Using Deep Learning to classify AFM Images

Jardi Timmerhuis

July 21, 2020

1 Introduction

In this report we will discuss how we have applied techniques form deep learning to classify Atomic Force Microscope (AFM) images as either 'Good' or 'Bad'. 81 labelled images were used, consisting of 44 good images and 37 bad images. A Graphical User Interface (GUI) has been made as well, which will be discussed briefly in appendix A. The code and a README on how to use the code can be found in the GitHub Repository [1].

1.1 Deep Learning

We will assume in this report that the user has some knowledge of deep learning and neural networks. However we will provide a brief introduction on deep learning here. The information in this section is adapted from the book by Nielsen [2]. For a more elaborate description of the theory on neural networks than the one presented in this chapter, the reader is encouraged to read this book or, for a more technical overview, the book by Goodfellow, Bengio and Courville [3].

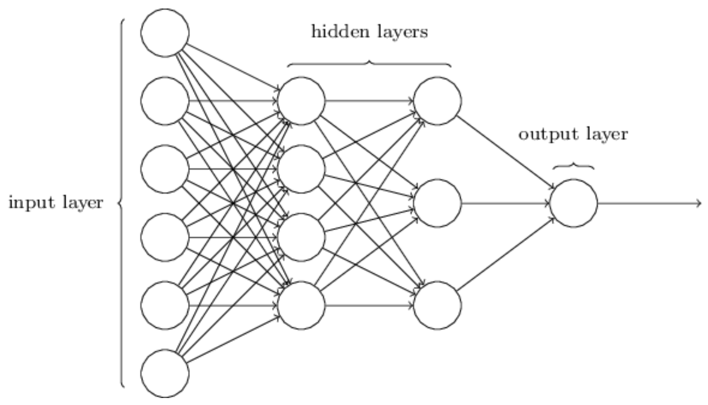

Figure 1: A feedforward neural network with two hidden layers [2].

Figure 1 shows a common feedforward neural network. There are several layers with each layer consisting of multiple nodes. Each node has some input vector $\vec{x}$ and output a described by

$$a = f(\vec{w} \cdot \vec{x} + b)\,, \tag{1}$$

with $\vec{w}$ weights corresponding to each input in $\vec{x}$, b a bias term and $f(z)$ an activation function, typically a sigmoid, hyperbolic tangent or Rectified Linear Unit (ReLU). The outputs of each node in a layer make up the inputs to the nodes in the next layer, as indicated schematically in figure 1. The weights and bias terms can be altered during training, which changes the output of the network. In a way the neural network is a function approximator, which makes it very suitable for classification.

The layers described above are fully connected feedforward layers. Other layers, such as convolutional and pooling layers, also exist, but will not be discussed in detail here. Convolutional layers are very important for feature learning in image classification, so they have been used in this project. There are many online resources with detailed explanations of convolutional neural networks [2–4].

During supervised learning, labeled data is used for training. Each input has a corresponding label. The network has as many output nodes as labels, i.e. there are two output nodes if you have the labels 'dog' and 'cat'. To train the network a cost function or loss function is defined, which represents the error from the desired output. Common costs functions are the Mean Squared Error function (MSE), $C(\vec{w}, b) = \frac{1}{2n}\sum_x \sum_j \left\| y_j - a_j^L \right\|^2$, and the cross-entropy cost function, $C(\vec{w}, b) = -\frac{1}{n}\sum_x \sum_j [y_j \ln(a_j^L) + (1 - y_j)\ln(1 - a_j^L)]$, with a_j^L being the output and y_j the desired output of the jth node in the output layer given input $\vec{x}$. When you minimise the cost function, $\vec{a}^L$ will be as close as possible to $\vec{y}$.

