Tackling Class Imbalance with Deep Convolutional Neural Networks
— Final —

Alexandre Dalyac, Prof Murray Shanahan, Jack Kelly; Imperial College London

September 24, 2014

Abstract

Automatic image classification experienced a breakthrough in 2012 with the advent of GPU implementations of deep convolutional neural networks (CNNs). These models distinguish themselves by learning features, convolving them to achieve feature equivariance, making use of max pooling operations to achieve robustness to cluttered backgrounds, and a hierarchical structure to enable the learning of complex features with high generalisation performance.

This report provides an overview of supervised deep learning, followed by an in-depth explanation of the workings of a deep convolutional neural network, which despite its performance, gathers criticism for its tendency to be used as a black box. The rest of the report goes over the research that was carried out. The first is a simple mathematical analysis which proposes an explanation for the recent success of the Rectified Linear Unit as the chief activation function of CNNs. The next sections are applied: understanding why learning fails with class imbalance and noisy labels, and exploring ways of tackling it – including a novel cost function with a probabilistic interpretation which delivered up to 25 additional percentage points in classification performance.

Contents

1 Acknowledgements **4**

2 Introduction **5**
2.0.1 Data Overview . . . 5
2.0.2 Semantic Complexity . . . 5
2.0.3 Domain Change . . . 6
2.0.4 Small Dataset Size . . . 6
2.0.5 Class Imbalance . . . 7

3 Literature Review **8**
3.1 Supervised Learning . . . 8
3.2 Approximation vs Generalisation . . . 8
3.3 Models of Neurons . . . 8
3.4 Feed-forward Architecture . . . 11
3.5 Justifying Depth . . . 12
3.6 Backpropagation . . . 14
3.6.1 Compute Error-Weight Partial Derivatives . . . 14
3.6.2 Update Weight Values with Gradient Descent . . . 14
3.6.3 Stochastic Gradient Descent . . . 14
3.7 Overfit . . . 14
3.7.1 Cross Validation . . . 15
3.7.2 Data Augmentation . . . 16
3.7.3 Dropout . . . 16
3.8 Deep Convolutional Neural Networks . . . 18
3.8.1 Pixel Feature . . . 19
3.8.2 Non-linear Activation . . . 21
3.8.3 Pooling aka Spatial Feature . . . 21
3.8.4 Contrast Normalisation . . . 21
3.9 Local vs Global Optimisation . . . 22
3.10 Transfer Learning . . . 23
3.10.1 Linear Support Vector Machines . . . 23
3.11 Class Imbalance . . . 23

4 Analysis 1: ReLU Activation **25**
4.1 Motivations . . . 25
4.2 Mathematical Analysis . . . 26
4.2.1 How the Gradient Propagates . . . 26
4.2.2 An Example . . . 26
4.2.3 Vanishing Gradient . . . 27
4.2.4 Impact of the ReLU . . . 28

5 Experiments 1: Identifying Class Imbalance **30**
5.1 Implementation: Cuda-Convnet . . . 30
5.2 Experimentation . . . 30
5.2.1 Non-Converging Error Rates . . . 30
5.2.2 Increase Validation Error Precision . . . 31
5.2.3 Periodicity of the training error . . . 32
5.2.4 Poor, Sampling-Induced Corner Minima . . . 32
5.2.5 Mislabelling . . . 34

6 Experiments 2: Tackling Class Imbalance **36**
6.1 Definition . . . 36
6.2 Motivations . . . 36
6.3 Implementation . . . 37
6.4 Experimentation . . . 37
6.4.1 Test Run . . . 37
6.4.2 Transfer Learning . . . 39
6.4.3 Hyperparameter Optimisation . . . 39
6.4.4 Over-Sampling . . . 41
6.4.5 Under-Sampling . . . 41
6.4.6 Threshold-Moving . . . 44
6.4.7 Bayesian Cross Entropy Cost Function . . . 45
6.4.8 Error-accuracy mismatch revisited . . . 49

7 Conclusions and Future Work **50**

1 Acknowledgements

Many thanks to Prof Murray Shanahan for having been the chief supervisor of this research project; it would not have been possible to pursue it otherwise.

Thanks to Jack Kelly, the second supervisor of this project, for having created it. Were it not for him I would not have been able to carry out exciting, meaningful research for a real world problem.

Thanks to Razvan Ranca for precious strategic advice on which experiments to choose for the exploration of hypotheses, help with C++, feedback on the report, as well as musical and caffeine tricks for staying up all night. I look excitingly forward to the times ahead cofounding a machine learning startup together.

Thanks to Alexandre de Brebisson for his great insights into convnet architecture that helped me make sense of experimental results, and the great conversations we had sharing our passion for deep learning.

The Google+ Deep Learning community was perhaps the 2nd most precious resource for this project, after arXiv and before stackoverflow. It provided a large and steady stream of useful resources, comments and guidance. It also seemed like a revolutionary way to be in touch with the world's leading researchers such as Prof Yann LeCun and Prof Yoshua Bengio. I owe particular thanks to Dr Soumith Chintala for his comments on transfer learning and Prof Francisco Zamora-Martinez for his on class imbalance.

2 Introduction

A number of significant challenges arise from this task: multi-tagging, semantic complexity, domain change, small dataset size and class imbalance. Before going into them, an overview of the data is given on figure 1.

2.0.1 Data Overview

ControlPoint recently upgraded the photographical equipment with which photos are taken (from 'Redbox' equipment to 'Bluebox' equipment), which means that the resolution and finishing of the photos has been altered. There are 113,865 640x480 'RedBox' images. There are 13,790 1280x960 'BlueBox' images. Label frequencies for the Redbox images are shown on figure 1.

Characteristic	Redbox Count	Bluebox Count
Fitting Proximity	1,233	32
Inadequate Or Incorrect Clamping	1,401	83
Joint Misaligned	391	35
No Clamp Used	8,041	1,571
No Ground Sheet	30,015	5,541
No Insertion Depth Markings	17,667	897
No Visible Evidence Of Scraping Or Peeling	25,499	1,410
No Visible Hatch Markings	28,155	3,793
Other	251	103
Photo Does Not Show Enough Of Clamps	5,059	363
Photo Does Not Show Enough Of Scrape Zones	21,272	2,545
Soil Contamination High Risk	6,541	3
Soil Contamination Low Risk	10	N/A
Soil Contamination Risk	?	529
Unsuitable Photo	2	N/A
Unsuitable Scraping Or Peeling	2,125	292
Water Contamination High Risk	1,927	9
Water Contamination Low Risk	3	7
Water Contamination Risk	?	296
Perfect (no labels)	49,039	4,182

Table 1: Count of Redbox images with given label

2.0.2 Semantic Complexity

Certain visual characteristics are semantically more complex than normal object classes, because ControlPoint has rules for what counts to raise a flag depending on the nature of the joint. For example, in the case of clamp detection, for tapping-T joints, for the Redbox images, the glint of a slim portion of a clamp is sufficient to judge it present. An example is shown on figure 1.

Worse still, sometimes the presence of clamps 'does not count': these are cases for which the purpose of the clamp is other than to secure the welding. Therefore, if such a clamp is present, but the clamp that serves to secure the weld is absent, then the image is assigned the 'No Clamps' label. For example, bottom left, a clamp can clearly be seen, but it's not a weld clamp. So this image should have a 'No Clamps' flag raised (sadly, it doesn't). Bottom right: the thin metallic clamp that is fastened on the vertical pipe is not the clamp we're interested in. The glint from the thin metallic rod going along the thick, horizontal pipe tells us that a tapping-T clamp is present, even though that clamp is hidden underneath the pipe. An example is shown on figure **??**.

Figure 1: The clamp wraps around under the pipe - the glint of a metal rod gives it away

(a) The clamp is not a weld clamp

(b) The clamp on the vertical rod is not a weld clamp

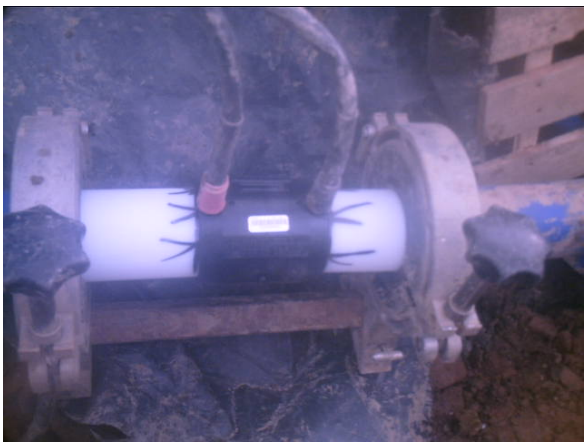
2.0.3 Domain Change

Domain change can be lethal to computer vision algorithms: for example, a feature learned (at the pixel level) from the 640x480 Redbox images could end up being out of scale for the 1280x960 Bluebox images. However, this simple example is not relevant to a CNN implementation, since the largest networks can only manage 256x256 images, so Bluebox and Redbox images will both be downsized to identical resolutions. However, of greater concern is the difference in image sharpness between Redbox and Bluebox images, as can be seen on figure **??**. It remains to be seen how a CNN could be made to deal with this type of domain change.

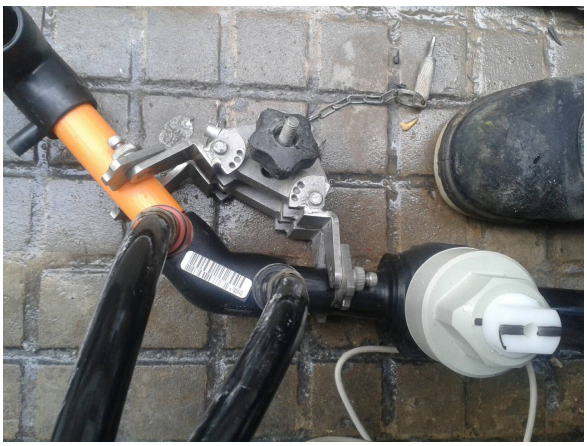
Nevertheless, evidence has been found to suggest that deep neural networks are robust to it: an experiment run by Donahue et al on the *Office* dataset [22], consisting of images of the same products taken with three different types of photographical equipment (professional studio equipment, digital SLR, webcam) found that their implementation of a deep convolutional neural network produced similar feature representations of two images of the same object even when the two images were taken with different equipment, but that this was not the case when using SURF, the currently best performing set of hand-coded features on the *Office* dataset [24].

2.0.4 Small Dataset Size

Alex Krizhevsky's record-breaking CNN was trained on 1 million images [1]. Such a large dataset enabled the training of a 60-million parameter neural network, without leading to overfit. In this case, there are 'only' 5,000, and 43% of them are images of 'perfect' welds, meaning that these are label-less. Training a similarly sized network leads to overfit, but training a smaller network could prevent the network from learning sufficiently abstract and complex features for the task at hand. A solution to consider is that of transfer learning [25], which consists in importing a net which has been pretrained in a similar task with vast amounts of data, and to use it as a feature extractor. This would bring the



(a) A Redbox photo



(b) A Bluebox photo

major advantage that a large network architecture can be used, but the number of free parameters can be reduced to fit the size of the training set by 'freezing' backpropagation on the lower layers of the network. Intuitively, it would make sense to freeze the lower (convolutional) layers and to re-train the higher ones, since low-level features (such as edges and corners) are likely to be similar across any object recognition task, but the way in which these features are combined are specific to the objects to detect.

