Deep Learning with Small Data

Huw Evans (470333523)
Acknowledgements to Lionel Ott

November 18, 2017

1 Introduction

Deep learning takes advantage of two trends—larger and larger amounts of data and increasing computing power—to train large mathematical models that produce expected outputs from given inputs. But in many cases (for example in medical imaging or self-driving cars), examples from a class can be very rare and expensive to collect [1] or large amounts of data can be infeasible to label (real world datasets often fall in the range of 10^2–10^5 items). This raises the question—how much data do we need to achieve the required performance from a network?

This report explores the relationship between the number of data items available to train on, and various measures of performance for a network by running a number of tests with increasingly large samples of the MNIST [2] and CIFAR-10 [3] image recognition datasets.

1.1 Background

A neural network is a mathematical approximation of the way our brains process data. A typical network is made up of interconnected nodes, which each perform a mathematical operation and pass the result to other nodes. Usually they operate as **feedforward** networks, such that these connections all point in the same direction from an input to an output. Just like their biological counterparts, each 'neuron' (or node) has a number of inputs and a single output.

Typically, a neuron will perform a weighted sum of its inputs and add a bias term (input $\cdot$ weights + bias) to get an output. Like a biological neuron, that output is then passed through an **activation function** to determine how much that neuron should output. A simple example of an activation function is the Rectified Linear Unit (ReLU) [4]:

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

Which is equivalent to saying 'only output if your value is positive'.

A **deep network** is any network that has a layer of input nodes, a layer of output nodes (often corresponding to single categories), and a number of 'hidden' layers in between [5]. Such a network can be 'trained' to output correctly by minimising some **loss**

function (the L_2-norm in simple cases, or cross-entropy in others [6]), and that minimisation occurs through a range of algorithms, mostly focused on **gradient descent**. Gradient descent is like rolling a ball down a hill—look at the direction of the steepest decrease from where you are, and take a step of a fixed size that way (sometimes it also encodes the idea of momentum).

Often a standard deep network is enough to learn a simple dataset like MNIST, which is made up of black and white images of the digits 0-9. For something like that, the input layer is a 28×28 grid of pixel intensities, then the first layer detects simple edges, the second forms those edges into larger shapes, and the output layer forms shapes into digits. But for CIFAR-10, which has 10 classes of 32×32 colour images (i.e. $\times 3$ colour channels), a more complex approach is necessary.

A **convolutional network** is one that replaces the standard weighted sum with the convolution operation. To visualise this, think of a small window that can detect a feature, like an edge or a loop, regardless of where on the image it's placed. The feature-detecting mechanism is called a *kernel*, and consists of learnable parameters—far less than if we tried to detect the same edge at every point on the image separately.

The following definitions are also useful to understand this project:

- **Pooling**: Usually applied after convolution, this will output the maximum value over a small window.
- **Adadelta**: An optimisation algorithm similar to gradient descent, but it modifies the size of the step based on a decaying average of previous steps. It also has different step sizes for different variables, rather than a fixed step for the whole algorithm. [7]
- **RMSProp**: Very similar to Adadelta, but incorporates a learning rate. [8]
- **Softmax**: An activation function described by:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

  Softmax squashes the vector $\mathbf{z}$ into a vector of real values $\in [0, 1]$ that add up to 1, that approximate probabilities of the original image being in that category. This makes it easier to compute cross-entropy, which relies on probabilities to determine the loss of a categorical classifier. It's usually only used as the last layer in a network.

1.2 Related Work

The field has taken a number of different directions to reduce or otherwise optimise the amount of data required to sufficiently train a network.

Data Augmentation Involves flipping, rotating, translating and adjusting brightness, contrast & other dimensions of an image many times, then adding it to the dataset. This way, a small dataset can be made much bigger and encompass a much larger set of potential observations [9].

Transfer Learning Humans can usually repurpose the basic edge and shape detection of their eyes to new problems without having to re-learn each and every substructure. Transfer learning applies this idea to machine learning by training a large network on a very general data set and saving it, then tuning its last few layers to the problem at hand [10, 11, 12].

Data Preprocessing If a dataset is not uniformly distributed, a network can easily minimise its losses by learning the most common elements—but in the contexts of medicine or autonomous cars the edge cases are often the most important to learn. Often manually preprocessing the data to make it uniform or close to uniform is a way of improving training on a model [13].

2 Method

To explore the relationship between data set size and performance, a simple convolutional network was created that could be trained quickly and easily for experiments. For each set, the training data was shuffled so that classes would occur at reasonably uniform rates, then sliced into increasingly large subsets. For MNIST, the subset sizes were $\{1000, 2000, \ldots, 55000\}$ items, while for CIFAR-10 they were $\{1000, 2000, \ldots, 50000\}$ items (where the maximum size of each was the maximum amount of data available). The subset sizes were chosen after some initial experimentation to provide sufficient granularity while allowing the experiments to complete in a reasonable amount of time.