There are several algorithms to minimise the cost function, with the most popular being gradient descent. For instance, in gradient descent a weight w_k is updated according to

$$w_k \rightarrow w_k' = w_k - \eta \frac{\partial C}{\partial w_k}\,, \tag{2}$$

and a bias term b_l according to

$$b_l \rightarrow b_l' = b_l - \eta \frac{\partial C}{\partial b_l}\,, \tag{3}$$

with η the learning rate. In this report we will use the gradient descent with momentum algorithm [5] and the Adam algorithm [6].

2 Methods

In this section we will discuss some techniques and methods that were used in this project. A lot of these methods are used a lot in machine learning.

2.1 Train and Validation Data

It is very common to split the labeled data, in our case AFM images, in a training data set and validation data set. The training data set is then used to train the data: the weights and biases are updated according to a learning algorithm such as gradient descent. Very often the model is overfitting on the training data. An example of overfitting is shown in figure 2. If we use only training data to train we could achieve close to 100% accuracy, however this would also include outliers and wrongly labeled data, so the model would not be able to generalise very well.

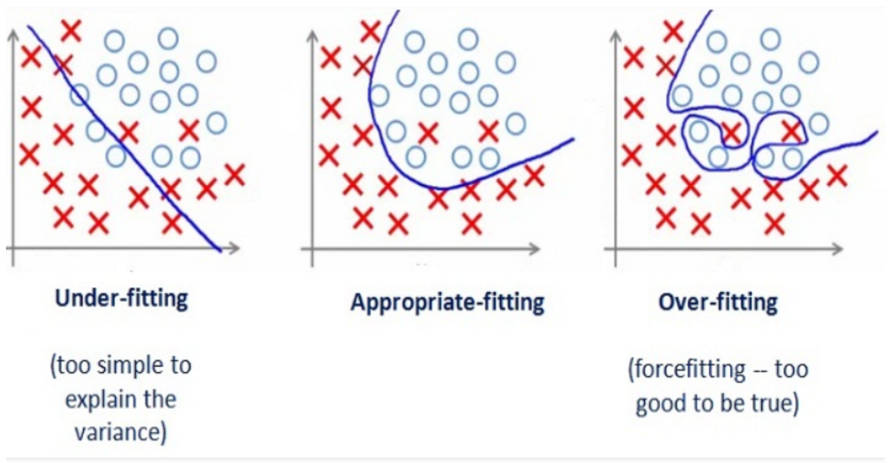
Under-fitting
(too simple to explain the variance)
Appropriate-fitting
Over-fitting
(forcefitting -- too good to be true)

Figure 2: A simple example showing an underfitted and overfitted model [7].

To get a good indication of the accuracy of the model, a validation data set is used. Validation data is not used during training and can therefore give a good indication of the accuracy: an overfitted model would result in a lower accuracy than a well fitted model. It is important that the validation set is similar to the training set. You cannot have a different ratio of 'good'/'bad' labels in the training set compared to the validation set for instance. More information on reducing overfitting (or underfitting) can be found in online resources [7].

In our case we split the data into a training set with 60 AFM images, of which 32 are labeled as 'good', and a validation set of 21 images, of which 12 are 'good'.

2.2 Transfer Learning

A large training data set often reduces variance and therefore overfitting. When you have a small data set it is often difficult to train a good model and analyse their performance. Transfer learning offers a solution. In transfer learning a network pre-trained on a large dataset, such as ImageNet [8], is used.

We can then either use the network as a fixed feature extractor or we can fine-tune the whole network. When using the network as a fixed feature extractor we replace the output layer and retrain only the output weights. When fine-tuning the whole network we use the pre-trained weights as initial weights and fine-tune each and every weight in the network. Fine-tuning on a small data set can lead to overfitting, so fixed feature extraction is often encouraged. More can be found in the Stanford CS321n course notes [9].

