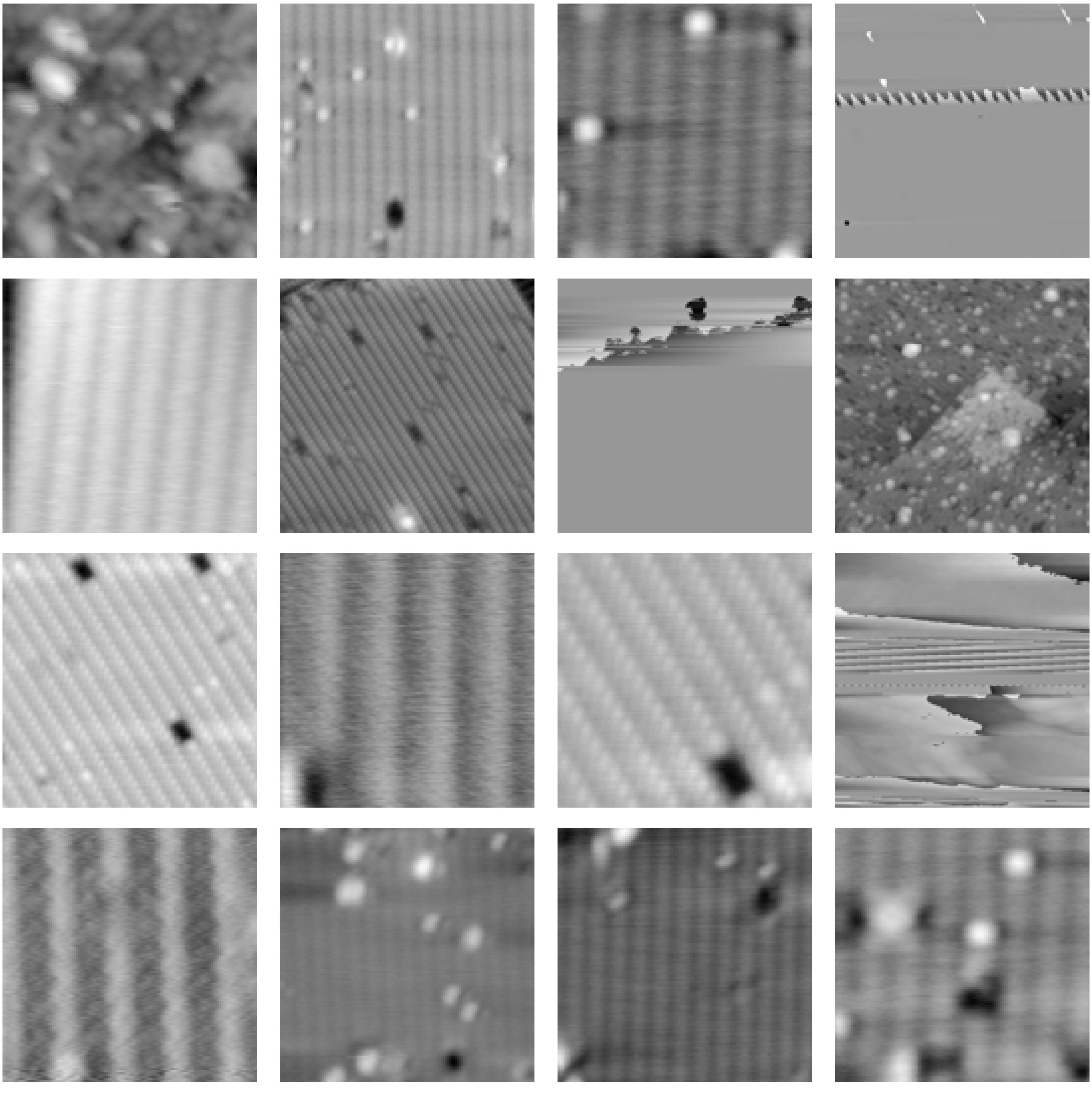
July 31, 2018

STM Image Classification

Pierre d'Hondt



Contents

1. Introduction

The Scanning Probe Microscopy (SPM) is a microscopy technique for mapping the relief or other physical quantity by scanning a surface using a very fine tip (ideally, atomically sharp). When that tip is brought to a few nanometers of the surface, a distance dependent signal is recorded. The SPM includes imaging techniques such as STM, for which the signal will be a quantum tunneling current.