After some preliminary experiments, each dataset was trained on an example network from the Keras library [1], which were found to deliver results close to state of the art [14] (as well as being easy to implement).

Each network was run for 20 training epochs (i.e. the network was exposed to every image 20 times before completing training) on a GPU using TensorFlow and the TFLearn library, and at the end of each epoch a confusion matrix was generated from each validation set and saved.

The specific network structures were as follows:

[1] MNIST: `https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py`
CIFAR-10: `https://github.com/fchollet/keras/blob/master/examples/CIFAR10_cnn.py`

MNIST	CIFAR-10
	Augmentation: Flip Horizontally
	Augmentation: Shift (max 10%)
Input $28 \times 28 \times 1$	Input $32 \times 32 \times 3$
3×3 conv. 32 ReLU	3×3 conv. 32 ReLU (same pad)
3×3 conv. 64 ReLU	3×3 conv. 32 ReLU
2×2 max-pooling stride 2	2×2 max-pooling stride 2
Dropout 0.25	Dropout 0.25
Dense 128 ReLU	3×3 conv. 64 ReLU (same pad)
Dropout 0.5	3×3 conv. 64 ReLU
Dense 10 softmax	2×2 max-pooling stride 2
	Dropout 0.25
	Dense 512 ReLU
	Dropout 0.5
	Dense 10 softmax

The MNIST network was trained with an Adadelta optimiser (using the default parameters $\alpha = 1.0$, $\rho = 0.95$, $\epsilon = 10^{-8}$, decay $= 0$), while the CIFAR-10 network was trained using an RMSProp optimiser ($\alpha = 0.0001$, decay $= 10^{-6}$). The results were not expected to be affected by choice of optimiser, provided each one trained the network.

For more detail, the code for the networks is stored in the repositories `mnist-conv`[2] and `cifar-conv`[3] along with the raw result data.

3 Results & Discussion

The confusion matrices reveal some interesting trends about the data sets they underlie. In MNIST, the confusion matrices fluctuate a little but clearly and quickly converge on the correct data. To better visualise what the classifiers are getting wrong, all of the confusion matrices were modified by subtracting the correct entries (which lay along the diagonal).

The first insight from the confusion matrices is that they have a distinct structure over the dataset, visualised in figures 1 and 2 (which sum the errors over all iterations to get a bigger picture). In MNIST, a lot of 7s have been learned as 2s, and 4s and 9s—which makes sense, given their similar shapes. CIFAR-10 is even more interesting, since it has a clear 'island' of animals being misclassified as each other (the most prominent of which being cats classified as dogs and vice-versa), surrounded by man-made vehicles being misclassified as each other. However, these two areas of error have very little overlap (with the amusing exception of birds being classified as airplanes), demonstrating that the CIFAR network was quickly able to distinguish between man-made and organic objects (and struggling more to make distinctions within those groups).

While these insights are interesting, the purpose of this analysis is to see how quickly these distinctions form on small amounts of data. Figure 3 shows that MNIST has almost exactly the same structure after only seeing 1000 handwritten digits (although

[2] `https://github.sydney.edu.au/heva9329/mnist-conv`

[3] `https://github.sydney.edu.au/heva9329/cifar-conv`

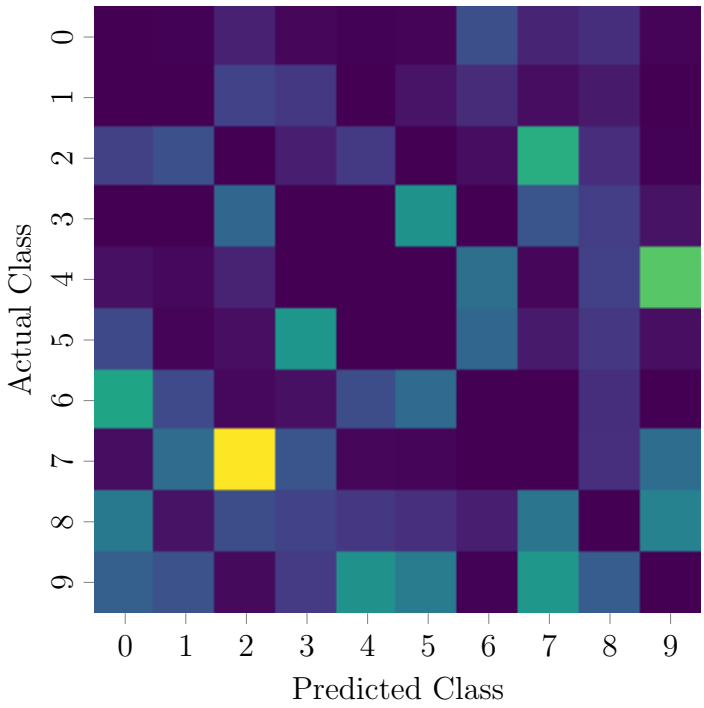

Figure 1: MNIST Confusion Matrix (summed)

