it is only possible to select one category with the value 0 or 1, while in others it is possible to provide multiple categories but still only as 0 or 1. This is as networks are

typically used to make binary choices (e.g. there IS a DOG and there IS a TREE and there IS NO HORSE). As we were only able to reach 85% agreement during our classifications, it would be bias and misleading to proclaim that an image, for example, is/is not atomic, when people have given conflicting answers.

When it is possible to use the mean scores to train with, datasets can be produced with the trainer_preprocess_mean.m script and it is recommended to use this file. We only had time to do small scale training with this, but were able to achieve a promising level of training in one category after some weighting. More training time may address this, although it may still be better to train individual categories.

To make use of our mean answers, we therefore weight the data by repeating images based on the mean score for an individual category. For example, if the preprocess file is run with a maximum weight of 4 and an image has mean scores of bad/blurry=0.4 and tip change=0.8, the training dataset will have 2 repeats of the same image labelled as bad/blurry=1 and everything else=0, and 4 repeats of the same image labelled as tip change=1 and everything else=0. These datasets are made by the trainer_preprocess_unbalanced.m script. Through averaging, the network should hopefully be able to therefore take means into account.

However, datasets produced by this script may not train in a useful manner and with a deceptively high accuracy. As about 80% of the images are in the bad/blurry category, a network may "cheat" and simply guess 80% bad/blurry for every image and vice versa for all other categories. This will give a low loss and high accuracy, but means that the network is not actually making any predictions. A network may still be able to overcome this by training for a long time.

Alternatively, by knowing the fraction of images in each category, it is possible to produce even more repeats of images and in doing so correct the imbalance in the dataset by "overbalancing" it. Such datasets can be produced with the trainer_preprocess.m script.

In either case, we used the methods we did in order to take out our own biases and avoid entering our own; instead letting the network organically resolve issues. The only removal of data was for images with a low overall agreement score of less than 40%.

We also weighted image repeats per category per image, instead of only per image as done in Big_Data_Helper.m, as it would have otherwise been possible for a category with a very high mean score to be repeated less times than a different image with a category with a very low mean score, on account of no other features being present.

These scripts also allow the images to be resized. Using the full scale 256x256 or 512x512 images is likely to yield poor results, as they will take longer to train. If the resolution is doubled, a computer will take at least 4x longer to process the dataset. This will also be done with a lower accuracy, as the system will have to find more precise weights. As a good rule of thumb, if it is possible to observe features by eye at a certain resolution, then this is an resolution that can be trained with. In this case, 32x32 images are ideal.

When using any of these scripts, a .mat file containing the raw greyscale images in the same order as the excel sheet is generated. This file can be used to verify the success of an a network with the classifier.py script.

References

[1] Adam Sweetman, Sam Jarvis, Rosanna Danza, and Philip Moriarty. Effect of the tip state during qplus noncontact atomic force microscopy of si (100) at 5 k: Probing the probe. *Beilstein journal of nanotechnology*, 3(1):25–32, 2012.

[2] Jack C Straton, Taylor T Bilyeu, Bill Moon, and Peter Moeck. Double-tip effects on scanning tunneling microscopy imaging of 2d periodic objects: unambiguous detection and limits of their removal by crystallographic averaging in the spatial frequency domain. *Crystal Research and Technology*, 49(9):663–680, 2014.

[3] Mehmet Ozkan, Benoit M Dawant, and Robert J Maciunas. Neural-network-based segmentation of multi-modal medical images: a comparative and prospective study. *IEEE transactions on Medical Imaging*, 12(3):534–544, 1993.

[4] Saikat Basu, Sangram Ganguly, Supratik Mukhopadhyay, Robert DiBiano, Manohar Karki, and Ramakrishna Nemani. Deepsat: a learning framework for satellite imagery. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 37. ACM, 2015.

[5] Nicholas Blackburn, Åke Hagström, Johan Wikner, Rocio Cuadros-Hansson, and Peter Koefoed Bjørnsen. Rapid determination of bacterial abundance, biovolume, morphology, and growth by neural network-based image analysis. *Applied and Environmental Microbiology*, 64(9):3246–3255, 1998.