Although the STM has been used daily by researchers for decades, the manufacture of atomically sharp tips is still a very long and redundant process. The tip apex can be refined by etching it, annealing it or crashing it inside a surface, and then given a settle time for it to reach is lowest energy state. Plus in order to obtain a image showing the features of interest of the studied surface, the operator must change a bench of parameters such as scan speed, voltage and tunneling current set point. In order to have a correct working environment, the STM operator must look at the scan results in real time, and launch automated routines with hope to improve the tip if the result is not suitable. A non-suitable image is one being blurred, stretched and/or self repeated [1][2]. Such images are the consequence of the tip not being atomically sharp. This double tip/image optimization process is time consuming and frustrating and would benefit greatly from automation.

Neural networks have been used for image classification in a number of fields, such as medical imaging[3], cell identification [4],[5], satellite imaging[6] and facial recognition[7]. More recently, Wolkow *et al*[8] demonstrated the use of neural networks for distinguish between STM images acquired by single-atom ended tip and by multiple-atoms ended tip on H passivated Si(100) surface.

The aim of this work is to design, develop, train and use Convolution Neural Networks (CNN) for STM images classification. Those trained CNN could then be used along with autonomous STM scanning procedure and tip conditioning routine in order to achieve automatic tip manufacture.

The STM images used for the training of neural nets were recorded by the group of prof P.Moriarty at the University of Nottingham. The images display the H passivated Si(100) plane at variable temperature. The classification of those images into a coherent dataset for neural networks training was made last year by Oliver Gordon and Nicole Landon, 4thyear students at the University of Nottingham. Their work is available on GitHub[9].

After presenting the software/hardware environment used for CNN training, will be presented some of the steps taken during this project, as well as some intermediate results on keras. Finally, the state of the project at the end of it will be detailed, with an explanation of how to use the classification code which achieves **98.2%** accuracy.

2. Environment

As a continuity with the work of past year students, the keras[10] module in python was used to create the neural networks. Keras provide simple frameworks to work with. Keras offers the possibility to use different back end, such as TensorFlow[11] and Theano. It is important to keep the back end consistent between training and classification scripts. As last year project used the TensorFlow back end, that's also the choice for this work.

Keras and TensorFlow can be easily installed on Windows, Linux and macOS following the instructions located on their respective websites.

- `keras.io/#installation`
- `www.tensorflow.org/install/`

Please make sure that the GPU support is enabled as training on GPU is order of magnitude faster than on CPU.
If you only have access to a computer with no GPU, please consider using *Google Colaboratory* to train CNN. For informative purpose, the local environment used during this work is described after a presentation of Colaboratory.

2.1 Google Colaboratory

Colaboratory[12] is a research project created to help disseminate machine learning education and research. It's a Jupyter notebook environment that requires no setup to use. The service is free to use and the code runs on Google servers. All you need is a web browser and a google account.

Colaboratory was created to help machine learning development, so the platform is offering the possibility to run python code on powerful GPU NVIDIA Tesla K80 (4992 CUDA cores ; 24 GB RAM) for free, using the TensorFlow back end.

Colaboratory allows the use of other Google services, especially Google Drive as a permanent storage. Indeed, Colaboratory has your code running on a temporary virtual machine (VM) that is stopped and erased after inactivity. (the test I made showed that the the VM stay active during 1 hour after you close the web connection, aka closing the web browser window). The same thing appends if the web connection is maintained but no instruction is send to the VM. Because of that, you need to periodically save the important files on Google Drive. Code snippets are available.

2.2 Local Environment

Most of the training of the CNN of this work was made on Colaboratory.
However, a local environment was set using the following software. Installation instructions are available on the Keras[10], TensorFlow[11] and NVIDIA websites.

Hardware

CPU	i3-7100 @ 3.9GHz
RAM	8 GB
OS	Windows 10 Enterprise ; Creators Update; x64 ;

Table 2.1: Hardware used

Software

Python libs

Software name	Version	Comments
Python	3.6.5	x64 architecture
CUDA Toolkit	9.0	use the version recommended on TensorFlow web site [11]
CuDNN	7.1	files to add in CUDA folders
NVIDIA drivers for CUDA		

Table 2.2: Software used and versions

Lib name	Version	Comments
TensorFlow [11]	1.8.0	machine learning library
Keras [10]	2.1.6	wraper for TensorFlow
h5py	2.8.0	to save CNN to drive
PyDrive [13]	1.3.1	wraper to acces Google Drive
NumPy	1.14.3	array manipulation

Table 2.3: python libs and versions

3. Playing with Keras

When training CNN for a specific purpose, a lot of parameters can be changed in order to optimize the training time, the final accuracy of the network, or to make the network insensible to important pattern position or orientation.