2.0.5 Class Imbalance

The dataset suffers from a similar adverse characteristic to that of medical datasets: pathological observations are significantly less frequent that healthy observations. This can make mini-batch training of the network especially difficult. Consider the simple case of training a neural network to learn the following labels: No Clamp Used, Photo Does Not Show Enough Of Clamps, Clamp Detected (this label is not in the list, but can be constructed as the default label). Only 8% of the Redbox images contain the first label, and only 5% contain the second label, so if the partial derivatives of the error are computed over a batch of 128 images (as is the case with the best implementations [1], [25], [9]), one can only expect a handful of them to contain either of the first two labels. Intuively, one may ask: how could I learn to recognise something if I'm hardly ever shown it?

3 Literature Review

3.1 Supervised Learning

Learning in the case of classification consists in using the dataset $\mathcal{D}$ to find the hypothesis function f^h that best approximates the unknown function $f^* : 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$ which would perfectly classify any subset of the instance space $\mathcal{X}$. Supervised learning arises when $f^*(x)$ is known for every instance in the dataset, i.e. when the dataset is labelled and of the form $\{(x_1, f^*(x_1)), (x_2, f^*(x_2)), ..., (x_n, f^*(x_n))\}$. This means that $|\mathcal{D}|$ points of f^* are known, and can be used to fit f^h to them, using an appropriate cost function $\mathcal{C}$. $\mathcal{D}$ is therefore referred to as the *training set*.

Formally, supervised learning therefore consists in finding

$$f^h = \underset{\mathcal{F}}{\operatorname{argmin}}\, \mathcal{C}(\mathcal{D}) \tag{1}$$

where $\mathcal{F}$ is the chosen target function space in which to search for f^h .

3.2 Approximation vs Generalisation

It is important to note that supervised learning does not consist in merely finding the function which best fits the training set - the availability of numerous universal approximating function classes (such as the set of all finite order polynomials) would make this a relatively simple task [16]. The crux of supervised learning is to find a hypothesis function which fits the training set well *and* would fit well to any subset of the instance space, including unseen data. Approximation and generalisation together make up the two optimisation criteria for supervised learning.

3.3 Models of Neurons

Learning a hypothesis function f^h comes down to searching a target function space for the function which minimises the cost function. A function space is defined by a parametrised function equation, and a parameter space. Choosing a deep convolutional neural network with rectified linear neurons sets the parametrised function equation. By explaining the architecture of such a neural network, this subsection justifies the chosen function equation. As for the parameter space, it is $\mathbb{R}^{\mathbb{P}}$ (where P is the number of parameters in the network); its continuity must be noted as this enables the use of gradient descent as the optimisation algorithm (as is discussed later).

Before we consider the neural network architecture as a whole, let us start with the building block of a neural network: the neuron (mathematically referred to as the *activation function*). Two types of neuron models are used in current state-of-the-art implementations of deep convolutional neural networks: the rectified linear unit and the softmax unit (note that the terms 'neuron' and 'unit' are used interchangeably). In order to bring out their specific characteristics, we shall first consider two other compatible neuron models: the binary threshold neuron, which is the most intuitive, and the hyperbolic tangent neuron, which is the most analytically appealing. It may also help to know what is being modelled, so a very brief look at a biological neuron shall first be given.

Multipolar Biological Neuron A multipolar neuron receives electric charges from neighbouring incoming neurons through its dendritic branches, and sends electric charges to its neighbouring outgoing neurons through its axon. Neurons connect at synapses, which is where the tip of the telodendria of one neuron is in close vicinity of the dendritic branch of another neuron. Because a single axon feeds into all of the telodendria but mutiple dendritic branches feed into the axon hillock, a neuron receives multiple inputs and sends out a single output. Similarly, all of the neuron models below are functions from a multidimensional space to a unidimensional one.

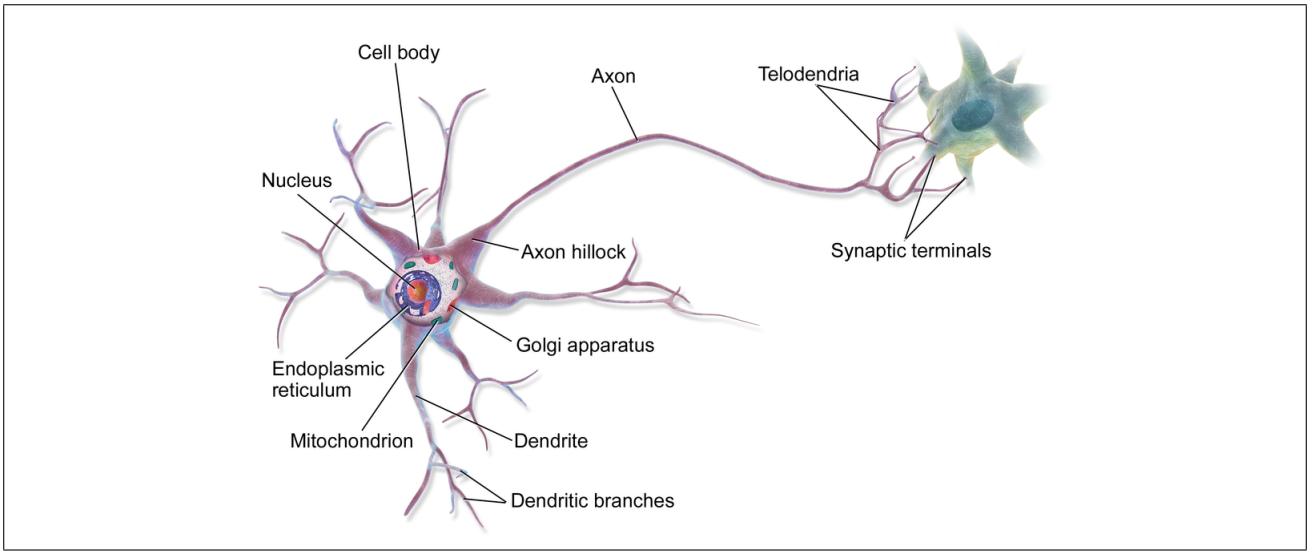


Figure 4: a multipolar biological neuron

Binary Threshold Neuron

$$y = \begin{cases} 1 & \text{if } M <= b + \sum_{i=1}^{k} x_i \cdot w_i \text{ , where } M \text{ is a threshold parameter} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Intuitively, y takes a hard decision, just like biological neurons: either a charge is sent, or it isn't. y can be seen as producing spikes, x_i as the indicator value of some feature, and $w_[i]$ as a parameter of the function that indicates how important x_i is in determining y. Although this model is closer than most most to reality, the function is not differentiable, which makes it impossible to use greedy local optimisation learning algorithms - such as gradient descent - which need to compute derivatives involving the activation functions.

Logistic Sigmoid Neuron

$$y = \frac{1}{1 + \exp(-z)}, \text{ where } z = \sum_{i=1}^{k} x_i \cdot w_i \tag{3}$$

Like the binary threshold neuron, the output domain of this neuron is bounded by 0 and 1. But this time, the function is fully differentiable. Moreover, it is nonlinear, which helps to increase performance [17]. To see why, the graph plot below lends itself to the following intuition: if the input x is the amount of evidence for the components of the feature that the neuron detects, and y is the evidence for the feature itself, then the marginal evidence for the feature is decreasing with the amount of evidence for its components (in absolute value terms).

This is like saying that to completely convince y of the total presence or absence of the feature, a lot of evidence is required. However, if there is not much evidence for either case, then y is more lenient. A disadvantage of this neuron model is that it is computationally expensive to compute.

Rectified Linear Neuron

$$y = \max\{0, b + \sum_{i=1}^{k} x_i \cdot w_i\} \tag{4}$$

As can be seen in the graph plot on figure 6, the rectified linear neuron is neither fully differentiable (not at 0), nor bounded above. Moreover, it only has two slopes, so its derivative with respect to x_i can only be one of two values: 0 or w_i. Although this may come as a strong downgrade in

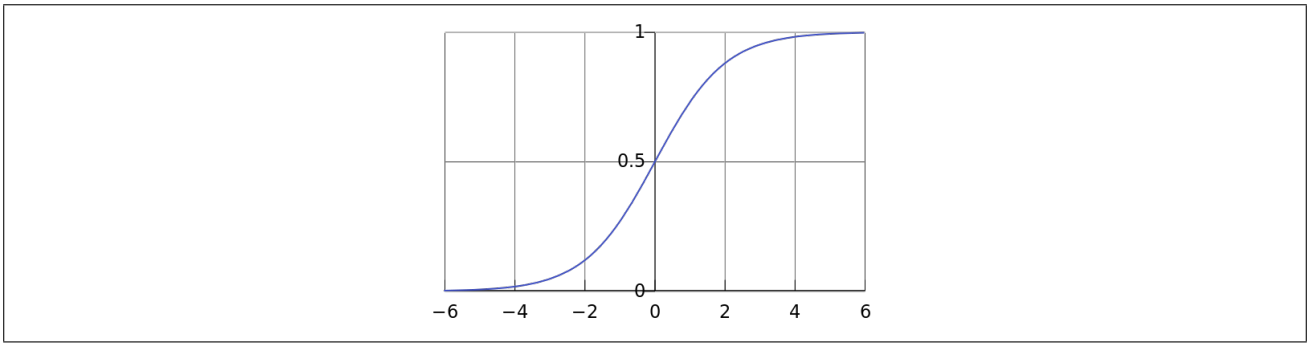


Figure 5: single-input logistic sigmoid neuron

sophistication compared to the logistic sigmoid neuron, it is so much more efficient to compute (both its value and its partial derivatives) that it enables much larger network implementations [1]. Until now, this has more than offset the per-neuron information loss - and saturation risks - of the rectifier versus the sigmoid unit [2].

ReLU introduces a non-linearity with its angular point (a smooth approximation to it is the softplus $f(x) = \log(1 + e^x)$).

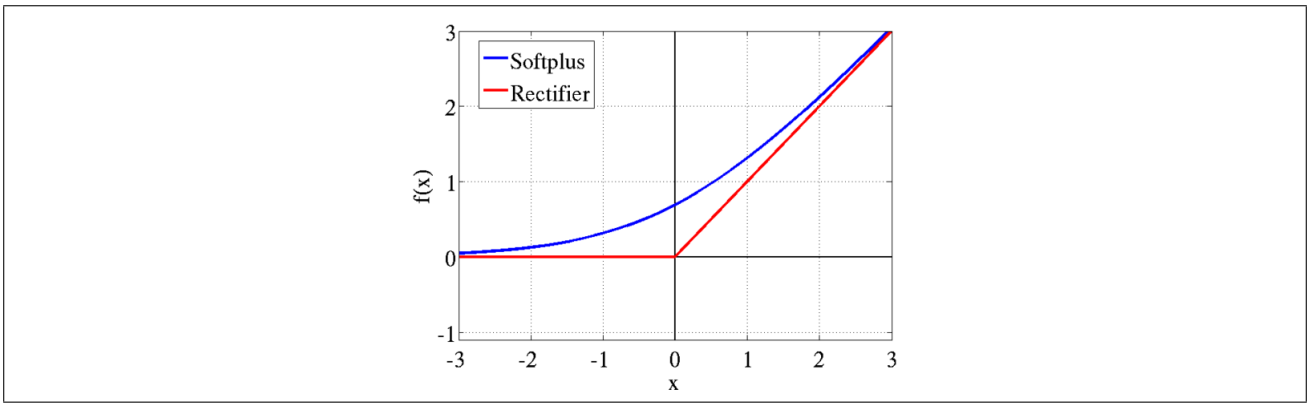


Figure 6: single-input rectified linear neuron

Softmax Neuron

$$y_j = \frac{\exp(z_j)}{\sum_{i=1}^{k} \exp(z_i)}, \text{ where } z_j = \sum_{i=1}^{k} x_i \cdot w_{i,j} + b \tag{5}$$

The equation of a softmax neuron needs to be understood in the context of a layer of k such neurons within a neural network: therefore, the notation y_j corresponds to the output of the j^{th} softmax neuron, and $w_{i,j}$ corresponds to the weight of x_i as in input for the j^{th} softmax neuron. A layer of softmax neurons distinguishes itself from others in that neighbouring neurons interact with each other: as can be seen from the equation, the input vectors of all the softmax neurons $z_1, z_2, ..., z_k$ serve to enforce $\sum_{i=1}^{k} y_i = 1$. In other words, the vector $(y_1, y_2, ..., y_k)$ defines a probability mass function. This makes the softmax layer ideal for classification: neuron j can be made to represent the probability that the input is an instance of class j. Another attractive aspect of the softmax neuron is that its derivative is quick to compute: it is given by $\frac{dy}{dz} = \frac{y}{1-y}$. Note that the name 'softmax' comes from the fact that competition between the neurons of a softmax layer results in a sort of smooth indicator function for the max: if a softmax neuron's input is maximal across the inputs of the softmax layer, then the maximum is present there and not anywhere else; the output for that softmax neuron will

be close to 1 and the output for the other neurons will be close to zero.