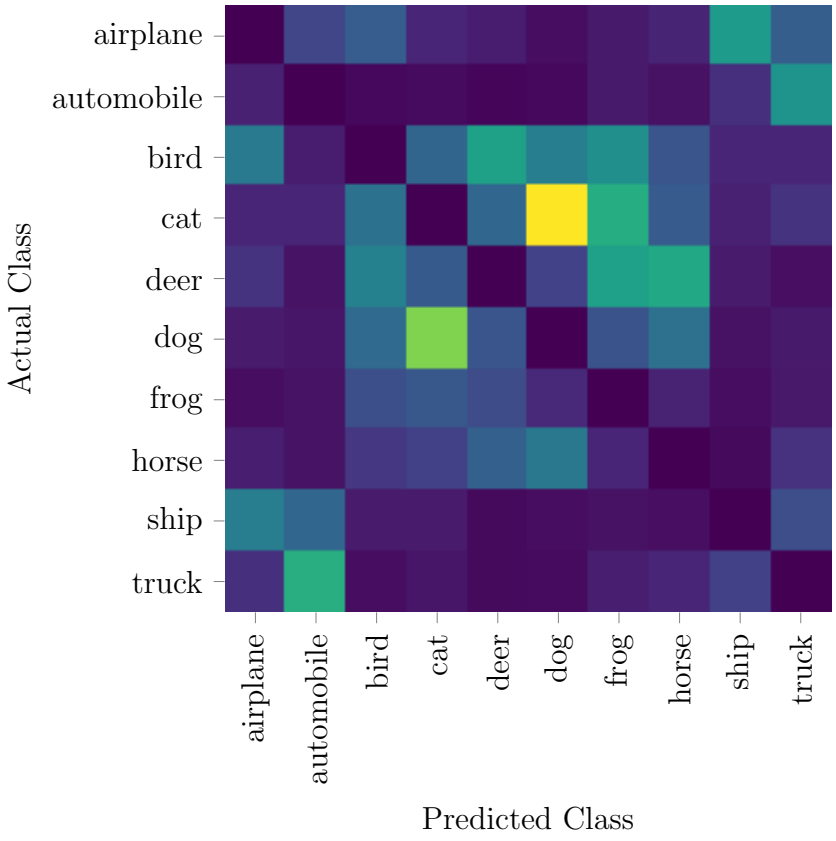

Figure 2: CIFAR-10 Confusion Matrix (summed)

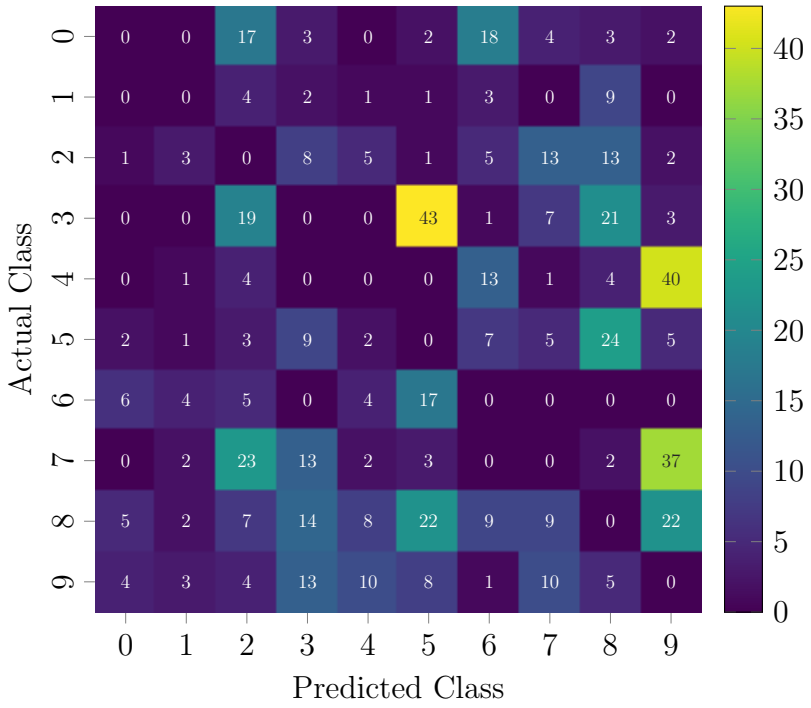

Figure 3: MNIST Confusion Matrix (1000 samples)

MNIST Confusion Matrix (4000 samples)