Those parameters can be part of the CNN itself, like its shape : how many layers, what kind of layer, what size for each layer, etc. Training parameters such as optimizing algorithms used, losses (aka the measure that training aims to decrease) and batch size (aka how many pictures go through the net before modifying it) can also improve the training time, etc. A rapid comparison of the different losses and optimizers that keras offers has been done.

3.1 Training parameters analysis

Some vocabulary from Keras documentation[10] :

Batch : *a set of N samples.* The samples in a batch are processed independently, in parallel. If training, a batch results in only one update to the model. A batch generally approximates the distribution of the input data better than a single input. **The larger the batch, the better the approximation**; however, it is also true that the batch will take longer to process and will still result in only one update.

Sample : one element of a dataset.

Epoch : one pass over the entire dataset.

Loss function : This is the objective that the model will try to minimize

Batch size effect

Batch size is often chosen as a power of 2. The graph 3.1 shows the evolution of the system accuracy every 10 epochs for different batch sizes. Some curves are shorter than other because the measure had to stop if the accuracy decreased 2 times in a row.

We notice on this graph that the bigger the batch size, the sooner high accuracy is reached. Which could lead to a shorter training time if the reach of a certain accuracy value is set as a condition to stop the training.

Optimizer and loss effect

Same kind of experiments was made for different losses and optimizer provided by keras. (graphs 3.2 and 3.3)

On those graphs, most of the optimizer and loss functions test provide the same level of accuracy.

Disclaimer : The "best" optimizer and loss function is very dependent on the context. Similar plots were made with different dataset and different kind of CNN, which lead to other "best" optimizer and loss function. *In other words, always try different parameters with your data.*