3.4 Feed-forward Architecture

A feed-forward neural network is a representation of a function in the form of a directed acyclic graph, so this graph can be interpreted both biologically and mathematically. A node represents a neuron as well as an activation function f, an edge represents a synapse as well as the composition of two activation functions $f \circ g$, and an edge weight represents the strength of the connection between two neurons as well as a parameter of f. Figure 7 (taken from [17]) illustrates this.

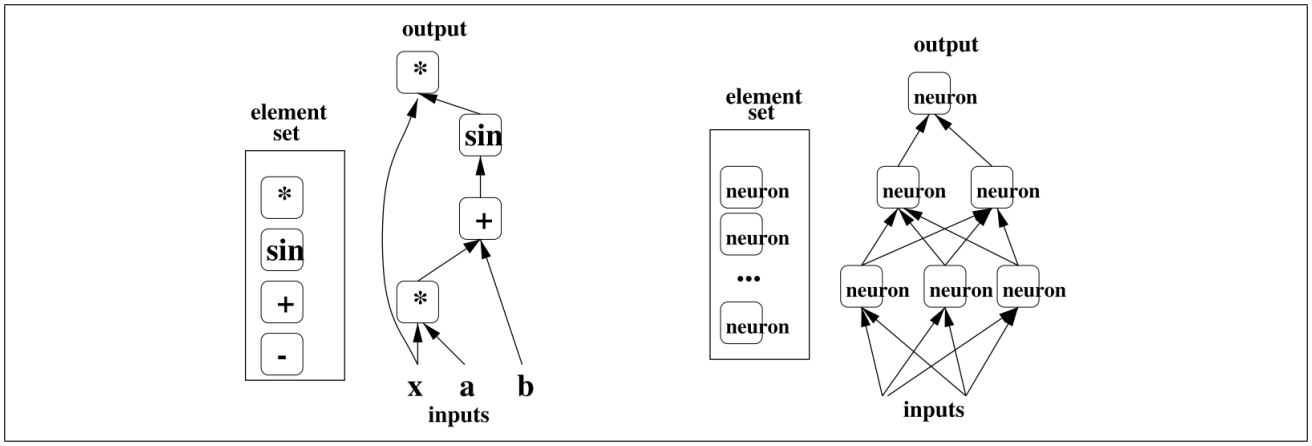


Figure 7: graphical representation of $y = x * sin(a * x + b)$ (source: Bengio 2009)

The architecture is feed-forward in the sense that data travels in one direction, from one layer to the other. This defines an input layer (at the bottom) and an output layer (at the top) and enables the representation of a mathematical function.

Shallow Feed-Forward Neural Networks: the Perceptron A feed-forward neural net is called a perceptron if there exist no layers between the input and output layers. The first neural networks, introduced in the 1960s [17], were of this kind. This architecture severely reduces the function space: for example, with $g_1 : x \to sin(s)$, $g_2 : x, y \to x * y$, $g_3 : x, y \to x + y$ as activation functions (i.e. neurons), it cannot represent $f(x) \to x * sin(a * x + b)$ mentioned above [17]. This was generalised and proved in *Perceptrons: an Introduction to Computation Geometry* by Minsky and Papert (1969) and lead to a move away from artificial neural networks for machine learning by the academic community throughout the 1970s: the so-called 'AI Winter' [**?**].

Deep Feed-Forward Neural Networks: the Multilayer Perceptron The official name for a deep neural network is Multilayer Perceptron (MLP), and can be represented by a directed acyclic graph made up of more than two layers (i.e. not just an input and an output layer). These other layers are called hidden layers, because the 'roles' of the neurons within them are not set from the start, but learned throughout training. When training is successful, each neuron becomes a feature detector. At this point, it is important to note that feature learning is what sets machine learning with MLPs apart from most other machine learning techniques, in which features are specified by the programmer [17]. It is therefore a strong candidate for classification tasks where features are too numerous, complex or abstract to be hand-coded - which is arguably the case with pipe weld images.

Mathematically, it was proved in 1989 that MLPs are universal approximators [21]; hidden layers therefore increase the size of the function space, and solve the initial limitation faced by perceptrons.

Intuitively, having a hidden layer feed into another hidden layer above enables the learning of complex, abstract features, as a higher hidden layer can learn features which combine, build upon and complexify the features detected in the layer below. The neurons of the output layer can be

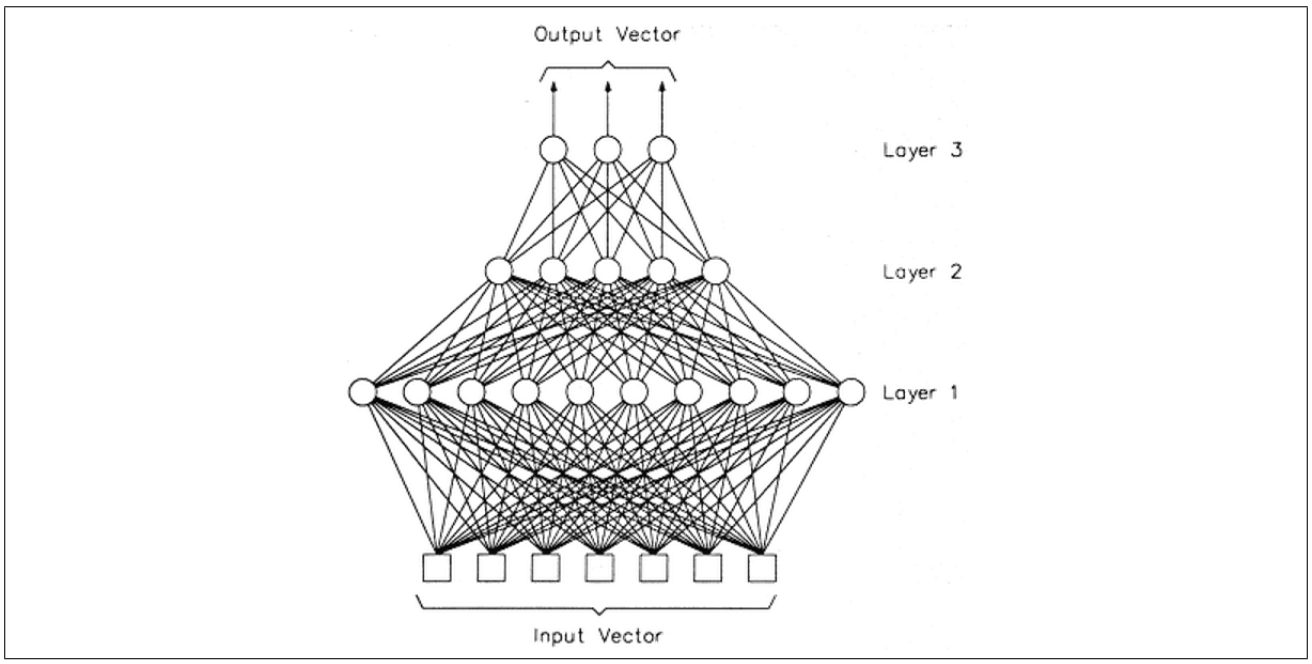


Figure 8: Multi Layer Perceptron with 2 hidden layers

viewed as using information about features in the input to determine the output value. In the case of classification, where each output neuron corresponds to the probability of membership of a specific class, the neuron can be seen as using information about the most abstract features (i.e. those closest to defining the entire object) to determine the probability of a class membership.

3.5 Justifying Depth

To make the case for deep architectures, consider a model with the same number of parameters but fewer layers (i.e. a greater number of neurons per layer). (Goodfellow et al 2013) [4] ran experiments to compare and found that depth is better: intuitively, if neurons are side by side, they cannot use the computation of their neighbour, whereas with depth, the neurons above can make use of the work done by the neurons below.

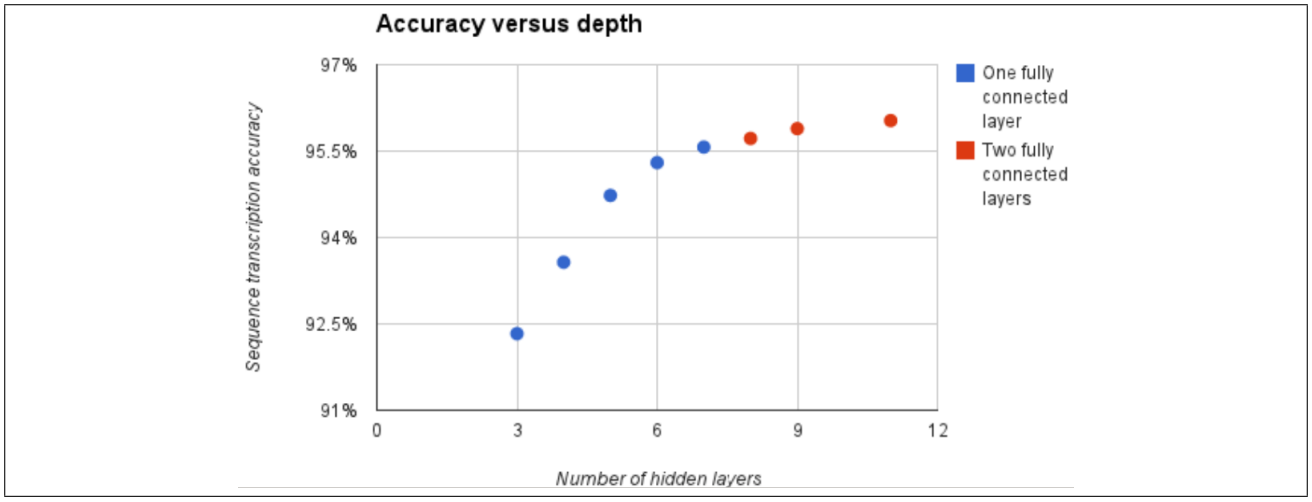


Figure 9: LeNet7 architecture: each square is a kernel

As a formalisation of the intuition given above, (Bengio 2009) [17] advances that having multiple layers creates a **distributed representation** of the data, rather than a **local representation**. Consider for example the one-hot encoding of the integer set $\{0, .., n-1\}$, consisting of n-bit vectors where

the i-th bit is 1 if the vector represents the integer i. This encoding is sparse, because every element of the set is represented by a vector of n-1 zeros and one 1. Moreover, it is local, because the k-th vector entry marks the presence of a feature that is present in only a single element, the integer k. On the other hand, the binary representation of the integers $0, .., n-1$ is a dense distributed representation, because it requires only $log_2(n)$ bits, and the k-th vector entry marks the presence of a feature that is shared by multiple elements of the set: whether or not not the integer is $\geq 2^k$. The binary representation is richer and more powerful than the one-hot representation because its features are not mutually exclusive: each class (i.e. each integer) is represented by a combination of features rather than a unique one. It is in this sense that the representation is distributed, and that the features are general.

