PhUSE EU Connect 2018

Paper ML02

Want to get started with deep learning? Prototyping a deep learning image classifier

Thomas Ellebæk, Ferring Pharmaceuticals A/S, Copenhagen, Denmark

ABSTRACT

Machine learning, and more specifically deep learning, is becoming increasingly popular across industries and something many pharmaceutical companies would like to explore. For programmers working in the pharmaceutical industry understanding how machine learning works and how to get started is therefore becoming more and more relevant. This paper outlines the pathway from idea to fully functional prototype of an image classifier based on convolutional neural network (CNN) techniques, using Python, the TensorFlow™ framework and the Keras API.

INTRODUCTION

Humans are very good at detecting patterns in images. Based on just a few examples and an anticipated context, humans can learn sophisticated rules governing a set of images. Machines on the other hand, need a lot more examples to learn the same rules. However, machines have some competitive advantages to humans, like scalability and consistency. So what kind of tasks are appropriate to delegate to a computer? Obviously it needs to be tasks that a machine can learn. First of all, it can be tasks that are unattractive to humans because they are repetitive or dangerous to carry out. Secondly, it could also be tasks that require highly specialized skills where it is costly to train a human expert, tasks where human involvement has become a bottleneck, or tasks where we wish to eliminate human subjectivity.

The boundary for what a machine can learn is constantly being pushed, and in areas like computer vision and natural language processing, which were previously reserved for humans, algorithms utilizing deep machine learning techniques have now shown great advances.

This paper provides an introduction to some of the key concepts related to deep learning image classification and a concrete example using Ferring data.

CLASSIFICATION

Classification is the task of looking at a specific input example $x' \epsilon\ X$ and estimating the correct output class $\hat{y}' \epsilon\ \{C_1, \dots, C_K\}, K \geq 2$. An example is classifying an incoming email into spam or not spam. Assuming there is a pattern to the mapping of interest, $f: X \rightarrow \{C_1, \dots, C_K\}$, the objective is to identify the best model $\hat{f}$ subject to a selected loss function or performance metric, e.g. the proportion of incoming emails correctly classified.

CLASSIC MACHINE LEARNING VERSUS DEEP LEARNING

The task of spam filtering can effectively be solved with classic shallow machine learning. Indeed, Naïve Bayes is known to be a good model type for separating spam from not spam emails. As illustrated in Figure 1, with classic machine learning it is important to hand design intelligent features based on the raw email input, which the model can utilize to distinguish between spam and not spam. That is, we seek the best possible representation of the input, in order to optimize the likelihood of the algorithm establishing a high performing mapping from feature space to output. In machine learning projects, feature engineering is the key to success, and typically where most of the effort goes [1].

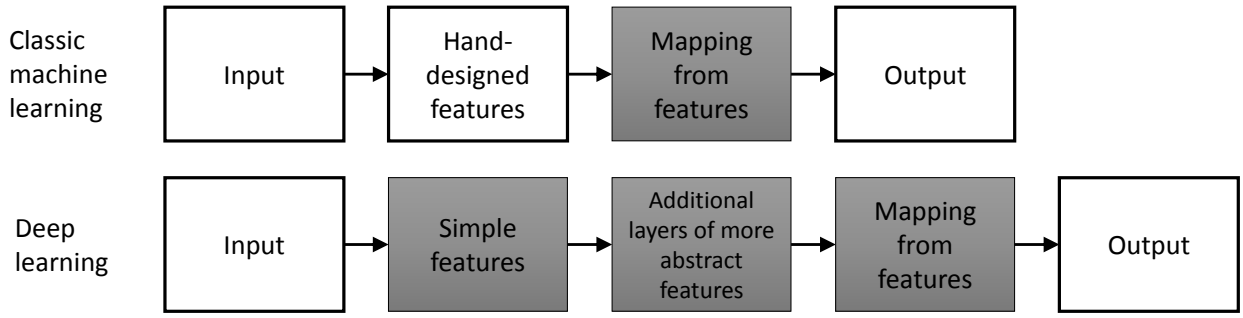

Figure 1: Flowchart showing the difference between two machine learning methodologies. Shaded boxes indicate components that are able to learn from data [2].