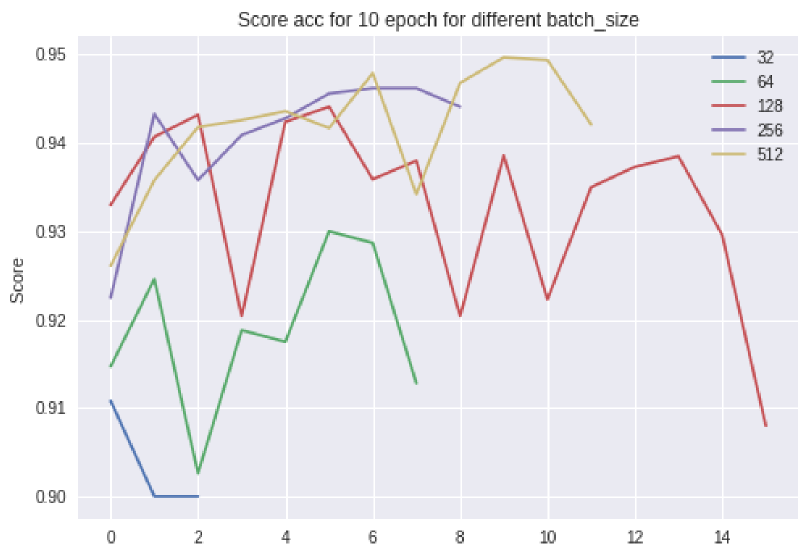


Figure 3.1: Score accuracy for different batch size

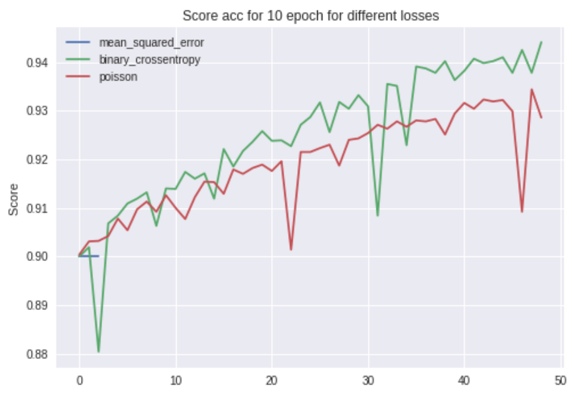


Figure 3.2: Score accuracy for different losses

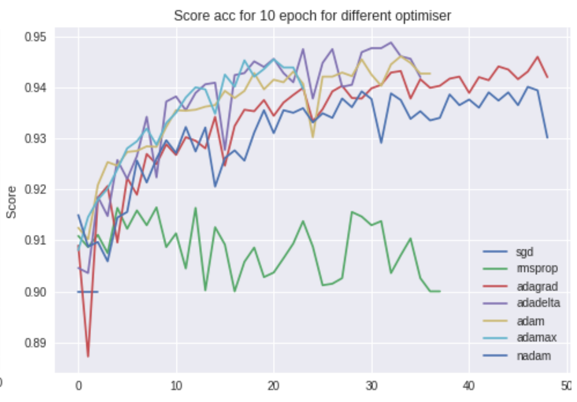
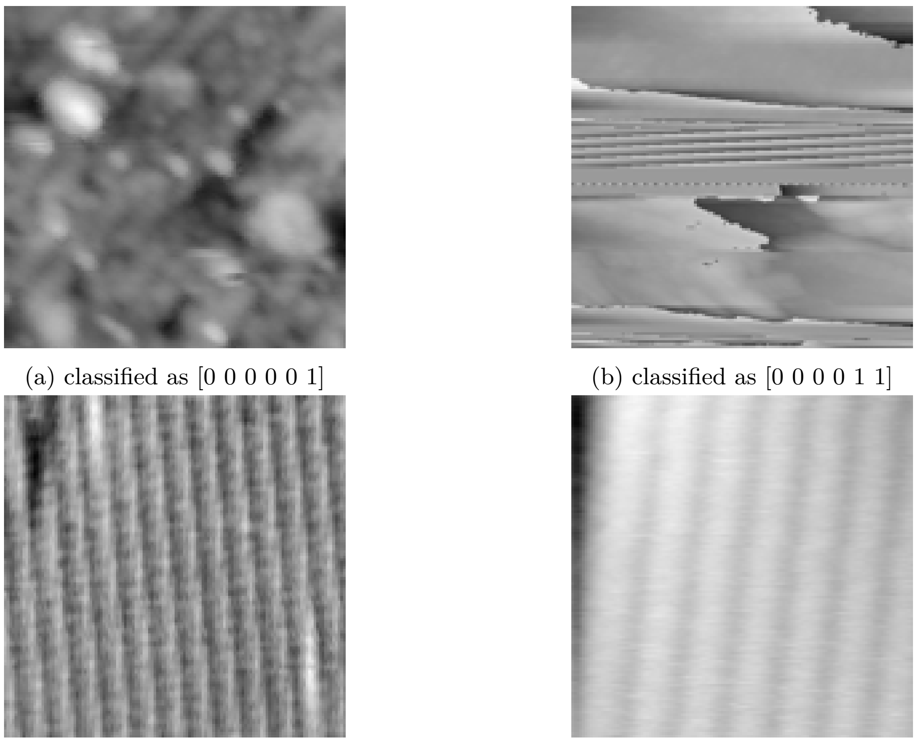


Figure 3.3: Score accuracy for different optimizer


(a) classified as [0 0 0 0 0 1]

(b) classified as [0 0 0 0 1 1]

(c) classified as [.5 0 0 .5 0 1]

(d) classified as [0 0 0 1 0 .5]

Figure 3.4: Some images from the dataset and their classification in the 6 categories of the dataset

3.2 STM Images dataset

The dataset presented before contains 49 690 STM images and their classification into 6 categories. Those categories are :

1. Asymmetries
2. Individual atoms
3. Dimers
4. Rows
5. Tip change
6. Bad/blurry

Each image can be in several categories. As the construction of the dataset was made by averaging the opinion of different person, it is a float between 0 and 1 that represents either or not the image has been classify in a specific category. With **0** being no one classify the image into that category and **1** being everybody does. Because it make no sense that the CNN returns the average opinion of humans, the dataset has been binarized. Every float >0.5 becomes a 1, others become 0.

The complete dataset stores 49 690 images, which are 128*128 pixels, grayscale. Fig 3.4 shows some of the images and their classifications.
The dataset is split in a training set (80% = 39 752 images) and a testing set (20% = 9 938 samples).

More detailed explanation can be found in the original report for the making of that dataset[9].

The part of each category in the dataset can be found in table 3.2.

3.3 Convolution Neural Networks

As mentioned before, the shape of the CNN itself plays a huge role in the efficiency of the model. A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of *convolutional* layers, *fully connected* layers, *pooling* layers and *normalization* layers[14].