(Bengio 2009) [17] details the belief that a deep architecture incites such representations, since, by having a neuron feed into multiple neurons of another hidden layer above, the feature it learns will be used by several feature detectors (or eventually, class detectors) above. The crucial benefit of this is generalisation: features learned from data in a specific region of the input space (a.k.a data space) are used in other regions of the input space, even if little to no data from this region is present to learn from. It is in this sense that the representation is general, not local.

However, it may seem 'risky' to be imposing the use of features in regions of the input space that are unknown. 'As usual in ML, there is no real free lunch' [23]. Indeed, one can think of this as imposing a prior on the function space to be searched, that is of the form: the observed data can be explained by a number of underlying factors which can be learned without seeing all of their configurations. If the prior is true for the classes we are learning, the gains are large. Successes in deep learning seem to indicate that this prior covers many tasks that humans deal with [23]. This prior is particularly powerful when learning from high dimensional data, where the curse of dimensionality means that a vast majority of the configurations are missing from the training set.

3.6 Backpropagation

Now that the architecture of a deep neural network has been motivated, the question remains of how to train it. Mathematically: now that the function space has been explained, the question remains of how this space is searched. In the case of feed-forward neural networks and supervised learning, this is done with gradient descent, a local (therefore greedy) optimisation algorithm. Gradient descent relies on the partial derivatives of the error (a.k.a cost) function with respect to each parameter of the network; the backpropagation algorithm is an implementation of gradient descent which efficiently computes these values.

3.6.1 Compute Error-Weight Partial Derivatives

Let t be the target output (with classification, this is the label) and let $y = (y_1, y_2, ..., y_P)$ be actual value of the output layer on a training case. (Note that classification is assumed here: there are multiple output neurons, one for each class).

The error is given by

$$E = \mathcal{C}(t - y) \tag{6}$$

where $\mathcal{C}$ is the chosen cost function. The error-weight partial derivatives are given by

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} \cdot \frac{\partial y_i}{\partial net} \cdot \frac{\partial net}{\partial w_{ij}} \tag{7}$$

Since in general, a derivative $\frac{\partial f}{\partial x}$ is numerically obtained by perturbing x and taking the change in $f(x)$, the advantage with this formula is that instead of individually perturbing each weight w_{ij}, only the unit outputs y_i are perturbed. In a neural network with k fully connected layers and n units per layer, this amounts to $\Theta(k \cdot n)$ unit perturbations instead of $\Theta(k \cdot n^2)$ weight perturbations [1]. Therefore, backpropagation scales linearly with the number of neurons.

3.6.2 Update Weight Values with Gradient Descent

The learning rule is given by:

$$w_{i,t+1} = w_{i,t+1} + \tau \cdot \frac{\partial E}{\partial w_{i,t}} \tag{8}$$

Visually, this means that weight values move in the direction that will (locally) reduce the error quickest, i.e. the direction of steepest (local) descent on the error surface is taken. Notice that given the learning rule, gradient descent converges (i.e. $w_{i,t+1}$ equals $w_{i,t+1}$) when the partial derivative reaches zero. This corresponds to a local minimum on the error surface. In figure 10, two training sessions are illustrated: the only difference is the initialisation of the (two) weights, and the minima attained in each case are not the same. This illustrates a strong shortcoming with backpropagation: parameter values can get stuck in poor local minima.

3.6.3 Stochastic Gradient Descent

In practice, in order to ensure use precise weight updates, the partial derivatives are obtained by averaging over a number of training cases (which are often said to be grouped in 'mini batches'). This is called Stochastic Gradient Descent [17], and is also referred to as 'mini batch training'.

3.7 Overfit

As mentioned previously, learning is not a mere approximation problem because the hypothesis function must generalise well to any subset of the instance space. A downside to using highly expressive models such as deep neural networks is the danger of overfit: training may converge to a function that,

[1] the bound on weight perturbations is no longer tight if we drop the assumption of fully connected layers

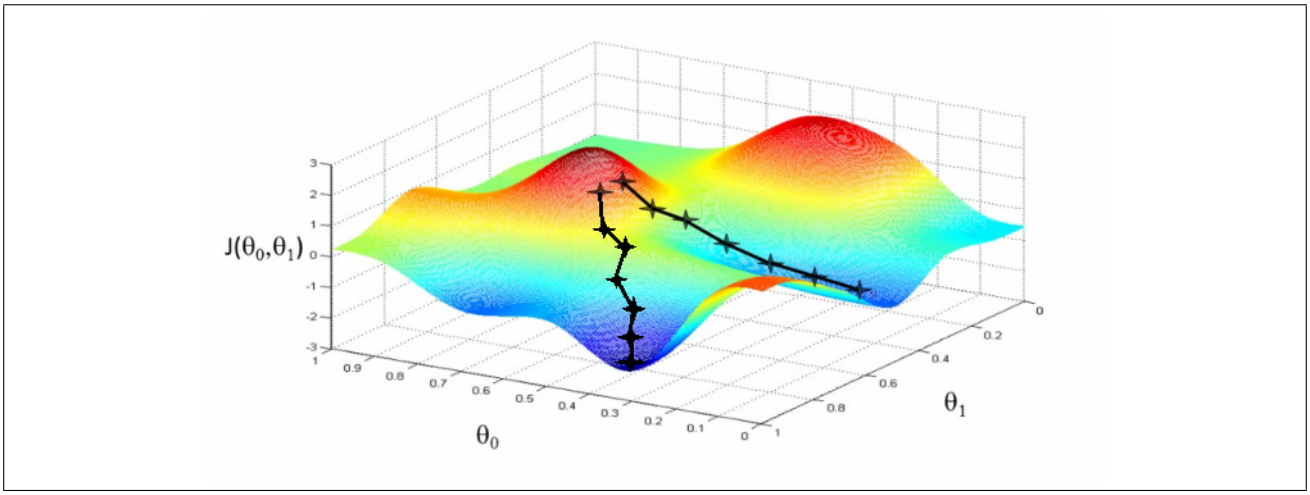


Figure 10: an error surface with poor local minima

despite having zero error over the training set (i.e. perfectly fits the training set), performs poorly on unseen data. Overfit can be easily understood with the regression example of fitting a polynomial to a set of points sampled uniformly with noise from a curve, as shown on figure 11.

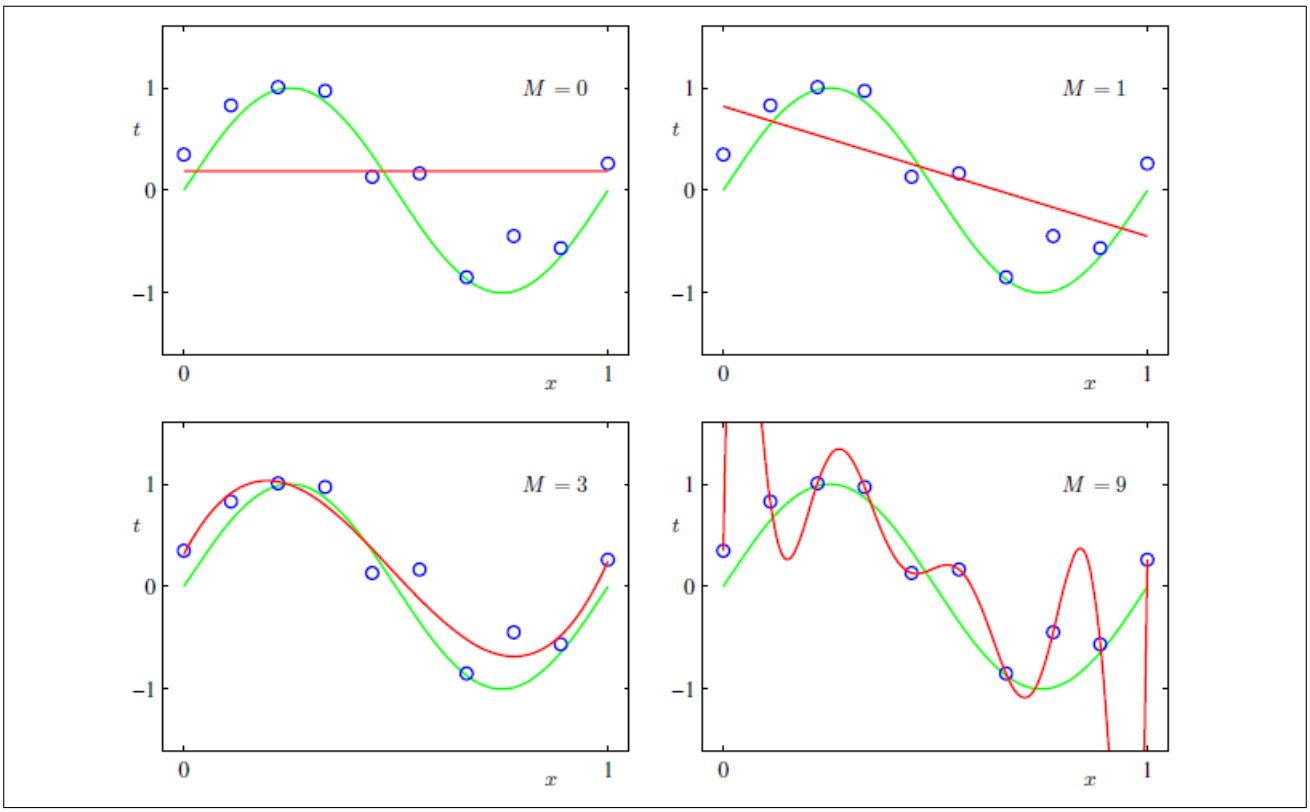


Figure 11: Overfit as polynomial order increases (source: Bishop 2010)

M is the order of the polynomial used and governs the expressiveness of the model. Clearly, the fit of the model increases with M, but one can see that for the case $M = 9$, the given polynomial fits the true (green) function poorly: in other words, the model generalises poorly. How well the model generalises can be evaluated by measuring the fit an another sample of points.

3.7.1 Cross Validation

This is the intuition behind cross validation, which consists in separating the labelled dataset into a training set, a validation set and a test set. The partial derivatives are computed from the error over

the training set, but the function that is retained at the end of training is the one that minimises the error over the validation set, and its performance is measured on the test set. The distinction between the test set and the validation set is to obtain a stochastically impartial measure of performance: since the function is chosen to minimise the error over the validation set, there could be some non-negligible overfit to the validation set which can only be reflected by assessing the function's performance on yet another set.