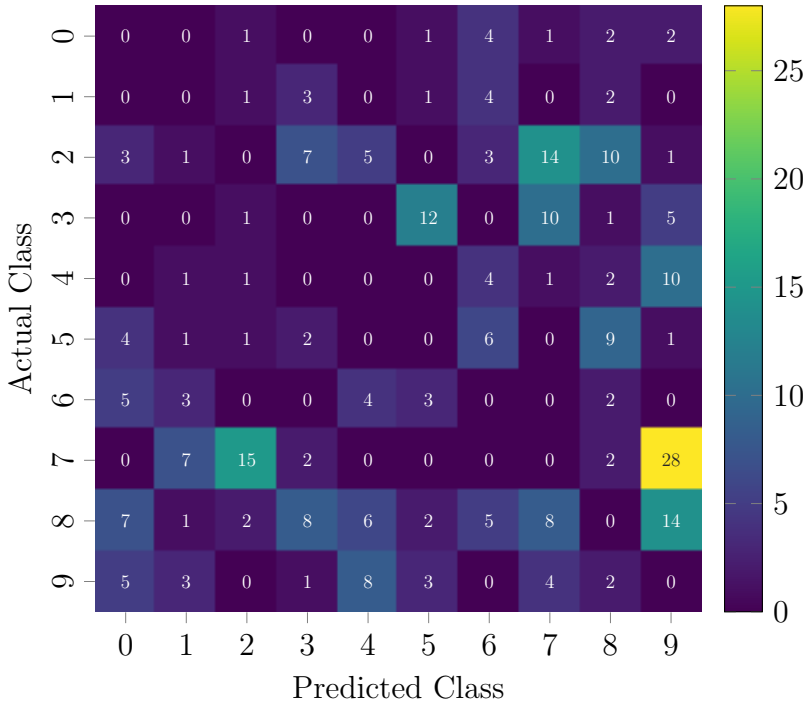

Figure 4: MNIST Confusion Matrix (4000 samples)

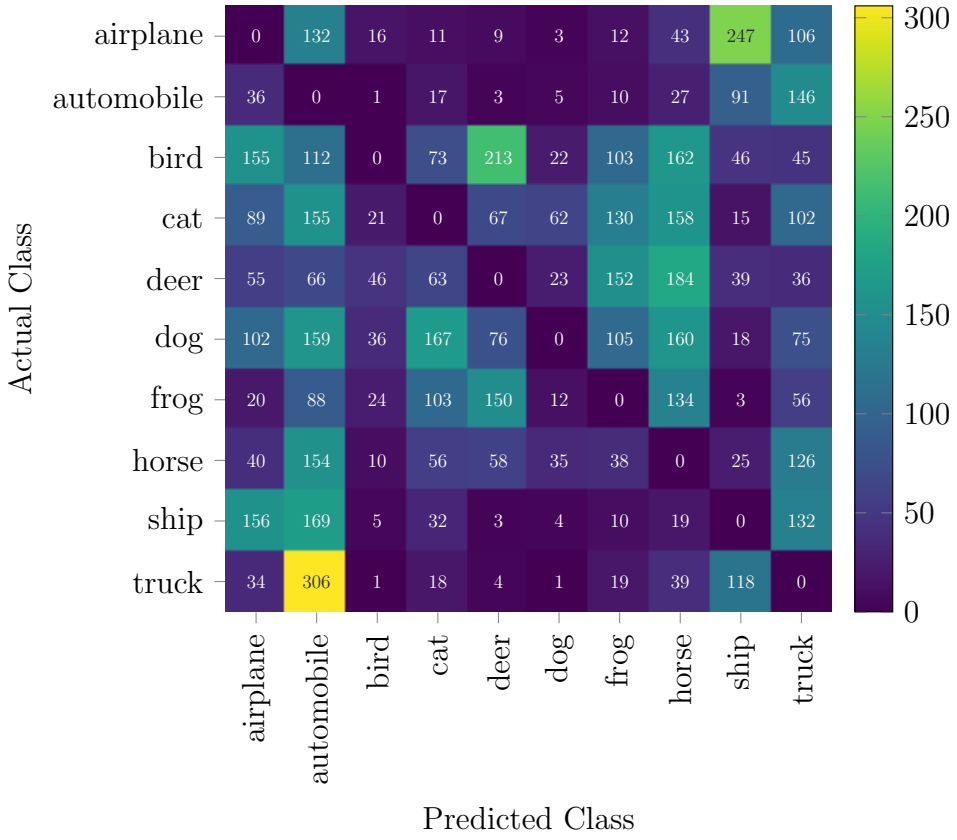

Figure 5: CIFAR-10 Confusion Matrix (1000 samples)

with different peaks), and by 4000 iterations (figure 4) it has the same misclassification problems. This indicates that after the first few thousand images, the hard work in MNIST is from learning the difficult examples. Essentially, a lot of MNIST is generic examples which are useless to the network—past a threshold of generic data points, the network likely only needs to be exposed to outliers to improve on its metrics, and hence we see a pattern of diminishing returns in the accuracy (7).

Figure 5 shows that the main structure has started to form, but is predicting far too many things as automobiles, trucks, and airplanes. This makes sense; 1000 points are far less useful in a dataset with more colour channels and a wider range of things to learn. However, by 4000 data points (figure 6), the network has approached its main structure. While 4000 is not a magic number, it is quite low—less than 10% of the data was necessary to train the network on typical examples, and the rest seems only useful for providing the atypical examples it needs to distinguish cats from dogs, or birds from planes.