Convolutional : Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. The name in keras is **Conv2D**

Fully connected : Fully connected layers connect every neuron in previous layer to every neuron in the next layer. It is the same principle as the traditional *multi-layer perceptron* neural network (MLP)[15]. The name in keras is **Dense**

Pooling : Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer. This is often used in order to reduce input size. For example, *max pooling* returns the maximum value from each of a cluster of inputs it received from the prior layer. Another example is *average pooling*. The name in keras is **xxxPooling2D**

Dropout : Dropout consists in randomly setting a fraction of input units to 0 at each update during training time, which helps prevent overfitting. The name in keras is **Dropout**

Flatten : Flattens the input, from any shape into a 1D vector. The name in keras is **Flatten**

Batch Normalization : Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. The name in keras is **BatchNormalization**

More layers and info can be found in official Keras documentation for layers :
https://keras.io/layers/about-keras-layers/ .

From there, a *model* designates a blueprint to build a CNN. A model contains the shape, aka the list of layers, of a CNN but not the weights of the net.

During this work, 3 CNN models were mostly used. Here comes a brief description of each of them, along with some comments. However, the keras code is easy to read and understand, so a detailed description of the models, aka layers order, etc. will not be made here. Please refer to the CNN generation python script, available on the GitHub repository of this work [16].

3.4 VGG like convnet

This model is from the keras website examples. It is similar to the famous VGG convnet from the Visual Geometry Group at Oxford University. The VGG model has proven its efficiency by its very good performance int the Image Net Large Scale Visual Recognition Challenge 2014 (ILSVRC)[17]. The example code has been modified to allow the model to input the images correctly and have 6 outputs.

3.5 DeepDog

This strange name is from the original article[18] where I found the idea and a code snippet of a high-performance CNN. That model is inspired by the work of a team who developed a CNN to recognize either the input picture shows a hotdog or not. That CNN was design to be run inside a mobile app. Who says mobile environment says light and fast CNN, which are 2 qualities one can use for automatized STM probe preparation. The other advantage of this model is that it was design in order to make binary-classification (aka "it is" or "it is not") from a huge dataset (every real life photographs) with few positive samples (how many hotdog pic do you have compared to everything else ?).

That binary classification will be very useful for the rest of the project as detailed in the Part 3.8 Table 3.1 shows the accuracy of Deepdog models trained for binary-classification over the 6 categories, along with the fraction of the dataset that that category represents.

	Category	Accuracy(%)	% of dataset
1	Asymmetries	98.5	6.3
2	Individual atoms	99.6	7.4
3	Dimers	99.1	34.6
4	Rows	99.4	17.3
5	Tip change	98.3	13.4
6	Bad/blurry	94.2	39.0

Table 3.1: Results of Deepdog CNN and dataset composition
(the total is >100% because some images are is several categories)

3.6 CNN from Wolkow article

That article[8]by Wolkow *et al.* described a CNN used for a very similar topic. The CNN allows an automated routine to provide atomically sharp good STM tip. The CNN is used for distinguish a simple atom ended tip from a more than one ended tip. As the results described in the article are rather good (accuracy >99%) and the topic closely similar, it was worth a try to train that model with our dataset.

But the results were not there. The CNN couldn't improve during training, being stuck at very low accuracy. The results are summarized in table 3.2. We notice that the accuracy results are similar or inferior to a constant output (less than the fraction of the dataset) and far less than random output (<50%)

	Category	Accuracy(%)	% of dataset
1	Asymmetries	6.1	6.3
2	Individual atoms	7.6	7.4
3	Dimers	35.3	34.6
4	Rows	17.5	17.3
5	Tip change	13.3	13.4
6	Bad/blurry	38.7	39.0

Table 3.2: Results of CNN from Wolkow article and dataset composition
(the total is >100% because some images are is several categories)

3.7 Size on disk

A trained VGG model takes **108 MB** on disk, a trained "Wolkow model" takes **211 MB** and a trained Deepdog, only **17 MB**. As Majority Polling will be used, a lot a model will be load in memory so the lighter the model is the faster the polling will be. Plus the **Deepdog** model is the one offering the best results. So this is model used until the end of this work.

3.8 Majority Polling

The majority polling consists in training several CNN for the same goal, make each of them classify an image and use the most represented answer as the final vote. Keras does not provide easy majority polling so I had to develop it. That's where binary classification become useful. If the CNNs can only vote "it is" (represented by a 1) or "it is not" (by a 0), an odd numbers of voting CNNs will always give a clear final vote.

4. Final Family

In that part is presented the "Final Family" which is the group of CNN that gave the best accuracy during the whole project. That family is composed of 18 DeepDog CNN members, distributed in 6 groups. Each group of 3 is trained for binary-classification of a specific dataset category. A majority polling is then made between the 3 votes in order to determine the final vote for that category. Once the 6 groups have "final-voted", the output is returned as the list of final-vote for each category.