In practice, **early stopping** is the form of cross validation used: training continues while the validation error decreases, and terminates as soon as it begins to rise (even if the training error is still decreasing). One may realise that a non negligible assumption underlies this: convexity of the time series of the validation error, i.e. once the validation error has reached a minimum, then this is the global minimum. This property is always observed in practice [18], but one may wonder why. Section 'Analyis 2' of the report proposes a mathematical explanation of this.

3.7.2 Data Augmentation

Consider the polynomial overfit figure above: the points on the fitted curve which are furthest away from the true curve are those which lie far away (in terms of X axis distance) from the sampled points. If the sample contained points with those X coordinates, the fitted polynomial would not have scored as low an error, and would not have been the polynomial of best fit. In other words, more data would have reduced overfit. This is what underlies artificial data augmentation, introduced by (Ciresan et al 2011) [8], which consists in applying label preserving transformations to the training set, i.e. 'affine (translation, rotation, scaling, horizontal shearing) and elastic deformations'. This was reported to bring the test error rate on MNIST (a benchmark dataset for image classification consisting of 100,000 28x28 greyscale images of handwritten digits, where each digit defines a class) down from 0.40% to 0.27%.

3.7.3 Dropout

Dropout is a regularisation technique for deep neural networks introduced by (Hinton et al 2012) [7] which consists in preventing co-adaptation of feature detectors by randomly turning off a fixed proportion $k \in (0, 1)$ of neurons at every training iteration, but using the entire network (with weights scaled down by k) at test time. As shown on figure 12, this technique has been shown to improve performance on the MNIST image classification benchmark task, though this increase is only significant when compared to models where neither data augmentation, convolutional layers or unsupervised pre-training are used (for which the best published record is 160 errors).

Dropout reduces over-fitting by being equivalent to training an exponential number of models that share weights in reasonable time: there exists an exponential number of different dropout configurations for a given training iteration, so a different model is almost certainly trained every time. At test time, the average of all models is used, which can be seen as a powerful ensemble method.

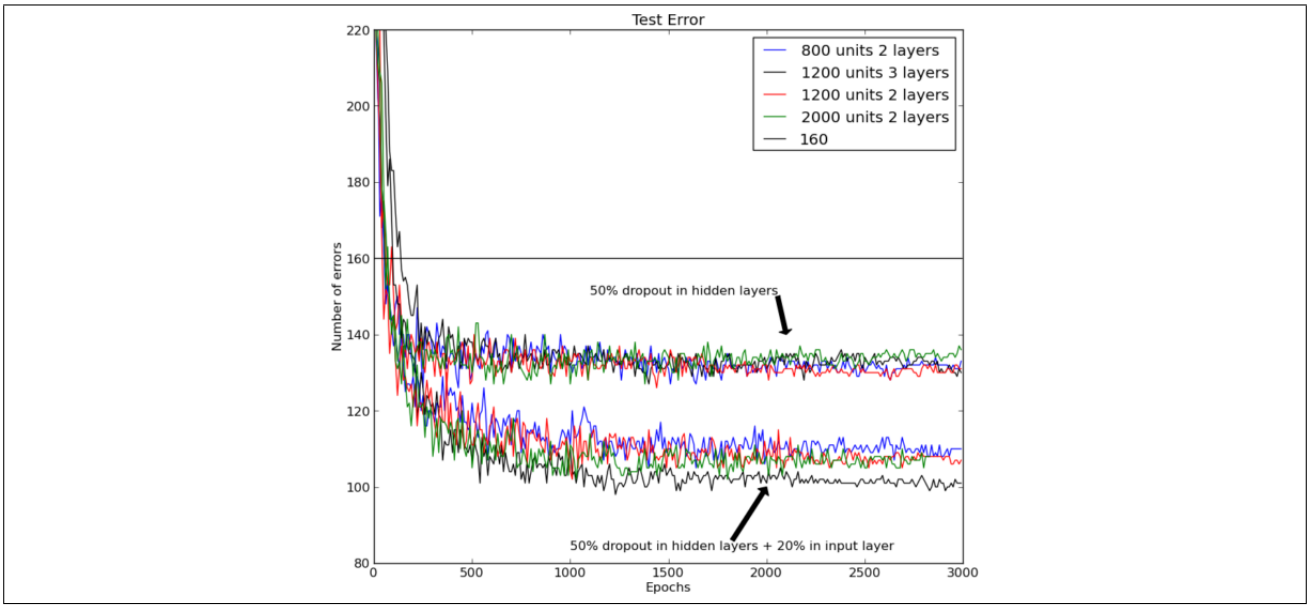


Figure 12: source: (Hinton et al 2012)

3.8 Deep Convolutional Neural Networks

A convolutional neural network is a deep feed-forward neural network with at least one convolutional layer. A convolutional layer differs from a traditional fully connected layer in that it imposes specific operations on the data before and after the data is fed through the activation functions. These specific operations are taken from the pre deep learning era of computer vision, and are detailed in this subsection.

Before diving into the details, a notable point to bear in mind is that these operations impose a prior on the underlying structure of the observed data: translation invariance of the features. Consider the following image of a geranium: a good (albeit complex) feature to classify this image would be the blue flower, regardless of the location of the blue flower. This feature could appear anywhere on the image; therefore, if the network can learn it, it should then sweep the entire image to look for it. It is in this sense that the pixel feature is 'convolved' over the image.

Figure 13: LeNet7 architecture: each square is a kernel

The following subsection is an explanation of the computation that occurs in a convolutional layer. Users of Convolutional Neural Networks for classification tasks are sometimes accused of using them as a black box without understanding them; therefore, this section is crucial in establishing an in-depth understanding of how they function and why they might be so successful.

The architecture from (Krizhevsky et al 2012)'s record-breaking image classification model – which has since been frequently reused for natural scale image classification tasks with CNNs [2] [4] [9] [10] [12] [19] [25] [28] – is summarised on figure 14. The softmax layer and fully connected layers having already been covered in section 3, the subsequent subsection is explains what remains: the inner workings of a convolutional layer.

A[2] convolution layer has a pixel feature i.e. filter which is convolved over the entire image, followed by a non-linearity, followed by a spatial feature, optionally followed by a normalisation between feature responses. This structure is similar to hand-crafted features in computer vision such as SIFT and HOG [3]. The key difference is that each operation is learned, i.e. optimised to maximise performance on the training set.

[2]A large portion of the content from this subsection on explaining the operations in a convolutional layer is heavily inspired from Prof Robert Fergus's NIPS 2013 tutorial [10].

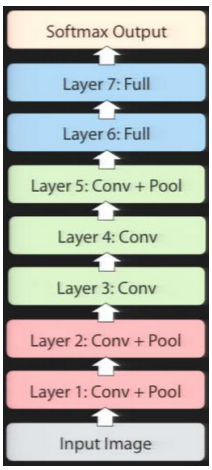


Figure 14: Architecture of CNN from (Krizhevsky et al 2012)

3.8.1 Pixel Feature

A[3] $k \times k$ pixel feature, also referred to as sliding kernel or sliding filter, is defined as a $k \times k$ matrix $\mathcal{W}$ that is applied to $k \times k$ windows $\mathcal{X}$ of the image by performing the matrix dot product $\sum_{i=0}^{k-1} \sum_{j=1}^{k} x_{ij} \cdot w_{ij}$. An example with $k = 3$ is shown on figure 15.

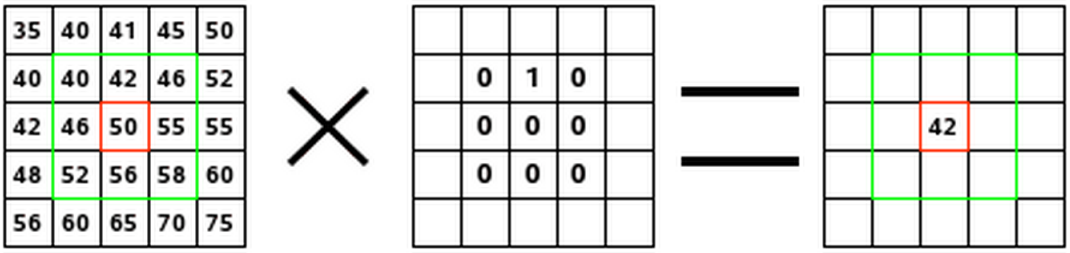


Figure 15: Selecting a pixel window and applying a kernel to it (source: convmatrix Gimp documentation)

Notice that this is equivalent to the vector product component of the generic neural network inputs combination $z = b + \sum_{i=0}^{n-1} x_i \cdot w_i$, where the pixel matrix representation of the entire image is flattened into the vector $\mathbf{x}$, the weights of the kernel are flattened into a portion of the weight vector $\mathbf{w}$, and all weights corresponding to a pixel that is not part of the window are set (and fixed) to zero. Fixing many weights to zero imposes a strong prior on the network and significantly reduces the function space, making it easier to search for good functions. The crucial aspect of CNNs is that, by representing these kernels as weight vectors of the network, a large set of optimal features can be learned over the dataset without having to handcraft them, as were for example SIFT and HOG [3].

By convolving the kernel over the image, one obtains a feature response map (referred to as kernel map in computer vision jargon):

Another advantage of such pixel features is that one can make sense of what the weights represent i.e. what the network is learning. For example, consider the following image, sliding kernel and resulting kernel map:

A zero value is obtained when pixel $(1, 0)$ and pixel $(1, 1)$ have the same value, which is likely to be the case, hence why most of the kernel map is black. The dot product is maximised when pixel $(1, 0)$ is of a much lower value than pixel $(1, 1)$, which occurs most often when the window is on a vertical edge separating a dark region to the left from a light region to the right. Therefore, an intuitive representation for the kernel matrix is the pixel window on figure **??** [4], which has become

[3] This subsection on explaining sliding kernels is heavily inspired from a blog post by Christopher Colah [12].