To a great extent deep learning has removed the problem of hand designing intelligent features [2]. The impressive advances in fields like computer vision and natural language processing are exactly due to the invention of model architectures enabling optimization algorithms to learn sophisticated representations of the input as part of the model training.

BREAKTHROUGH

2012 was a breakthrough year for deep learning in computer vision. For the first time, a deep convolutional neural network (CNN) model won a computer vision competition, and in the same year two additional contests were won by CNN models, showing great advances in three different disciplines; classification, object detection and image segmentation [3]. The most famous achievement is probably the first place in the ImageNet classification challenge, obtained by Alex Krizhevsky *et al.*[4], with a model which has become known as the AlexNet.

NEURAL NETWORKS

The goal with a neural network (NN) is to approximate some function f. The simplest form is usually called a fully connected feedforward neural network and is characterized by all neurons from each layer being connected to all neurons in the next layer, as illustrated in Figure 2. The first layer is called the input layer, the last layer is called the output layer and all layers in-between are called hidden layers.

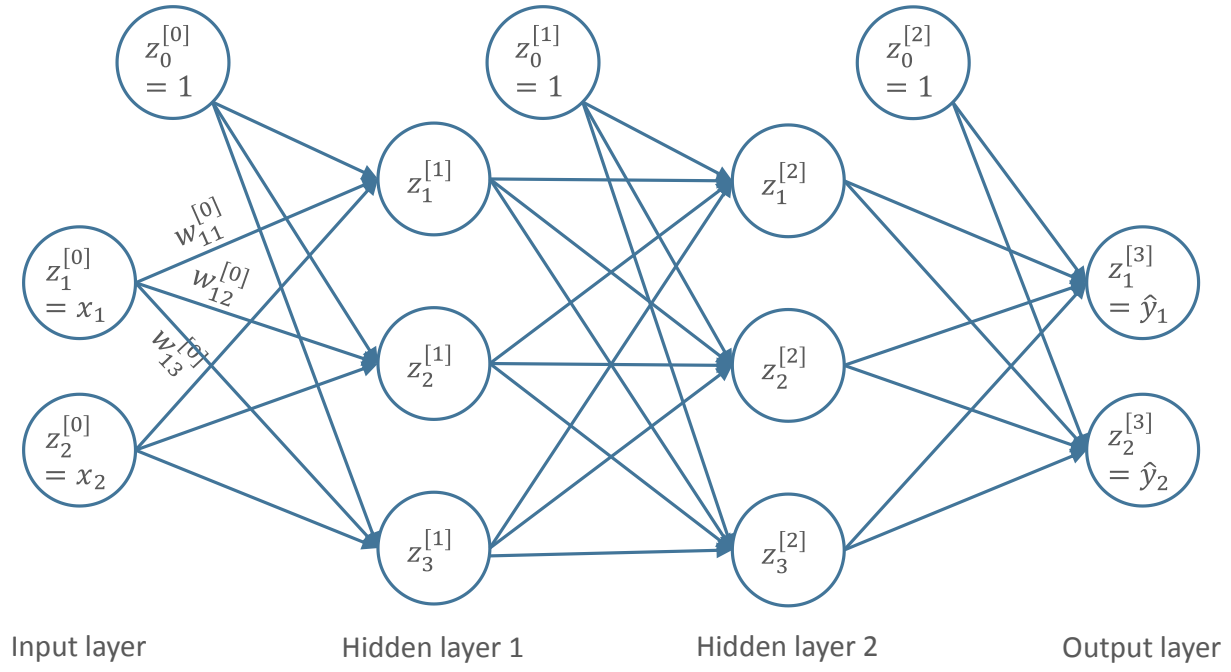

Figure 2: A 3-layer fully connected feedforward neural network with 2-dimensional input and 2-dimensional output. $w_{kj}^{[l]}$'s are learnable parameters (weights) to be estimated by an optimization algorithm and $z_0^{[l]} = 1$ represents bias terms associated with each layer. There is a weight on all arrows in the graphs, but for simplicity only three are labelled in this figure.