2.3 Preprocessing Data

In order to use a neural network the images have to be preprocessed. For instance, they all need to be the same size. In our case, most AFM images have a colour scale bar at the right of the image. We remove this by cropping the right 30% of the image. The small edge of the image is then rescaled to 256 pixels and a 224x224 crop is taken from the rescaled image. During preprocessing of training data the crop is taken to be random, however for the validation data the crop is always taken from the center of the image. This is to ensure that the validation data always gives the same result. The training images are also flipped horizontal 50% of the time. An illustration of these preprocessing transformations is shown in figure 3.

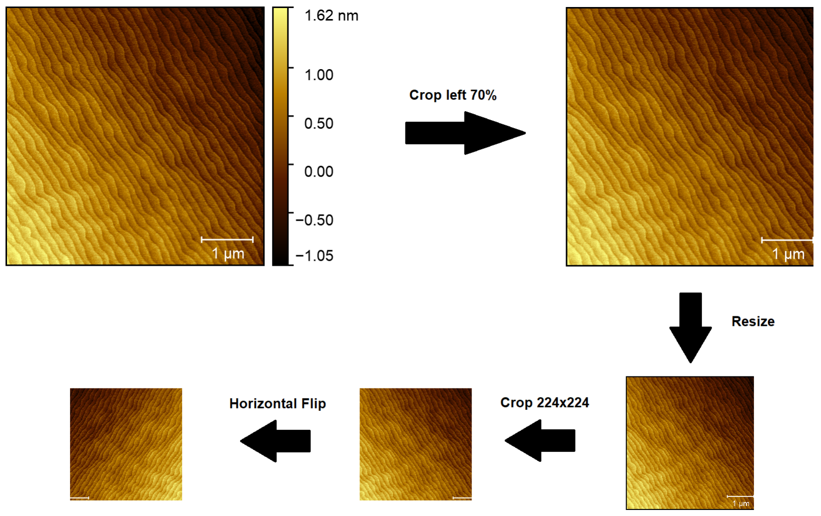

Figure 3: An illustration of the preprocessing transformations.

The image then needs to be converted to a Tensor, which is a matrix-like object used in neural network programming modules such as PyTorch. This Tensor contains three channels with the RGB intensities of each pixel. Finally, each channel is normalised with a certain mean and standard deviation. The resulting Tensor is the input of the network.

2.4 Training

During training there are several (hyper)parameters one can vary. Here we will discuss our training procedure quite generally. We use Python with the PyTorch module, which is made for training neural networks. We use two training algorithms: gradient descent with momentum [5] and the Adam algorithm [6] with the default parameters stated in the papers and given by the PyTorch module. One parameter you can vary in these algorithms is the learning rate. If the learning rate is too high, the training won't converge. However, if it is too low, training would take a very long time and the resulting error could get stuck in a local minimum. PyTorch provides schedulers to adjust the learning rate. We use StepLR with a step size of 7, which decays the learning rate every 7 epochs. Another scheduler we use is ReduceLROnPlateau with a patience of 3, which reduces the learning rate if there hasn't been any improvement in the loss for three epochs.

Here, an epoch is finished after each image in the training data set has been used to update the weights and bias terms. We often trained for 25 or 50 epochs and saved the model with the best validation accuracy. One can use one or multiple images each update step, which is called the batch size. We used batch sizes of 1, 4 and 20 images.

2.5 Used Networks

We have used three different networks to classify AFM images as either good or bad. The first network we designed ourselves. It is a convolutional neural network consisting of a convolutional layer with 3 input channels and 6 output channels, followed by a max pooling layer, then a convolutional layer with 6 input channels and 16 output channels, followed again by a max pooling layer. Both convolutional layers have a 5x5 kernel and both pooling layers a 2x2 kernel. This is then connected to two fully connected layer with respectively 120 and 84 nodes. Both layers have a ReLU activation function. Finally, we have a linear fully connected output layer with 2 nodes, each node corresponding to one of the labels.