[4] the pixel representation indeed has more pixels than the kernel has entries. but one can see that the dot product is

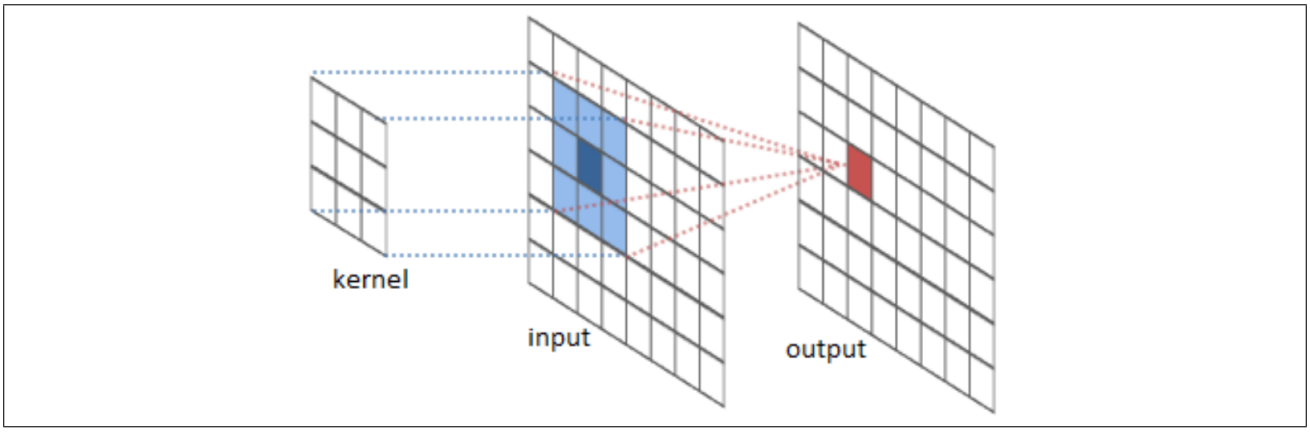


Figure 16: Producing a kernel map (source: River Trail documentation)

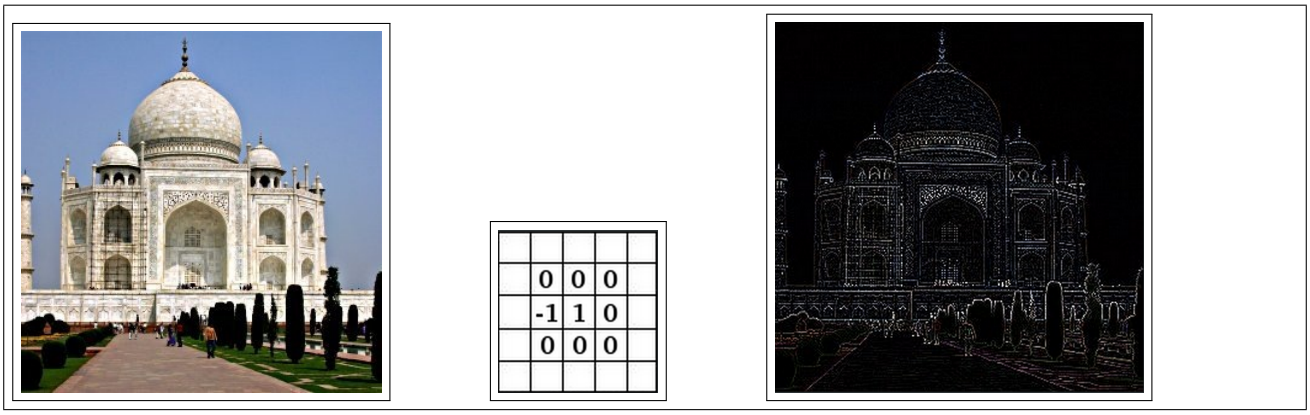


Figure 17: Image, kernel and resulting kernel map (source: convmatrix Gip documentation

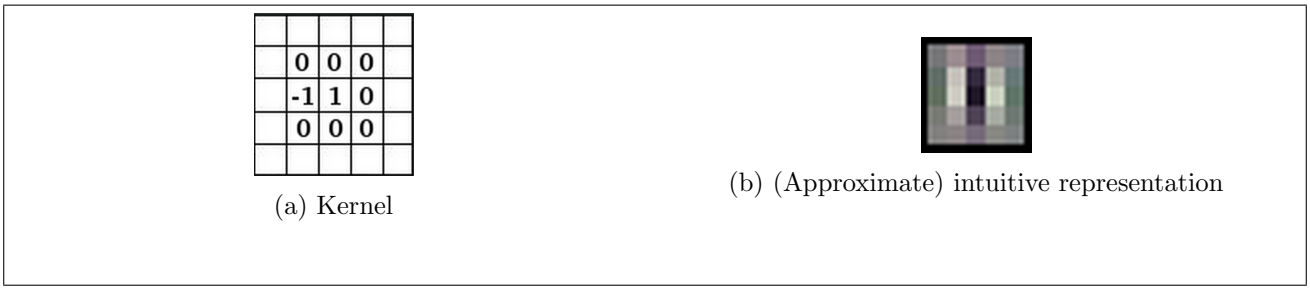
a popular way of representing 'deep learning [computer vision] features' [19]. One therefore realises that this feature visualisation technique does not show a set of weights learned by the network, but the pixel window which maximises the dot product between them and the set of learned weights.

In order to impose convolution i.e. application of the same kernel over the entire image, the learning rule (8) given in section 3.3.2 is modified to:

$$w_{i,t+1} = w_{i,t+1} + \tau \cdot \sum_{j \in \mathcal{K}} \frac{\partial E}{\partial w_{j,t}} \tag{9}$$

Where $\mathcal{K}$ is the index set for all neurons intended to apply the kernel to a different location on the image. This learning rule ensures that the i-th weight of all such neurons are updated by the same value. At initialisation, each i-th weight is also set to the same value. The result is that the weight vectors for all these neurons are identical throughout training, i.e. the corresponding kernel is convolved over the entire image.

maximal when the kernel is applied in the upper middle area of the pixel representation



(a) Kernel

(b) (Approximate) intuitive representation

3.8.2 Non-linear Activation

Going to be applied to each output of the feature map independently, no interaction between the elements in the feature map. it's here because the kernel convolution is a linear operation, and we want non-linearity. the currently best performing known activation is the ReLU (extensively discussed in section 5 of the report). an important element is the bias, which in the case of the ReLU allows to choose the input threshold at which the neuron turns off. the bias is the same across every neuron of the feature map.

3.8.3 Pooling aka Spatial Feature

Take a $k \times k$ spatial neighbourhood of the feature map, and compute some summary statistic over it (average or max). The most common operations are average and max. Max is the currently best performing pooling operation, and this is theoretically backed by (Boureau et al 2010). The intuition behind the theoretical analysis can be illustrated by the following example: given a pixel feature, a 3×3 spatial neighbourhood to pool over and two different images, consider the case where image A has the feature present in a single location of the neighbourhood (activation output is 1 for this location, 0 elsewhere), whereas image B does not (activation outputs are 0 across all locations of the neighbourhood). Since the feature map seeks to discriminate inputs based on the presence of its feature, we would wish for the pooling operation to assign starkly different values to images A and B. With max pooling, A receives pooled value 1 and B receives pooled value 0; with average pooling, A receives a mere $\frac{1}{9}$ and B receives 0. Therefore, max is better.

In general, a potential advantage of average over max is that it conserves more information; on the other hand, it will dilute an activation that is strong in a single location of the given neighbourhood and weak in the others. In the worst case, opposing activations will cancel each other out, and result in the pooled value as that for a neighbourhood with zero activations (however, note that this does not occur with ReLUs since they are not antisymmetric).

The key contribution of max pooling is robustness to noise and invariance to small transformations, since the same max pooling output is obtained regardless of where in the neighbourhood the maximal feature activation occurs, and regardless of what is present elsewhere in the neighbourhood. At higher convolutional layers, the resulting invariance can be spectacular. For example, figure **??** shows kernel maps for a certain feature of the 5-th convolutional layer of a net from (Zeiler and Fergus 2013) [19], computed for 9 different input images from the same class ('human'), which obtain similar pooled activation values. One may be struck by the fact that they receive similar pooled activation values, since one can tell that the similarities between the kernel maps are complex and would be difficult to describe in terms of 'small transformations'.

By stacking n convolutional layers with $k \times k$ pooling operations, the output of a pooling neuron on the n-th layer has a $k^n \times k^n$ receptive field over the image. Choosing the number of convolutional layers is therefore a matter of finding n such that k^{2n} equals the dimensionality of the input set, i.e. such that the highest convolutional layer can recognise complex features spanning the entire image, which is when total translation invariance is achieved.

3.8.4 Contrast Normalisation

Pick a single location of the output, have a little spatial neighbourhood around that, and perform the linear transformation on all the output values in this neighourhood that results in the best fit of a zero mean, unit standard deviation Gaussian. These transformations are used in image processing to handle images where contrast intensity varies across different areas of the image. The neighbourhood can be defined geographically over isolated kernel maps, or across kernel maps. This normalisation achieves a rescaling of the features at each layer. For example, if in a certain neighbourhood output

(a) 9 input images of the "human" class

(b) 9 corresponding kernel maps with similar pooling outputs

Figure 19: source: (Zeiler and Fergus 2013)

values are all low and in another neighbourhood all output values are high, this may be due to photographical factors (e.g. different expositions to light) which have nothing to do with the underlying objects of interest; local contrast normalisation will pull up the values of the low value neighbourhood and reduce those of the high value neighbourhood to put both neighbourhoods on an equal footing for the next layer to receive unbiased data.

3.9 Local vs Global Optimisation

Gradient descent is a local optimisation algorithm in that it ends when the first local minimum has been met. This is a major shortcoming when the error surface – defined by the deep architecture and the dataset – has minima that vary significantly in depth. However, empirical evidence suggests that this is not the case for convolutional neural networks [17]. Given a sufficiently large labelled dataset, the performance difference reached by a CNN trained with backpropagation with and without unsupervised pretraining on another dataset is negligible [2].

One can imagine that the requirement of a large enough dataset is to obtain a sufficiently precise approximation of the 'true' error surface, i.e. the error surface that would be obtained by integrating the gradient over all possible configurations of the class. But the theoretical explanation for depth similarities across local minima remain a mystery. The question was asked by this author to Yann LeCun at a conference for students in Paris on 12th June 2014. The answer was that:

'The optimisation is simple because there are results from random matrix theory that seem to suggest that when you build those systems, the function you are trying to optimise is akin to a high degree polynomial on the sphere, with lots of monomials. The properties of the critical points which are saddle points are relatively well analysed by people who have traditionally worked on things like spin glass theory. What is known is that the minima are clustered within a very small narrow band of energies, so if you have a process that's going to find a minimum, it will find one that will be as good as any minimum. The likelihood of being trapped in a bad minimum is small because there are exponentially more good minima than bad ones. Of course, there is an even larger number of saddle points, so you have to figure out how to avoid saddle points; but you don't get trapped by saddle points, you only get slowed down by them' [15].

No papers regarding this research were found. When the question was reposted to the Google+ Deep Learning community, Yann LeCun answered that *'This work is not yet published. Stay tuned'.*

The objective of the literature review until now has been to serve as a preamble, in order to understand the models which were used throughout the project. The two subsections that follow are more specific and relate to challenges encountered when training classifiers for ControlPoint. The papers discussed were read in an ad hoc manner, so their relevance will become clearer later on in the Experiments sections that bear the same name. Therefore, especially in the case of the class imbalance literature, deeper examination of the papers is left for then.

3.10 Transfer Learning

Transfer learning consists in initialising the weights of layers of a network to those of dimensionally identical layers of a network trained in a supervised fashion on a similar task. Until 2013, the established approach consisted in transferring the weights of a Deep Belief Net trained (in an unsupervised fashion) on a 'source' unlabelled dataset [27]. However, the 2014 paper "CNN Features off-the-shelf: an Astounding Baseline for Recognition" by Razavian et al shows that transferring the weights of OverFeat – a model that is identical to AlexNet in architecture and winner of the ILSVRC 2013 challenge [25] – to initialise the training of classifiers on 11 well known computer vision recognition tasks, establishes state-of-the-art results on 10 of them [11].

The transfer models are obtained by training linear Support Vector Machines on the feature space defined by the bottom seven layers of OverFeat. The subsection below is therefore a rapid overview of the mathematics that underlie the training of a linear SVM.