The input layer represents the input features $x \in \mathbb{R}^{d^{[0]}}$. The output of the j^{th} neuron in the l^{th} layer is usually defined as

$$z_j^{[l]} = g\left(\sum_{k=0}^{d^{[l-1]}} w_{kj}^{[l-1]} z_k^{[l-1]}\right), \qquad (1)$$

where g is some real valued non-linear differentiable activation function which can vary from layer to layer, $w_{kj}^{[l-1]}$ is the weight associated with k^{th} input to the j^{th} neuron in the l^{th} layer, d^l is the dimensionality (number of neurons) of the l^{th} layer and $z_0^{[l-1]} = 1$ represents the bias term associated with each neuron in the l'th layer.

Fitting a model is an optimization exercise where the weights of the model are iteratively updated until some stopping criterion is met. First the weights are assigned a non-zero initialization, then for each iterative update, training data is forward passed through the network and the distances (errors) to the correct labels are calculated and aggregated in the loss function. The partial derivatives of the loss function with respect to the individual weights are obtained by propagating the errors backwards through the network, and the weight vector is then updated by using the weight's gradient, before next forward pass is taking place. This type of gradient based optimization comes in many different forms, of which the most popular are readily available in the deep learning software libraries. In the example in this article we will be using Adam-optimization.

CONVOLUTIONAL NEURAL NETWORKS

CNNs is a special kind of NN, and has since the breakthrough in 2012 been state-of-the-art for computer vision tasks, leveraging three important ideas [2];

- Sparse interactions; not fully connected, reducing memory requirements and improves efficiency,
- Parameter sharing; kernels used across all locations, reducing memory requirements and improves efficiency, and
- Equivariant to translation; model not sensitive to translation.

The general idea behind CNNs is to obtain a representation of the input by applying several layers of convolution (and pooling), before a standard fully connected feedforward NN takes care of the final mapping to the output, as illustrated in Figure 3.

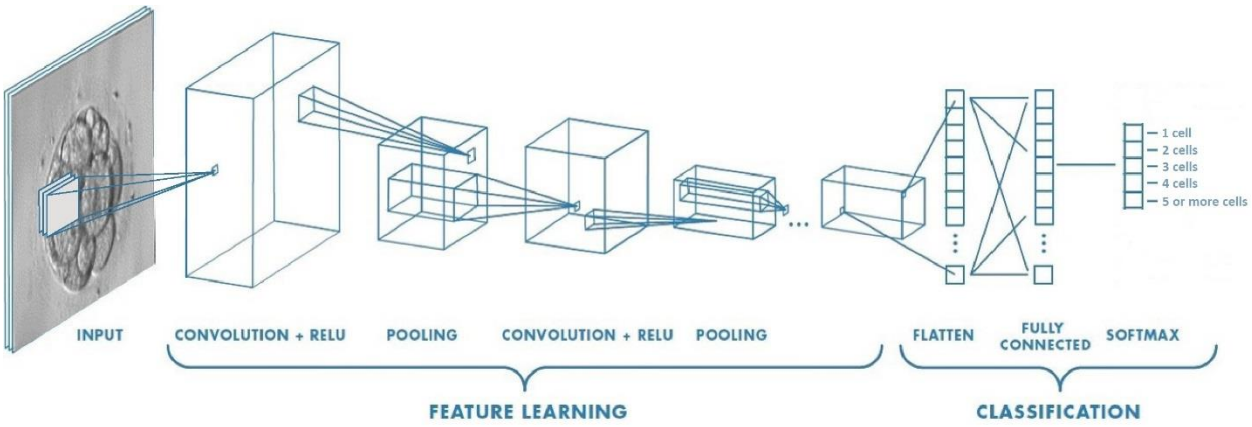

Figure 3: Simple CNN architecture illustrating the flow from original image pixel values through feature learning to a final classification step, mapping the learned features to the output, using a fully connected feedforward neural network like the one shown in Figure 2. The figure is modified from [9].