Looking at the accuracy of both networks over time, it's immediately clear that the MNIST network performs far better than CIFAR-10's network (in the range of 96-99% rather than 40-70%), which is a consequence of the relative ease of each problem (and the relative work to incrementally push the state of the art on MNIST over CIFAR-10). What also stands out, however, is the shape of the curves—both confirm that a diminishing returns effect exists on each network, again suggesting that showing the network typical examples only does so much to improve performance after a certain point. MNIST's

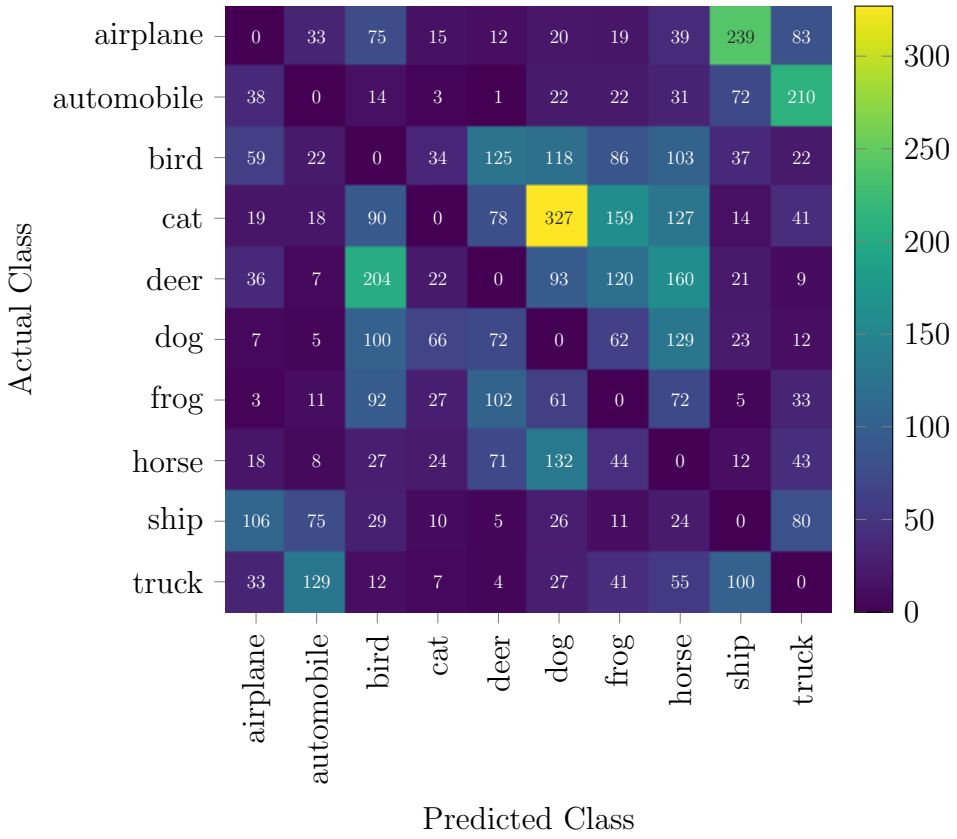

Figure 6: CIFAR-10 Confusion Matrix (4000 samples)