3.10.1 Linear Support Vector Machines

The hinge loss learns the linear decision boundary for classifying inputs which minimises the number of mis-classifications and maximises the *margin*, which is defined as the smallest distance between the decision boundary and any of the training cases [18]. Mathematically, the decision boundary is the hyperplane defined by the equation $\mathbf{w}^T\phi(\mathbf{x}) + b = 0$ where the parameters $\mathbf{w}$ and b are solution to:

$$arg\max_{\mathbf{w},b}\{\frac{1}{||\mathbf{w}||}\min_{1\leq i\leq n}\{t_i(\mathbf{w}^T\phi(\mathbf{x}_i)+b)\}\} \tag{10}$$

Where n is the size of the training set, $t_i \in \{-1, 1\}$ is the label for training case $\mathbf{x}_i$, ϕ is the function represented by the first seven layers of the transferred model, and $\mathbf{w}, b$ define the linear computation that produces classification predictions, namely:

$\mathbf{w}^T\phi(\mathbf{x}) + b > 0$ →predicted label value is 1
$\mathbf{w}^T\phi(\mathbf{x}) + b < 0$ →predicted label value is -1

The intuition behind the mathematical formula is that the distance from a point $\mathbf{x}_1$ to a hyperplane defined by the equation $h(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) + b = 0$ is given by $\frac{|h(\mathbf{x}_1)|}{||\mathbf{w}||}$. In the case of classification, when the decision boundary correctly separates the entire trainin set, $|h(\mathbf{x}_1)| = t_1 h(\mathbf{x}_1)$. $\min_{1\leq i\leq n}\{t_i(\mathbf{w}^T\phi(\mathbf{x}_i)+b)\}$ finds the points closest to the decision boundary.

Solving this optimisation problem with brute force would be intractable [18], but it can be shown that it is equivalent to the following constrained optimisation problem:

$$arg\min_{\mathbf{w},b}||w||^2, \text{ subject to } \forall 1 \leq i \leq n, t_i(\mathbf{w}^T\phi(\mathbf{x}_i)+b) = 1 \tag{11}$$

3.11 Class Imbalance

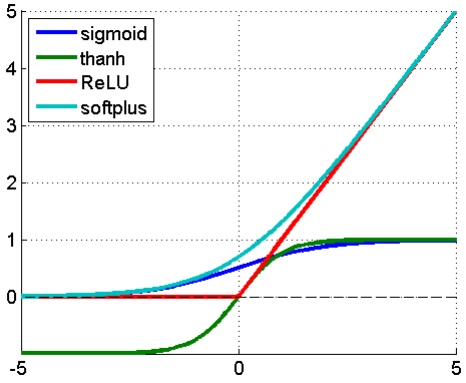
Definition The literature [34] [31] [32] [33] has defined class imbalance as the situation where the sample distribution of classes is significantly non-uniform, i.e. where there exists a class of significantly

smaller size than another. However, as will be seen later, this definition does not necessarily make class imbalance detrimental to learning. A definition for 'dangerous class imbalance' will be given with the aim of being the largest set of conditions that are necessary for class imbalance to be detrimental to training classifiers with stochastic gradient descent and standard cost functions.

The literature on deep learning with class imbalance was found to be scarce. Zhou and Liu in 'Training cost sensitive neural networks with methods addressing the class imbalance problem' (2005) [32] experiment with **under-sampling**, textbfover-sampling and **threshold-moving** to deal with class imbalance, and find threshold-moving to perform best.

'F-measure as the error function to train neural networks' by Pastor-Pellicer et al, 2013 [33], does not make use of these techniques and instead introduces a cost function for binary classifiers which is the harmonic mean of precision and recall. Deep neural networks are trained on it as well as on the Mean Squared Error for 3 image-cleaning tasks with levels of class imbalance below those experimented with in this report. The image-cleaning task is framed as a classification task where the probability of ink in each pixel of the clean image is to be predicted given the noisy image. MSE as a cost function for training deep neural network classifiers is known to train slower and converge to the same performance as when the cross entropy error is used [35].

The authors find that 'both training techniques perform quite well' on the basis that 'a well cleaned image gives good results on both metrics'. This seems to merely state that the error computed on a correctly classified input is low in absolute terms, for both cost functions. An analytical comparison of the cost functions by rescaling input domain and output range, such as the one between softmax cross entropy and hinge loss in (Bishop 2010) [18] and in the previous section, is not included. An empirical comparison of the cost functions by evaluating a model trained on each against a common performance metric such as percentage classification accuracy is not provided either. It appears that this cost function does not deliver significant performance gains.



(a) Activation functions

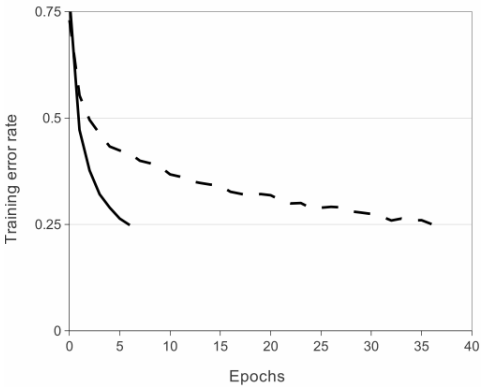


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons

(b) source: Krizhevsky et al 2012

4 Analysis 1: ReLU Activation

4.1 Motivations

Differentiable, symmetric, bounded activation functions that introduce a non-linearity smoothly over the entire input domain were the activation units of choice in deep feed-forward neural networks, until Rectified Linear Units were used for CNNs in 2012 [1] and found to outperform their counterparts. They make the observations that $f(x) = \tanh x = \frac{1-e^{-2x}}{1+e^{-2x}}$ or $f(x) = (1+e^{-x})^{-1}$ 'saturating non-linearities' are slower to train than the $f(x) = max(0, x)$ 'non-saturating non-linearity', and report an 'accelerated ability to fit the training set', but do not provide any explanations.

The importance attributed to non-linearity of activation functions is due to the function space that is spanned by composing such functions as deep neural networks do. If activation functions are linear, the overall network is linear too, and as seen in section 3.2.2, such networks are greatly limited in what they can learn. Therefore, one may wonder whether or how the function space is reduced by using an activation function such as the ReLU that is non-linear only in the neighbourhood of 0.

(Glorot, Bordes and Bengio 2013) [2] explain that 'the only non-linearity in the network comes from the path selection associated with individual neurons being fired or not. [...] We can therefore see the model as an *exponential number of linear models that share parameters*'. Since for a certain range of inputs, a ReLU will not fire, one can view the network as selecting different subsets of itself based on the input. Each subset is a linear function, but two inputs which are very close (in the sense of euclidean distance) in the input space might (de)activate different neurons, and therefore be processed by different (linear) functions that output completely different results. In this sense, non-linearity is achieved.

(Glorot, Bordes and Bengio 2013) also remark that 'because of [the ReLU's] linearity, gradients flow well on the active paths of neurons (there is no gradient vanishing effect due to activation non-linearities of sigmoid or tanh units)', but no explanations of enhanced flow or vanishing gradient are given (instead, the paper focuses on the benefits of sparse representations and their greater resemblance to neuron activations in the brain). Another empirical finding of the paper is that 'deep rectifier networks can reach their best performance without requiring any unsupervised pretraining'. This section of the report is a simple mathematical analysis of backpropagation that conveniently provides an explanation for these points as well as (Krizhevsky et al 2012)'s observations. In the

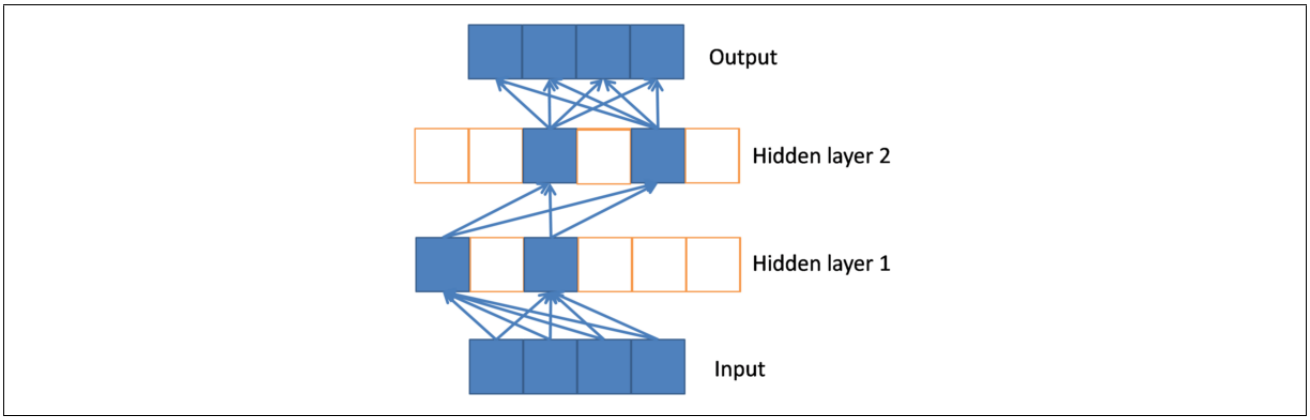


Figure 21: ReLU path selection (source: Glorot et al 2013)

context of building pipe weld classifiers, the motivation for doing is to understand the mechanics of training for the CNNs to be subsequently used.

4.2 Mathematical Analysis

Recall from section 3.3 that the model is trained with backpropagation: each of the weights w are adjusted by $\tau \cdot \frac{\partial E}{\partial w}$. The choice of activation function modifies $\frac{\partial E}{\partial w}$; this section looks at how ReLU does so compared to sigmoid or tanh.

4.2.1 How the Gradient Propagates

It may be useful for intuition to think of $\frac{\partial E}{\partial w}$ in the context of the gradient travelling through the network. With the following notation:

- y_j, the output of unit (a.k.a neuron) j, but also used to refer to the unit j itself
- w_{ij}, the weight of the edge connecting lower-layer neuron y_i to upper-layer neuron y_j
- $z_j := b + \langle x, w \rangle = b + \sum_{i=1}^{k} x_i \cdot w_{ij}$, the input vector for y_j
- ψ the activation function used – therefore $y_j = \psi(z_j)$
- f' to denote the derivative of any function f that takes a 1-dimensional input

The backpropagation algorithm can be formulated as a set of rules for propagating the gradient through the network:

– to **initialise**: $grad \leftarrow \mathcal{C}'(y_L)$, where y_L is the output unit

– to **propagate through a unit** y_j: $grad \leftarrow grad \cdot \psi'(z_j)$

– to **propagate along an edge** w_{ij}: $grad \leftarrow grad \cdot w_{ij}$

– to **stop at an edge** w_{ij}: $grad \leftarrow grad \cdot y_i$

4.2.2 An Example

Given the figure above:

– for $\frac{\partial E}{\partial w_{31}}$: initialise, propagate through y_1, then stop at w_{31}: $\mathcal{C}'(y_1) \cdot \psi'(z_1) \cdot y_3$