A convolution (CONV) is a mathematical operation between an input function $I(t)$ and a kernel $K(t)$ and is denoted $(I * K)(t)$, where t usually represents time or location. In continuous time a convolution is an integral transformation. In machine learning the input is usually multidimensional and the location variable take only discrete integer values. In the case of a grayscale image, the input will be 2D (height and width) and the convolution operation is defined as

$$CONV: \ (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n), \qquad (2)$$

where (i, j) is a location in the convolution and (m, n) iterates over the dimensions of the kernel (height and width). Most NN libraries however, implements cross-correlation (CC) instead of actual convolution, because it is more convenient to implement, it preserving the desirable property of convolution and it makes the kernel easier interpretable

$$CC: \quad (I \otimes K)(i,j) = \sum_m \sum_n I(i+m, i+n) K(m,n). \qquad (3)$$

A convolutional layer of a CNN involves several kernels, learning different properties of the input. Each kernel at each location of the input, produces input to a neuron in the next layer, and as in standard feedforward NNs, a bias term is added and the sum is passed to a non-linear activation function. If input to the first layer is $100 \times 100 = 10.000$ pixels and we want to learn 64 kernels of size 5×5, the dimensionality of the output of this layer is $(100 - 5 + 1) \times (100 - 5 + 1) \times 64 = 589.824 \gg 10.000$. Hence, a convolution layer is usually followed by a pooling layer, which reduces the dimensionality by only keeping the maximum over a nearby region of the map produced by each kernel. For each training example and each kernel, max-pooling corresponds to discarding all but the strongest activation over a nearby region. The fitting procedure (learning the kernels), is similar to the feedforward NN, i.e. gradient-based learning where each cell in each kernel correspond to one weight.

THE TASK AND THE DATA

The example shown in Figure 3 is used in the rest of this article. We want to create an image classifier that based on an image of a human embryo can determine how many cells the embryo has.

The target is to become as good as the expert human embryologist at cell counting. The proof of concept (PoC) is carried out on images of embryos that are two days old. Each image has been evaluated by three independent embryologists and their consolidated evaluation is considered the true label. Figure 4 shows a small sample of the dataset.

Figure 4: Five training examples with corresponding class label (number of cells at day 2).

As illustrated in Figure 4, counting cells in human embryos is not a trivial task. However, since the human embryologist can determine the number of cells based on an image, we believe that with a sufficient number of training examples, a machine will be able to learn the same task, with the benefit of being scalable, more consistent and faster. The variation in zoom, darkness etc. within the images can pose a problem for the algorithm, which needs to be addressed in a proper preprocessing step.

The data available for the PoC constituted 3124 images, of which 2246 were used for training, 440 for validation and 438 for testing. The distribution of number of cells at day 2, over the training dataset, is illustrated in Figure 5.

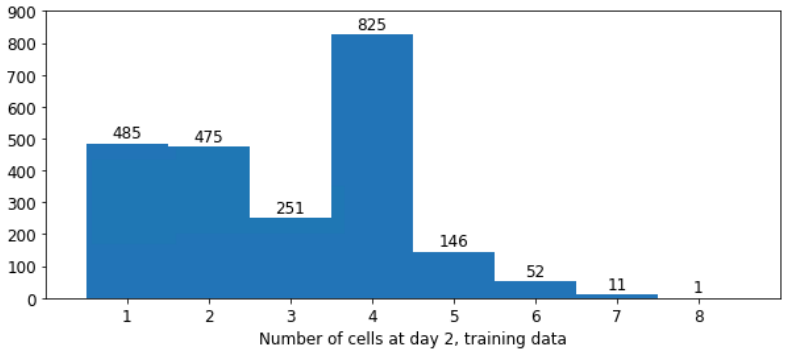

Figure 5: Distribution of number of cells over the training dataset. When modelling, and subsequently predicting, we will group number of cells greater than or equal to 5 into one group.

COMPUTING ENVIRONMENT

We decided to work with TensorFlow™ and Python due to an expert recommendation[1], and eventually we went on to use the Keras API. Downloading and installing Python, TensorFlow™ and Keras is easily done with the Anaconda data science platform, although there is a risk that company IT-policy or security rules makes life a bit difficult.

For the PoC we have been working out of Microsoft Azure, where we can configure our own Deep Learning Virtual Machines (DLVM) with storage accounts and graphical processing units (GPU) for accelerated computing. On the Microsoft DLVM a series of data science tools and languages are pre-installed, including Anaconda, Python, TensorFlow™ and Keras. Other platforms and IDEs exists for prototyping deep learning projects, but these have not been considered for this exercise.