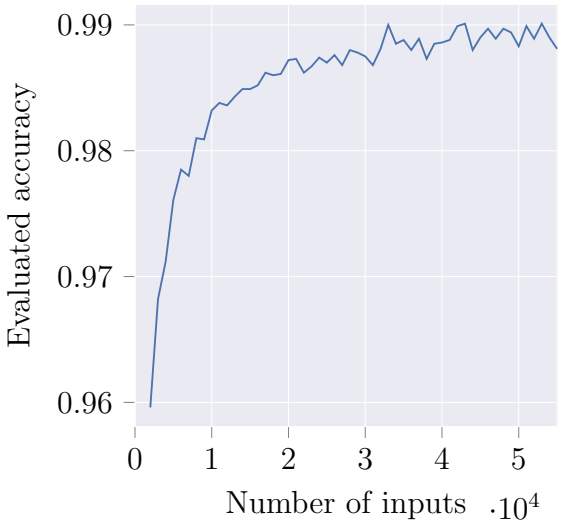

Figure 7: MNIST accuracy

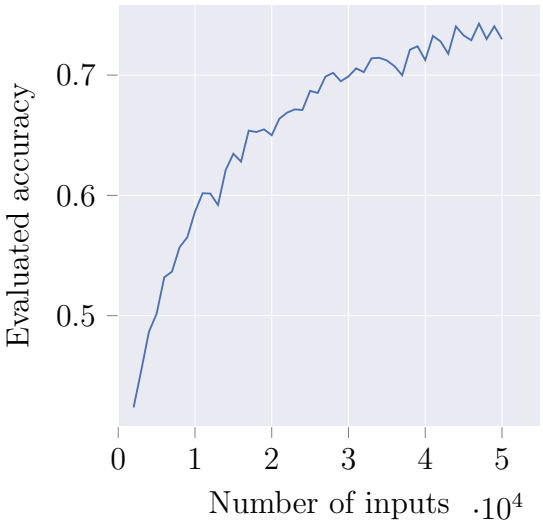

Figure 8: CIFAR-10 accuracy

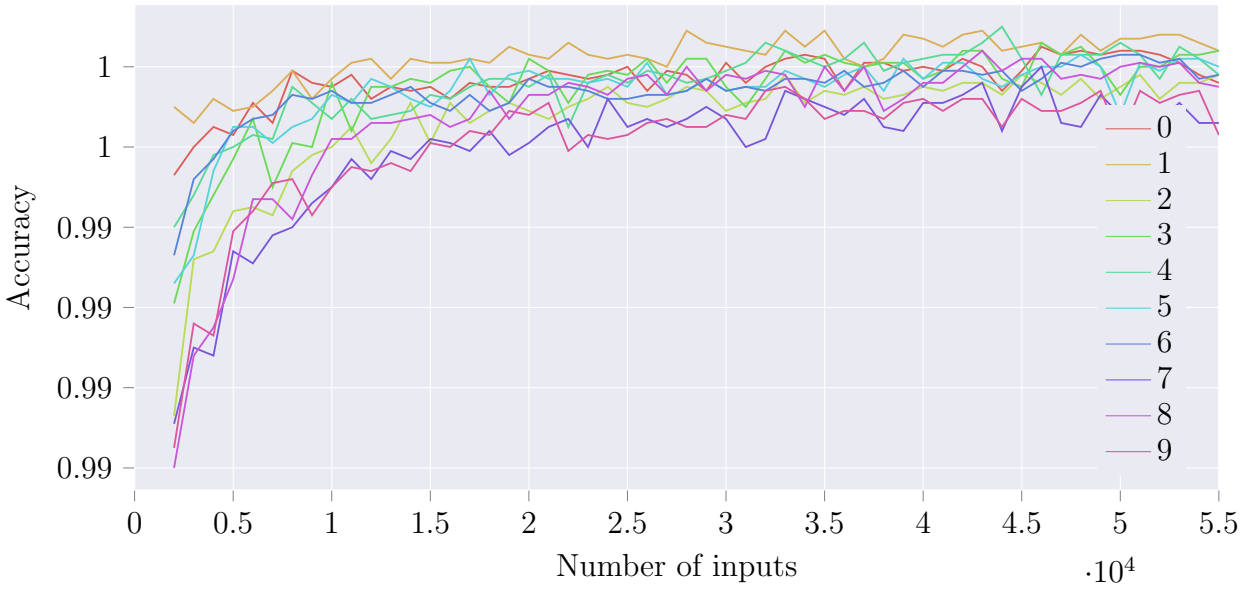

Figure 9: MNIST accuracy (by class)

graph appears to hit an upper bound at 99%, which suggests that any more data is likely to provide almost zero improvement. However, CIFAR-10's graph appears to still have some way to go before reaching an upper bound, and on comparing the two the curve is similar to the first half of MNIST's curve. More data is needed before this particular network will reach peak performance—or, perhaps, simply more training epochs (the Keras examples where the network is sourced from claim the network peaks at around 50 epochs).

Finally, the class-by-class plots of accuracy (figures 9 and 10) and f1-scores (figures 11 and 12) confirm the picture painted by the rest of the data—the harder to learn classes need more data to catch up to the performance of the easier ones, and the curvatures verify that MNIST plateaus very quickly, while CIFAR-10 could still use more data or training time.

4 Conclusion

Modern datasets are getting larger and larger, mostly because companies with lots of resources are willing to collect more and more data for training purposes (e.g. the Street View House Numbers set [15])—but perhaps they don't need to be. This report shows that a lot of the later incremental learning in deep networks is based on encountering novel, atypical examples, and that typical examples are learned using relatively few data points.

There are two useful ideas that come from this. Firstly, when collecting new datasets, it's useful to have a uniform (rather than normal) spread of typical and atypical examples. But this raises a significant challenge: How should a collector determine ahead of time which examples to collect? It could be enough for them to reply on their own instincts about what is typical or atypical—but this is often too variable between people (par-