The second and third networks use transfer learning. We use the resnet18 architecture [10], which has been trained on the ImageNet data set [8], as the pre-trained network. We use the network as a fixed feature extractor, dubbed FeatureNet, and we fine tune the whole network, dubbed FineNet. When using this network as a pre-trained network it is important that the network is normalised according to the ImageNet normalisation, which normalises the ImageNet data set to mean 0 and standard deviation 1.

3 Results

Table 1 shows the result of differently trained models. ConvNet was able to reach a maximum accuracy of 76%. However, the networks that used transfer learning were ablee to reach an accuracy of 90%. It is important to remind ourselves, however, that due to the small validation data sets these accuracies are inaccurate itself, but it gives a good indication of the possibilities.

It seems that Gradient Descent is mostly better than Adam, however this may be due to overfitting. A batch size of 4 seems reasonable. The scheduler has little influence. Finally, in the case of ConvNet ImageNet normalisation performed better than normalising our own data set to mean 0 and standard deviation 1.

Network	**Train algorithm**	**Batch Size**	**Scheduler**	**Normalisation**	**Accuracy**
FeatureNet	Gradient Descent	4	StepLR	ImageNet	90%
FeatureNet	Gradient Descent	4	ReduceLR	ImageNet	90%
FeatureNet	Adam	4	None	ImageNet	90%
FineNet	Gradient Descent	4	None	ImageNet	90%
FineNet	Gradient Descent	4	StepLR	ImageNet	90%
FineNet	Gradient Descent	4	ReduceLR	ImageNet	90%
FineNet	Gradient Descent	20	ReduceLR	ImageNet	90%
FeatureNet	Gradient Descent	4	None	ImageNet	86%
FeatureNet	Gradient Descent	4	StepLR	ImageNet	86%
FeatureNet	Gradient Descent	20	None	ImageNet	86%
FeatureNet	Gradient Descent	20	ReduceLR	ImageNet	86%
FeatureNet	Adam	4	StepLR	ImageNet	86%
FeatureNet	Adam	20	ReduceLR	ImageNet	86%
FineNet	Gradient Descent	4	StepLR	ImageNet	86%
FineNet	Gradient Descent	20	None	ImageNet	86%
FineNet	Gradient Descent	20	StepLR	ImageNet	86%
FineNet	Adam	4	StepLR	ImageNet	86%
FineNet	Adam	20	None	ImageNet	86%
ConvNet	Gradient Descent	4	None	ImageNet	76%
ConvNet	Gradient Descent	4	StepLR	ImageNet	76%
ConvNet	Adam	1	None	ImageNet	76%

Table 1: Results for multiple models trained with different algorithms and parameters. For FeatureNet and FineNet only the models with an accuracy of 86% or higher are shown.

4 Conclusions and Outlook

From our small network trained from scratch – ConvNet – we can clearly see it is possible to distinguish bad from good AFM images. However, the validation data set is small, so the accuracy might be skewed upwards. Similarly, the training data set is small, so there is some concern for overfitting.

Tranfer learning greatly improves results: it is possible to train a model with 90% validation accuracy. Both FineNet and FeatureNet are able to produce these results. It is important to keep in mind that finetuning a network on a small data set is susceptible to overfitting, so FeatureNet is more suitable for our application.

We have not given much attention to overfitting. Looking into possibilities to reduce overfitting would give more accurate models. Most importantly, the size of the data set needs to be increased, both to reduce overfitting and to give a better indication of the accuracy.

This project shows that it is possible to classify AFM images using neural networks, however the resulting accuracies should be taken with a grain of salt.