Access to GPU-accelerated computing is important for deep learning projects. There exist a lot of benchmark calculations comparing CPU measurements with various GPU-configurations, but none more reliable than your own of course. What we found was that even for our rather simple experiments with the CNN PoC, and the smallest DLVM available (Standard_NC6), the time for running one epoch (using all training examples once) fitting a model of 2 million parameters was reduced by a factor 40, from approximately 285 to 7 seconds per epoch. For more details see Table 1.

Machine	s/epoch
Intel® Core™ i7-6500U CPU @ 2.50GHz, 4 logical processors, 16 GB DDR3 RAM	285
Azure Standard_NC6, 6 CPU, 1 GPU (one-half NVIDIA Tesla K80 card), 56 GiB RAM	7

Table 1: Training time comparison for CNN with three convolutional layers followed by batch normalization and max pooling. Input size (206x260)x2246, batch size 32 and 2,048,421 trained parameters. Network is illustration in Figure 6.

Running our experiments on the DLVM allowed more experimentation, faster iterations and better use of the training data, running more epochs with and without data augmentation. But a development environment on a local laptop also has its advantages, being accessibly offline and being sufficient for most data cleaning, preprocessing and initial model experimentation.

TRAINING AND MODEL SELECTION

Khan *et al.* [5] has previously build a deep learning cell counter reusing the architecture of the earlier mentioned AlexNet [5]. Although AlexNet these days might be considered a small network with 5 convolutional layers and 3 fully connected layers, the training involves estimating 62,378,344 model parameters [6]. This was a breakthrough architecture in the ImageNet competition, where a dataset of more than 1.2 million high resolution training images in 1,000 categories were provided.

Khan *et al.* chose to construct a 5-class cell counter, classifying images with labels $\{1, 2, 3, 4, \geq 5\}$, where ≥ 5 is the class of 5 or more cells. Khan *et al.* had access to 265 embryo development movies consisting of a total of 148,993 frames. Their training strategy involved an undisclosed form of data augmentation and they successfully trained a model of the same architecture as the AlexNet. Khan *et al.* obtained an overall accuracy of 87.36% with this approach, and with further experimenting, they succeeded obtaining an overall accuracy of 94.05%, computing a bounding box enclosing the embryo, by incorporating information about the estimated cell counts of the nearby time frames and imposing the biological constraint that cell counts should be monotonically non-decreasing over time [3].

For this PoC we initially tried to follow a similar approach as Khan *et al.*, using the same labelling, starting out with the same network architecture and applying live batch data augmentation (rotation, zoom, width/height shift and horizontal flip). It turned out that AlexNet architecture and Adam-optimization with standard settings, could not learn from our data, supposedly due to the relative limited size compared to the data used by Khan *et al.*

It seemed natural to try to reduce the network until patterns were detectable. Experimenting with different model architectures and modern techniques like batch normalization, drop out and zero padding eventually led to a pattern being detected in the training data and the training error started to drop.

As Ali Rahimi [7] writes, *"machine learning has become alchemy"*, meaning that we have discovered techniques that work amazingly well, but we still know very little about *why* some of these operations work. It is somewhat unsatisfying that even small changes to a model which is able to fit training data, can lead to an untrainable model. Although understanding deep learning requires a certain skill-level in linear algebra, probability theory and numerical computations, it seems like empirical research is equally important.

[1] Christian Igel, Professor in Machine Learning at Department of Computer Science at University of Copenhagen.

FINAL MODEL

The final model for our PoC is illustrated in Figure 6. It consist of 3 convolutional layers and 2 dense fully connected layers. All convolutional layers are configured identically except for the number of kernels. We use kernels of size 5x5, a stride of 2x2 and no padding, reducing the height and width slightly more than a factor two. Convolution is followed by batch normalization before rectified linear unit (relu) activation. Each convolutional layer is followed by max pooling with dimension 2x2, again reducing the height and width by a factor two. Softmax is applied in the end to obtain the relative probabilities of the image belonging to each of the five classes described above. For the iterative gradient-based model fitting, Adam-optimization with standard settings and a batch size of 32 is used.