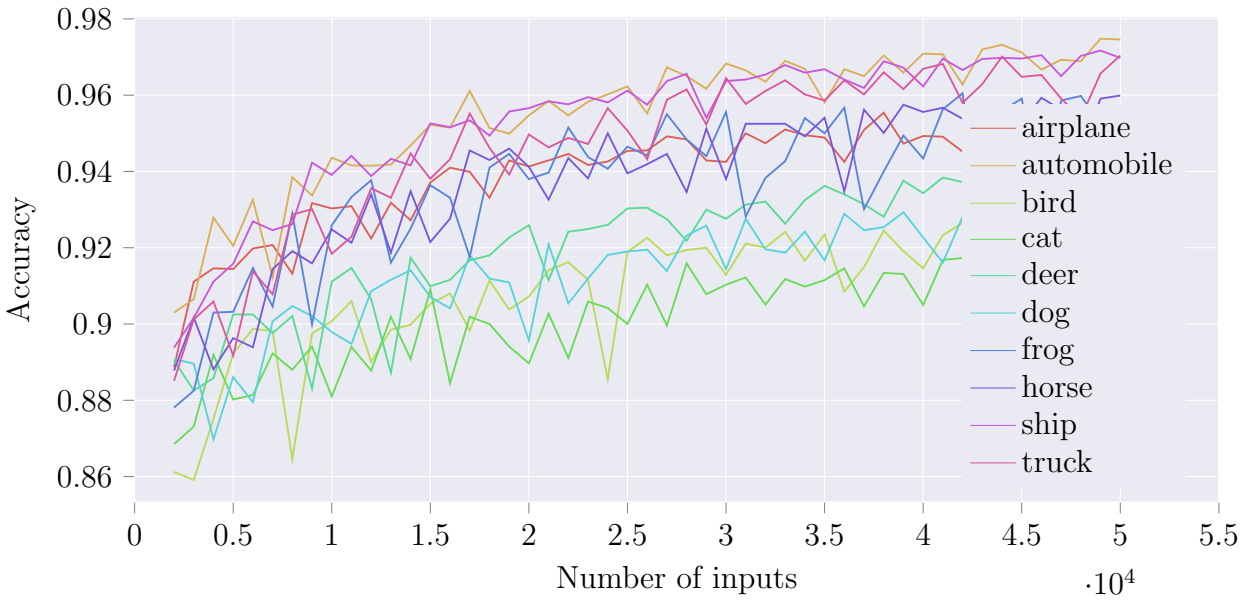

Figure 10: CIFAR-10 accuracy (by class)