[6] Zhi-Hua Zhou, Yuan Jiang, Yu-Bin Yang, and Shi-Fu Chen. Lung cancer cell identification based on artificial neural network ensembles. *Artificial Intelligence in Medicine*, 24(1):25–36, 2002.

[7] Henry A Rowley, Shumeet Baluja, and Takeo Kanade. Rotation invariant neural network-based face detection. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 38–44. IEEE, 1998.

[8] William H Wolberg, W Nick Street, and OL Mangasarian. Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates. *Cancer letters*, 77(2-3):163–171, 1994.

[9] William H Wolberg, W Nick Street, and Olvi L Mangasarian. Image analysis and machine learning applied to breast cancer diagnosis and prognosis. *Analytical and Quantitative cytology and histology*, 17(2):77–87, 1995.

[10] BM Gayathri and CP Sumathi. Comparative study of relevance vector machine with various machine learning techniques used for detecting breast cancer. In *Computational Intelligence and Computing Research (ICCIC), 2016 IEEE International Conference on*, pages 1–5. IEEE, 2016.

[11] Richard AJ Woolley, Julian Stirling, Adrian Radocea, Natalio Krasnogor, and Philip Moriarty. Automated probe microscopy via evolutionary optimization at the atomic scale. *Applied Physics Letters*, 98(25):253104, 2011.

[12] David Keller. Reconstruction of stm and afm images distorted by finite-size tips. *Surface Science*, 253(1-3):353–364, 1991.

[13] Igor Aizenberg, Dmitriy V Paliy, Jacek M Zurada, and Jaakko T Astola. Blur identification by multilayer neural network based on multivalued neurons. *IEEE Transactions on Neural Networks*, 19(5):883–898, 2008.

[14] Keras installation, https://keras.io/#installation.

[15] $sourceforge.net/projects/winpython/files/winpython_3.6/3.6.1.0/winpython - 64bit - 3.6.1.0qt5.exe/download$.

[16] Ankit Sachan. Keras tensorflow tutorial: Practical guide from getting started to developing complex deep neural network, http://cv-tricks.com/tensorflow-tutorial/keras/.

[17] Karlijin Willems. Keras tutorial: Deep learning in python, https://www.datacamp.com/community/tutorials/deep-learning-python#gs.2bo0eem.

[18] Keras tutorial: The ultimate beginners guide to deep learning in python, https://elitedatascience.com/keras-tutorial-deep-learning-in-python.

[19] Michael Nielsen. http://neuralnetworksanddeeplearning.com/.

[20] Siraj Raval. https://www.youtube.com/watch?v=voppzhpvtiq&t=1s.

[21] Jason Brownlee. Object recognition with convolutional neural networks in the keras deep learning library, http://machinelearningmastery.com/object-recognition-convolutional-neural-networks-keras-deep-learning-library/.

[22] Rutger Ruizendaal. Deep learning 3: More on cnns & handling overfitting, https://medium.com/towards-data-science/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d.

[23] Keras noise layers, https://keras.io/layers/noise/.

[24] Jason Brownlee. Dropout regularization in deep learning models with keras, http://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/.

[25] Jason Brownlee. How to improve deep learning performance, http://machinelearningmastery.com/improve-deep-learning-performance/.

[26] https://stackoverflow.com/questions/43906048/keras-early-stopping.

[27] http://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/.

[28] Agustinus Kristiadi. https://wiseodd.github.io/techblog/2016/06/22/nn-optimization/.

[29] Francois Chollet. Building powerful image classification models using very little data, https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html.

[30] Spiw toolbox, https://sourceforge.net/projects/spiw/.

[31] Scanning probe image classification, https://www.zooniverse.org/projects/oliver-gordon/scanning-probe-microscopy-image-classification.

[32] Morten Møller, Samuel P Jarvis, Laurent Guérinet, Peter Sharp, Richard Woolley, Philipp Rahe, and Philip Moriarty. Automated extraction of single h atoms with stm: tip state dependency. *Nanotechnology*, 28(7):075302, 2017.