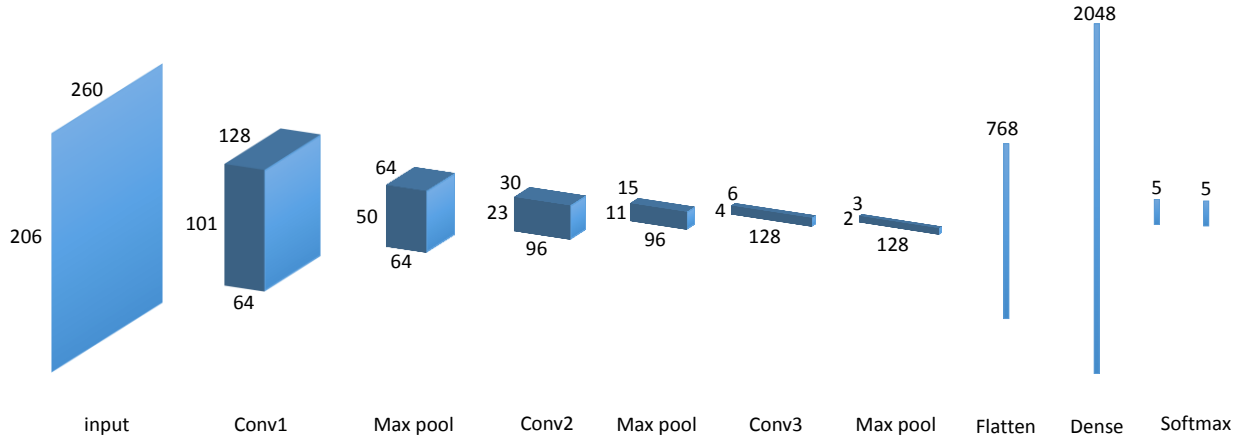

Figure 6: Final model architecture for the Ferring PoC. All conv layers are (5x5) stride (2x2) followed by batch normalization and relu activation. All max pool layers are (2x2). In total 2,048,421 trained parameters.

PERFORMANCE AND PREDICTION

Model training is focused on minimizing the categorical cross entropy (loss), but reporting accuracies and confusion matrices is more informative. Accuracy is simply the percentage of correctly predicted examples. Figure 7 shows the learning curves for the final model, where we see that the training error (loss) decreases throughout training, meaning that the model is able to learn from the training data. The validation error is a proxy for how well the obtained model is able to generalize to unseen data, and seems to be quickly plateauing, indicating that training beyond epoch 100 only causes the model to overfit to the training data. Looking at the accuracy over training "time", the story is almost the same, but on the other hand, the single best validation accuracy is obtained after 819 epochs, suggesting that with this chosen model architecture, it is worth training more than 100 epochs. According to theory, further regularization techniques (e.g. dropout) should improve generalization ability, but empirically this did not work.