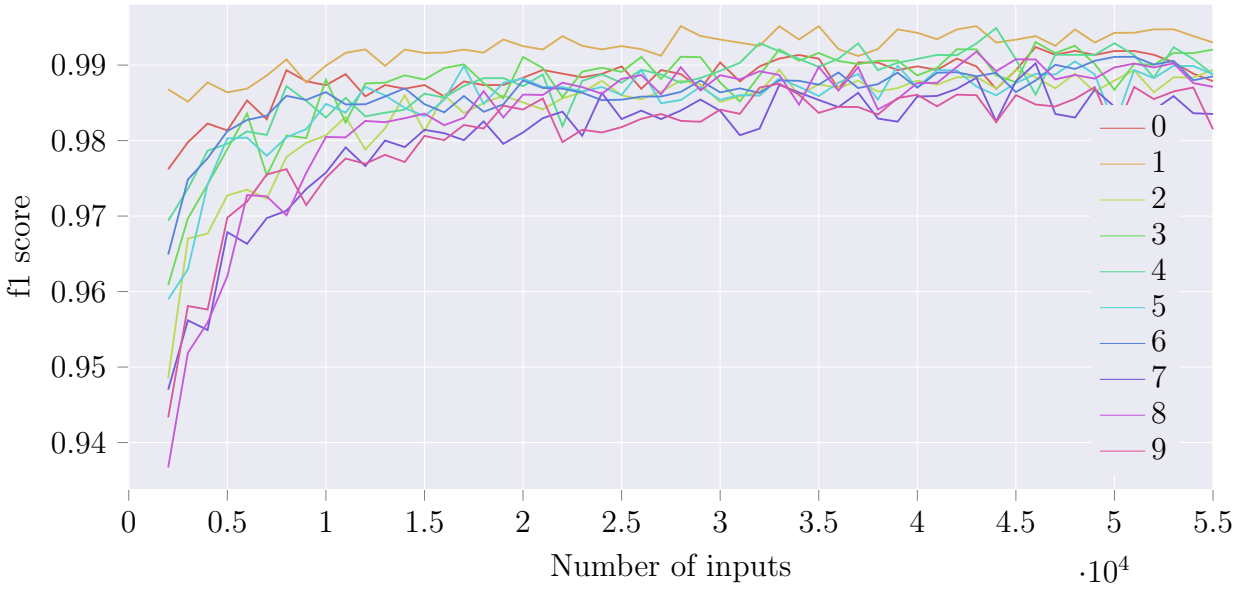

Figure 11: MNIST f1-scores

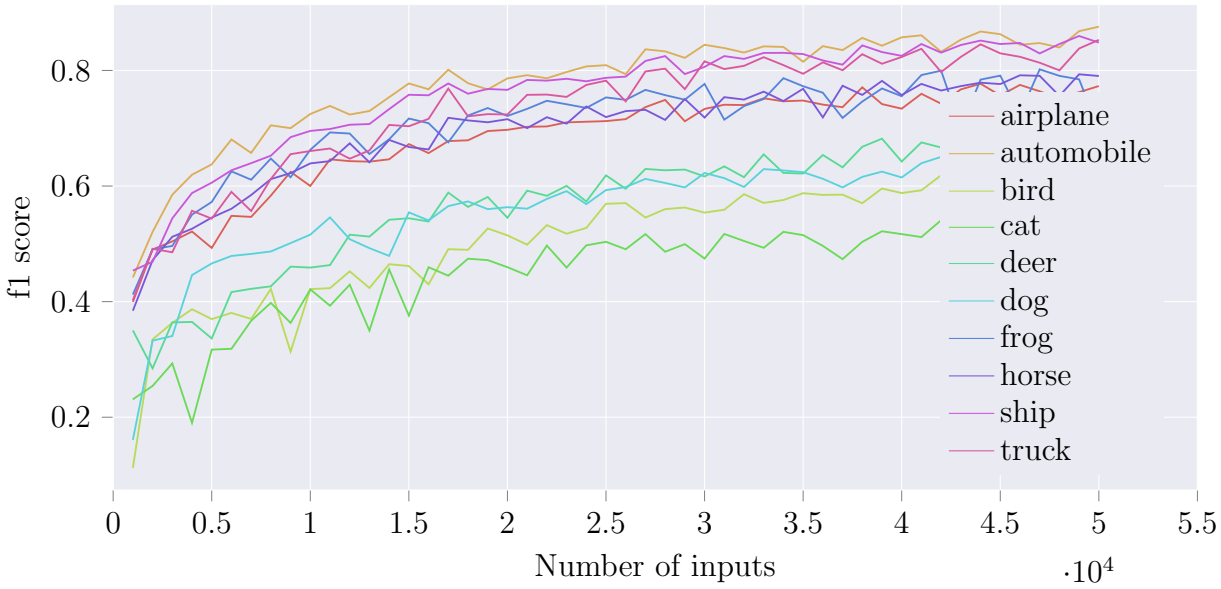

Figure 12: CIFAR-10 f1-scores

ticularly in medical professions where a doctor will have extensive experience with such atypical examples). However, if the data labels are crowdsourced through Mechanical Turk or CAPTCHAs, examples that are labelled differently by different people could be flagged for inclusion in the final set over examples which people consistently get correct.

The second useful idea is that repeated training epochs likely don't benefit from showing the model examples it has repeatedly gotten right. Training time for deep networks could be dramatically increased by designing an algorithm that trains more often on difficult examples—although such an algorithm would also be in danger of overfitting.

Finally, a network that only needs reasonable performance (e.g. by acting as a second opinion or backup to a human) does not need nearly as many example as are in some datasets. Ahead of time, it isn't clear exactly what numbers are useful—since this will very much depend on the data being collected—but by comparing the data being collected with existing sets a reasonable number could be estimated. If that number is too low, then simply collect more data until done.

References

[1] N. Kajbakhsh et al. "Convolutional neural networks for medical image analysis: Full training or fine tuning?" In: *IEEE Transactions on Medical Imaging* 35.5 (2016), pp. 1299–1312.

[2] Yann LeCun and Corinna Cortes. *The MNIST database of handwritten digits*. URL: `http://yann.lecun.com/exdb/mnist/`.

[3] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". MA thesis. 2009. URL: `http://www.cs.toronto.edu/%5C~%7B%7Dkriz/learning-features-2009-TR.pdf`.

[4] Richard Hahnloser et al. “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit”. In: 405 (July 2000), pp. 947–51.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521 (May 2015), pp. 436–444. URL: http://dx.doi.org/10.1038/nature14539.

[6] Chris Olah. *Visual Information Theory*. Oct. 2015. URL: http://colah.github.io/posts/2015-09-Visual-Information/.

[7] M. D. Zeiler. “ADADELTA: An Adaptive Learning Rate Method”. In: *ArXiv e-prints* (Dec. 2012). arXiv: 1212.5701 [cs.LG].

[8] T. Tieleman and G. Hinton. *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning. 2012.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016, pp. 240–241.

[10] G. Hinton, O. Vinyals, and J. Dean. “Distilling the Knowledge in a Neural Network”. In: *ArXiv e-prints* (Mar. 2015). arXiv: 1503.02531 [stat.ML].

[11] Yoshua Bengio. “Deep Learning of Representations for Unsupervised and Transfer Learning”. In: *ICML Unsupervised and Transfer Learning* 27 (2012), pp. 17–36.

[12] T. Chen, I. Goodfellow, and J. Shlens. “Net2Net: Accelerating Learning via Knowledge Transfer”. In: *ArXiv e-prints* (Nov. 2015). arXiv: 1511.05641 [cs.LG].

[13] Mez Gebre. *MainSqueeze: The 52 parameter model that drives in the Udacity simulator*. Feb. 2017. URL: https://mez.github.io/deep%20learning/2017/02/14/mainsqueeze-the-52-parameter-model-that-drives-in-the-udacity-simulator/.

[14] Rodrigo Benenson. *Who is the best at X?* Feb. 2016. URL: http://rodrigob.github.io/are_we_there_yet/build/.

[15] Yuval Netzer et al. “Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning”. In: (2011).