References

1. Timmerhuis, J. *AFMNet* `https://github.com/JardiT/AFMNet`. 2020.
2. Nielsen, M. A. *Neural Networks and Deep Learning* (Determination Press, 2015).
3. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* `http://www.deeplearningbook.org` (MIT Press, 2016).
4. Saha, S. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* `https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53` (2019).
5. Qian, N. On the momentum term in gradient descent learning algorithms. *Neural networks* **12,** 145–151 (1999).
6. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
7. Bhande, A. *What is underfitting and overfitting in machine learning and how to deal with it.* `https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76` (2020).
8. Deng, J. *et al. ImageNet: A Large-Scale Hierarchical Image Database* in *CVPR09* (2009).
9. Li, F.-F., Krishna, R., Xu, D. & Byun, A. *CS231n: Convolutional Neural Networks for Visual Recognition* `https://cs231n.github.io/transfer-learning/` (2020).
10. He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition. *CoRR* **abs/1512.03385.** arXiv: `1512.03385`. `http://arxiv.org/abs/1512.03385` (2015).

A GUI

In order to easily easily use the trained model architectures we created a Graphical Userin Interface (GUI). In this GUI we can load a pre-trained model and an AFM image. We can then predict if the image is a 'bad' AFM image or a 'good' AFM image. A screenshot of the GUI is shown in figure 4.

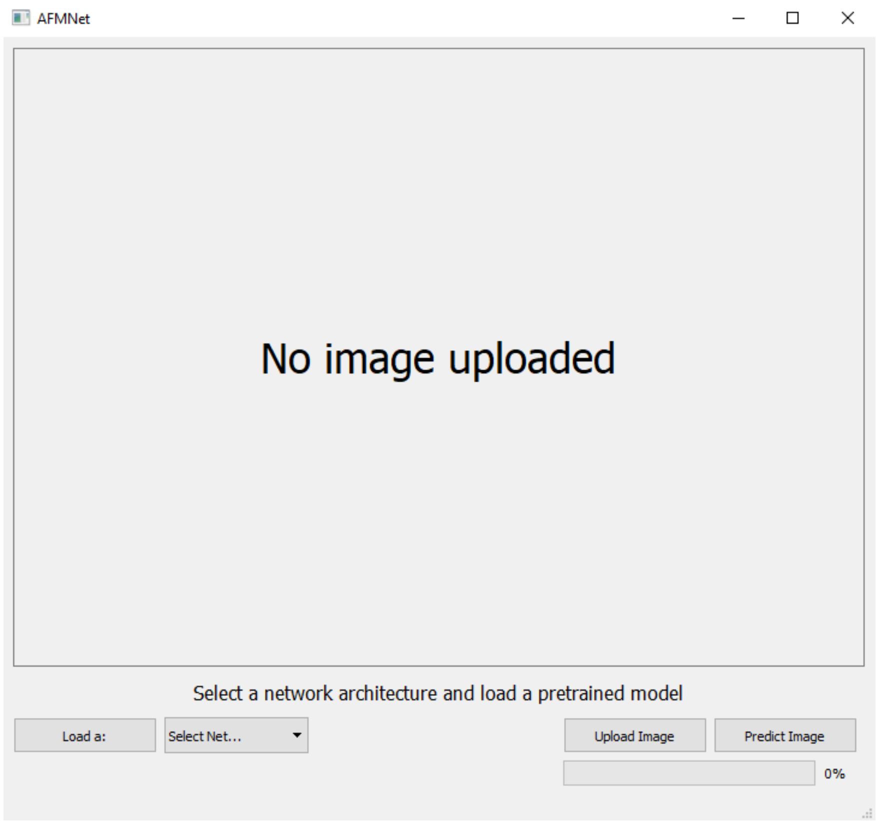

Figure 4: A screenshot of the GUI.

The GUI has three buttons: one to load a pre-trained model, one to load an AFM image and one to predict the label of that image. To load a pre-trained model one must first select a model type – ConvNet, FeatureNet, or FineNet – in the 'Select Net...' dropdown menu. You can then click on the 'Load a:' button to load a pre-trained model. One can use the Upload Image button to upload an image. Finally, you can use the Predict Image button to predict if the image is a 'good' or 'bad' AFM image. This button will not work if either the model or image is not loaded. More information on how to use the GUI can be found in the GitHub repository [1].