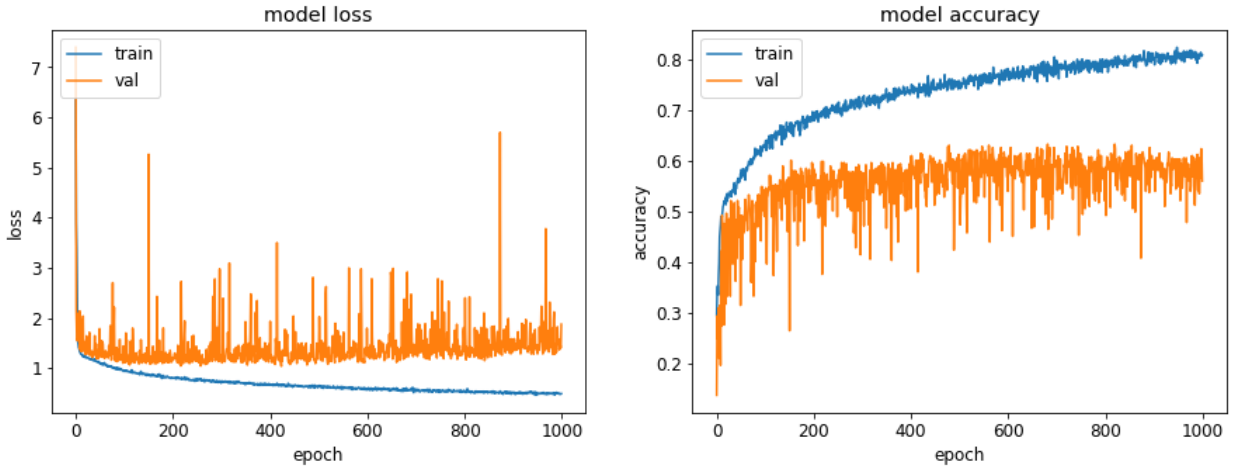

Figure 7: Learning curves for final model trained with live data augmentation (rotation, zoom, width/height shift and horizontal flip).

Testing the final model on our hold-out test dataset, yielded an out-of-sample accuracy of 55.9%, which is in the same level as the validation accuracy shown in Figure 7. The corresponding confusion matrix is shown in Figure 8, both as actual counts and percentages.

		Predicted labels				
		1	2	3	4	>=5
Actual labels	1	79	18	2	2	0
	2	8	76	17	19	0
	3	1	20	12	20	5
	4	0	18	19	67	18
	>=5	0	3	5	18	11

		Predicted (%)				
		1	2	3	4	>=5
Actuals (%)	1	0,78	0,18	0,02	0,02	0,00
	2	0,07	0,63	0,14	0,16	0,00
	3	0,02	0,34	0,21	0,34	0,09
	4	0,00	0,15	0,16	0,55	0,15
	>=5	0,00	0,08	0,14	0,49	0,30

Figure 8: Confusion matrix for final model evaluated on 438 test images showing an overall accuracy of 55.9%. Actual counts (left) and corresponding percentages (right).

Comparing the distribution of number of cells at day 2, shown in Figure 5, to the confusion matrices in Figure 8, we see that the model has a preference towards predicting the classes with the most examples, which is not surprising, but what is a bit surprising is that the biggest training class (4 cells) is not better predicted (55%).

DISCUSSION

Based on the relative small dataset, the performance of the final model is promising although still far from the level of the human expert embryologist. This indicates that it is not the technology but the access to large well-labeled databases that is the limiting factor for developing a deep learning human embryonic cell counter.

From this stage there are various ways of going forward. First of all a proper error analysis will reveal what the model is good at detecting and what the model is not detecting in the images, and the first step is to look at the confusion matrix and visualize some of the examples the model got right and what it got wrong.[2]

Figure 9 shows five selected examples from the validation dataset correctly classified by the model. These examples clearly illustrate, that it is possible to learn a representation of the images that makes it possible to correctly classify embryos by their number of cells.

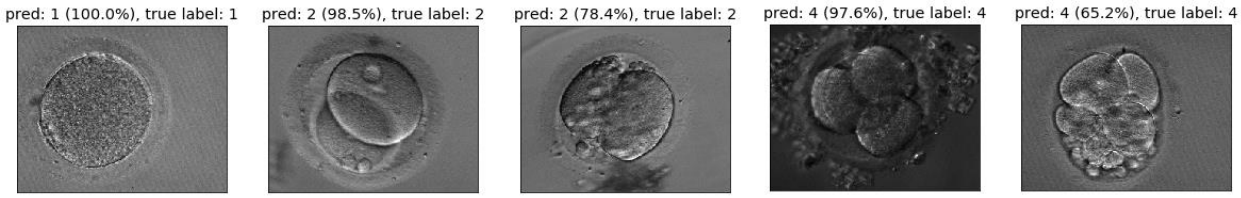

Figure 9: Five correctly predicted examples from the validation dataset.

The model is far from perfect though, and some of the wrongly predicted validation examples are illustrated in Figure 10. A human expert embryologist would probably classify all these correct, but the machine has apparently not been able to learn the patterns governing the classification of these examples, probably due to underrepresentation in the training dataset.