That way, the Final Family's output is compatible with how the output are stored in the dataset (see legend of Fig 3.4).

4.1 Results

The global accuracy of the Final Family over the 6 categories is **98.2%**. That means that the Family outputs the same results that the one stored in dataset for the 6 categories at the same time.

4.2 How to use it

Every code and python script discussed here are available on the GitHub repo [16].

The dataset must be split in 4 parts :

x_train **:** contains the picture of the training set in a matrix a shape $(nbOfImages, 128, 128, 1)$.
y_train **:** contains expected outpurs of the training set in a matrix a shape $(nbOfImages, 6)$.
x_test **:** similar to x_train with testing images
y_test **:** similar to y_train with testing expected outputs

The python function $majority\_polling(family\_dict, x\_test, y\_test)$ loads the previously described family and make each of the members of the family make a guess for each of the image in x_test.

If y_test is provided, then an estimation of the accuracy of the family response is computed. If not provided, the decision made by the CNN family is returned as it. This could be used to have the CNN family make a guess for not yet classified images. The *main_evaluate_guess.py* is provided with a already trained Final Family. One can just launch the script to evaluate a whole family or use part of it to have the family make new guesses.

If one wants to train his own family with a new model, or with an other dataset, a *main_training.py* script is provided.

One can simply add his own model generation function in the *CNN_generation.py* file and references his function is the*CNN_global()* function in the same file. A model generation function must return a *compiled* model and must take 2 arguments :

- *input_shape*, a tuple a integer ;
- *output_number*, a integer.

Then the training can simply be started by changing the parameters at the begining og the main part of the file *main_training.py*.

Similar changes need to be made to the file *main_evaluate_guess.py* in order to keep training and use consistent.

List of Figures

List of Tables

Bibliography

[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*, 3:25–32, 2012.

[2] Straton Jack C., Bilyeu Taylor T., Moon Bill, and Moeck Peter. 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.

[3] Mehmed Özkan, 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 3:534–44, 1993.

[4] N Blackburn, A Hagstrom, J Wikner, R Cuadros-Hansson, and PK Bjornsen. Rapid determination of bacterial abundance, biovolume, morphology, and growth by neural network-based image analysis. *Applied and environmental microbiology*, 64(9):3246—3255, September 1998.

[5] Zhi-Hua Zhou, Yuan Jiang, Yu-Bin Yang, and Shi-Fu Chen. Lung cancer cell identification based on artificial neural network ensembles. *Artif. Intell. Med.*, 24(1):25–36, January 2002.

[6] Saikat Basu, Sangram Ganguly, Supratik Mukhopadhyay, Robert DiBiano, Manohar Karki, and Ramakrishna R. Nemani. Deepsat - A learning framework for satellite imagery. *CoRR*, abs/1509.03602, 2015.

[7] Henry Rowley, Shumeet Baluja, and Takeo Kanade. Rotation invariant neural network-based face detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 1998.

[8] Mohammad Rashidi and Robert A. Wolkow. Autonomous scanning probe microscopy in situ tip conditioning through machine learning. *ACS Nano*, 12(6):5185–5189, 2018. PMID: 29790333.

[9] Oliver Gordon and Nicole Landon. Spm-toolbox, may 2018. https://github.com/OGordon100/SPM-Toolbox/.

[10] François Chollet et al. Keras. https://keras.io, 2015.

[11] Google. Tensorflow, may 2018. https://www.tensorflow.org.

[12] Google. Google colaboratory, jun 2018. https://colab.research.google.com.

[13] JunYoung Gwak. Pydrive website, may 2018. https://pythonhosted.org/PyDrive.

[14] Wikipedia. Convolutional neural network - wikipedia, jul 2018. https://en.wikipedia.org/wiki/Convolutional_neural_network.

[15] Wikipedia. Multilayer perceptron - wikipedia, jul 2018. https://en.wikipedia.org/wiki/Multilayer_perceptron.

[16] Pierre d'Hondt. Cnn-familypolling, jul 2018. https://github.com/piedho/CNN-FamilyPolling/.

[17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[18] Tim Anglade. How hbo's silicon valley built "not hotdog" with mobile tensorflow, keras & react native, jun 2018. https://medium.com/@timanglade/how-hbos-silicon-valley-built-not-hotdog-with-mobile-tensorflow-keras-react-native-ef03260747