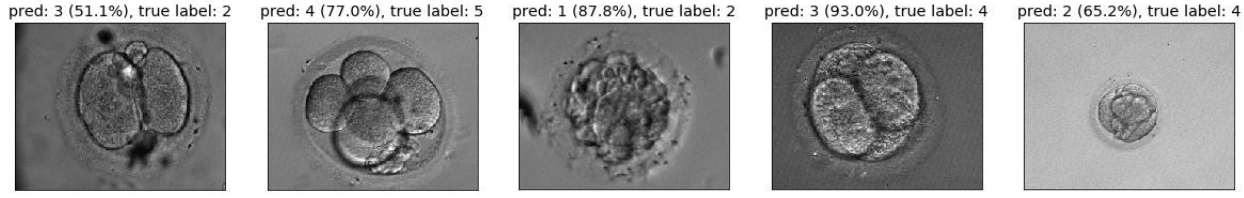

Figure 10: Five wrongly predicted examples from the validation dataset.

With more data, we should expect the machine to get the first two examples in Figure 10 correct, which from a human perspective are easy to classify. The third one has such a high degree of fragmentation that it probably is not interesting to count the number of cells, and it should suffice to classify it as *fragmented*. The fourth is a bit tricky in

[2] This should always be done before exposing the model to the hold-out test data.

the sense that the cells are completely covering the circular embryo area. This pattern is definitely underrepresented in the training data. The fifth example has a zoom which is underrepresented in the training data. Standardization of data collection or a proper pre-processing step would be the appropriate way to improve the performance for this type of errors.

A more thorough error analysis will provide more information about the strength and weaknesses of the model of this paper. An approach could be to visualize and build intuition about the active kernels for selected examples of interest, to understand why individual images are classified as they are. Another approach is to re-train the model including a so called deconvnet, which enables us to visualize what input pattern causes a given activation [8] and thereby understand what general patterns are detected by the model.

Typically more data will improve the model performance on unseen data, but acquiring more data is not always an option. Transfer learning is a popular method for compensating for lack of data and should be explored if data is limited. Yet another strategy to apply if data is limited, is to generate additional features to learn from, e.g. via image segmentation, which is either hand calibrated or machine learned.

CONCLUSION

Getting started with deep learning requires non-traditional statistical programmer skills, but training material is available and with the right mindset, flair for mathematics and programming, the road is not long. The technology is ready, and so are your competitors and third parties. It is all about identifying how your business can benefit and how to get hold of relevant data. The successful companies in this area will most likely be the companies acquiring faster access to sufficiently large and relevant data sources.

REFERENCES

[1] Domingos, P., A Few Useful Things to Know about Machine Learning, Communications of the Association for Computing Machinery, 2012, Vol. 55 No. 10, Pages 78-87

[2] Goodfellow, I., Bengio, Y. and Courville, A., Deep Learning, The MIT Press, 2016

[3] Schmidhuber, J., Deep learning in neural networks: An overview, Neural Networks, 2015, Vol. 61, Pages 85-117

[4] Krizhevsky, A., Sutskever, I. and Hinton, G. E., ImageNet Classification with Deep Convolutional Neural Networks, NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems. 2012, Vol. 1, Pages 1097-1105

[5] Khan, A., Gould, S. and Salzmann, M., Deep Convolutional Neural Networks for Human Embryonic Cell Counting, Conference Paper - European Conference on Computer Vision, 2016 Workshops, Pages 339-348

[6] Nayak, S., Number of Parameters and Tensor Sizes in a Convolutional Neural Network (CNN), https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/, May 22 2018, last visited Sep 22 2018

[7] Rahimi, A. and Recht, B., Reflections on Random Kitchen Sinks, http://www.argmin.net/2017/12/05/kitchen-sinks/, Dec 5 2017, last visited Sep 22 2018

[8] Zeiler, M. D. and Fergus, R., Visualizing and Understanding Convolutional Networks, European Conference on Computer Vision, 2014, Pages 818-833

[9] The MathWorks, Inc., Convolutional Neural Network 3 things you need to know, https://se.mathworks.com/discovery/convolutional-neural-network.html, last visited Sep 26 2018

ACKNOWLEDGMENTS

I would like to thank my manager Bjarke Mirner Klein for all the support and encouragement he provides in my search for machine learning projects within Ferring, and not least for the discussions we have related to new ideas and the work he puts into data-hunting.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Thomas Ellebæk
Ferring Pharmaceuticals A/S
Kay Fiskers Plads 11

2300 Copenhagen S
004528787983
Thomas.Ellebaek@ferring.com