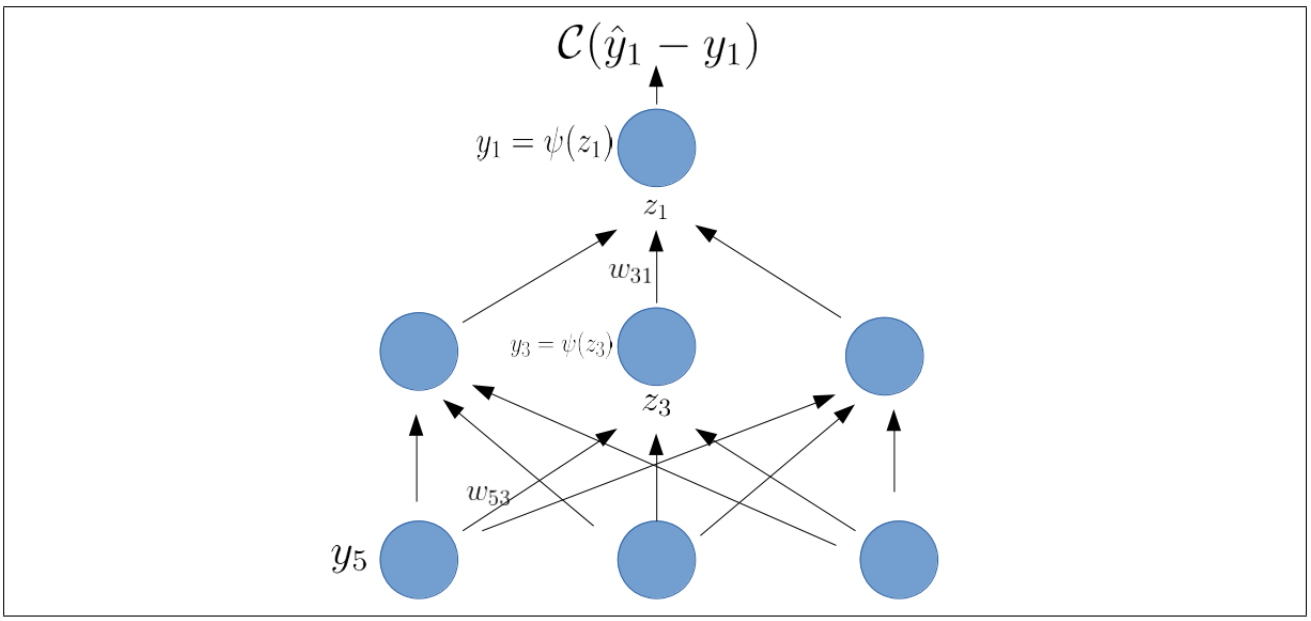


Figure 22: $\mathbb{R}^3 \to \mathbb{R}$ MLP with 1 hidden layer

– for $\frac{\partial E}{\partial w_{53}}$: initialise, propagate through y_1, then along w_{53}, then stop at w_{53}:
$\mathcal{C}'(y_1) \cdot \psi'(z_1) \cdot w_{31} \cdot \psi'(z_3) \cdot y_5$

Intuitively, the partial derivative with respect to a weight can be roughly seen as the product of the partial derivative of every component along the path from the weight to the output unit [5].

4.2.3 Vanishing Gradient

Notice that $\psi'(z_1)$ is a factor in both partial derivatives. Now, consider the derivatives of the tanh and sigmoid functions:

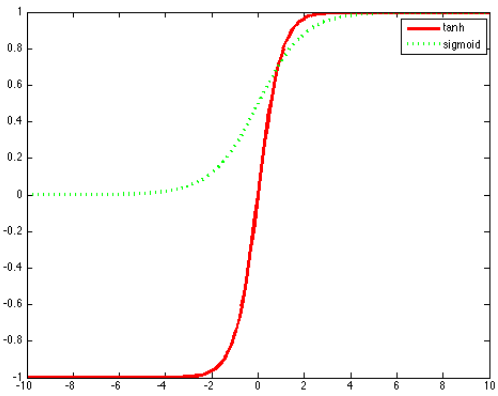
$$tan'(x) = 1 - tan^2(x) \tag{12}$$

$$sigmoid'(x) = \frac{e^x}{(1+e^x)^2} \tag{13}$$

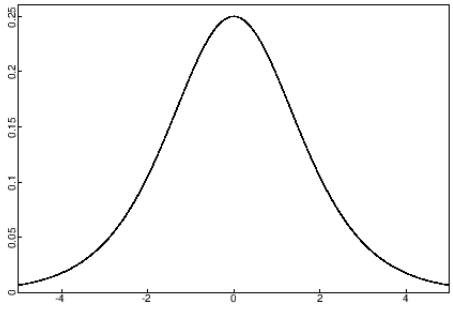
The formulae do not lend themselves to intuition, but their graphical representation given on figure **??** does [6]. Such functions are called 'saturating' by (Krizhevsky et al 2012) because, when the input is high in absolute value, the derivative approaches zero. In the context of backpropagation, this means that a tanh or sigmoid unit that is 'heavily activated' during the forward pass will strongly reduce the gradient as it propagates through it during the backward pass. Moreover, this will affect the partial derivative of every weight that lies behind the unit in some input-output path. As a result, during training it becomes slow to alter any such weight when the unit's weights are on average high in absolute value. This difficulty increases with the depth of the network, since in order to reach a weight that is low down in the network, the gradient must propagate through a higher number of units, so its probability of vanishing increases. This is consistent with empirical findings [17].

[5] this only goes for this simple case where we have one hidden layer and one output node. That is because, if we consider all paths from an input node to an output node, every edge exists in exactly one path. However, if we had more output nodes or more hidden layers, there would exist edges belonging to several input-output paths. In this case, the partial derivative would be the sum across all paths from the weight to some output unit.

[6] both are the same up to a scaling factor, therefore only one is given.



(a) tanh and sigmoid functions



(b) derivative for the tanh and sigmoid

One may argue that a unit which has learned a feature that is useful for the task is one that is heavily activated for certain inputs. Therefore, it is a good thing that it becomes rigid to change. However, traditional training of deep neural networks occurs by first randomly initialising the weights. At first, it is therefore unlikely that a unit is heavily activated because it is a detector for a meaningful feature. If the network is built on saturating activations, such non-meaningful features will be harder to get rid of.

4.2.4 Impact of the ReLU

On the other hand, the derivative of the ReLU is given by:

$$ReLU'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

Which means that for any positive input, the gradient will *not be modified* when propagating through the unit. It is in this sense that 'gradients flow well on the active paths of neurons'. On the other hand, for any negative input, the gradient is brought to zero i.e. halted from propagating altogether. This could be cause for concern, because if the weights of a ReLU are initialised to values that, given the distribution of the input, always lead to a zero activation, then there is no way of modifying them, and the neuron is effectively a useless component of the network. When training, one should therefore take care to choose a sufficiently high positive value for the neuron's initial bias [7]. On the other hand, halting the gradient in this regard has the intuitive benefit of reducing the areas of the network to which it propagates, and focusing it on the areas responsible for the error. If one goes back to the vision of a deep sparse rectifier network as an exponential number of linear models, gradient halting enables the training of a smaller network at every iteration.

Since vanishing gradient makes it slow to get rid of non-meaningful features that receive heavy activations, this provides an explanation for why 'deep sparse rectifier networks can reach their best performance without requiring any unsupervised pre-training', which initialises the weights of the network to features capable of reconstructing the data. It also explains (Krizhevsky et al 2012)'s observations that ReLU networks train faster. By providing a clear mathematical explanation to the main empirical findings regarding deep sparse rectifier networks, this report would argue that the ability of ReLU to enable gradients to 'flow well' is the key reason for their success.

A final remark concerning ReLUs is that they reduce the computation in both forward and backward passes by turning off a number of neurons and by not interfering with the gradient as it

[7] This makes the bias b in $z = b + \langle x, w \rangle$ a crucial element of a neural network, despite them being conventionally excluded from graphical representations

propagates through a unit.

5 Experiments 1: Identifying Class Imbalance

5.1 Implementation: Cuda-Convnet

Due to its success and frequent re-use [2] [4] [9] [10] [12] [19] [25] [28], the network architecture from (Krizhevsky et al 2012), often referred to as AlexNet, was chosen for this task (and throughout most of the project).

Cuda-Convnet is an open-source GPU implementation for training deep convolutional neural networks with stochastic gradient descent, written in CUDA C++ and python by Alex Krizhevsky. GPU implementations enable twenty-fold speedups [30] in training time. Knowledge of its use existed prior to this project since it had already been used for a group project. A shortcoming with the API is that it expects data in the form of batches consisting of numpy arrays of stacked jpg values in matrix format with RGB values split across, and a dictionary of labels and metadata. Python programs were written to achieve this and extract training data from the log files and plotting it. By re-using code written during the group project, the additional code needing to be written was limited to approx. 400 lines. The hardware used for training the network was an nVidia GeForce GTX 780 with 4GB RAM, which enables a twenty-fold increase in training speed compared to the CPU.

5.2 Experimentation

Training occurred over the 113,865 image Redbox dataset only, to exclude domain change as a potential reason for weak performance if it were to occur. The task consists in learning three classes: 'No Clamps Used', 'Photo Does Not Show Enough Of Clamps', and 'Clamp Detected' – which in fact is the default class: an image belongs to it if none of the two mentioned flags have been raised. Time series for training and validation errors were extracted, but not the test error, since it serves no purpose in training the model.

The error being computed and minimised by Cuda-Convnet is the negative of the log probability of the likelihood function:

$$-\frac{1}{n}\sum_{i=1}^{n} log(f(x_i, l_i, W)) \tag{15}$$

where $f(x_i, l_i, W)$ is the model's predicted probability for input x_i's label to be l_i, W are its parameters, n is the mini-batch size. This is also known as Maximum Likelihood Estimation; therefore backpropagation converges to the parameter values that maximise the joint likelihood of the training data. It can be shown that this is equal to the cross entropy of the softmax layer's output [17]. This is discussed in greater length in the experimentation section of Task 3 on class imbalance.

5.2.1 Non-Converging Error Rates

Training produced worrying results: none of the models trained showed signs of learning anything. The figure 24 plot of the training and validation errors covers 11 of the 44 epochs over which the model was trained, which amounts to two consecutive days of training. The errors display no trends, and this extends to all 44 epochs.

These results were confusing: the fact that the training error does not converge to zero means that the model is unable to perfectly fit the training set. This should be impossible, since the 60 million parameters of AlexNet make it a very powerful model[8] for which (Krizhevsky et al 2012) report overfitting to be 'a significant problem', even on the 1000 class, 1.3 million image dataset of the ILSVRC 2012 competition. It makes no sense that the same model would fail to fit a 3-class, 113,000 image dataset.

Moreover, with backpropagation, the network parameters are guaranteed to converge [17] since gradient descent will update weights by zero values once the error-weight partial derivatives are all zero. Such a point in the parameter space exists if there is a local minimum in the error surface at this point. If so, then gradient descent is guaranteed to reach a minimum: intuitively, if one is in a mountain range, then by walking downhill, one is bound to reach some morsel of flat ground at some

[8] powerful as an approximator, i.e. as a 'fitter to the data'.

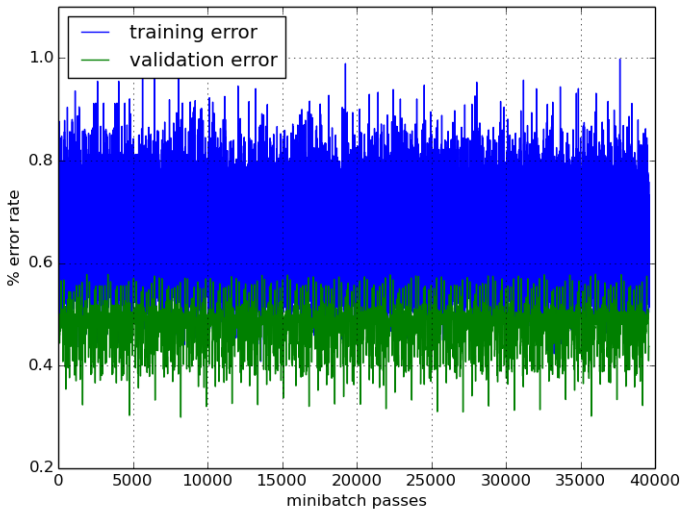


Figure 24: Test Run Training Results

point. Once the parameters have converged, the model is fixed, so its error is expected to be similar across random samples of the training and validation sets. In this case, the persistent high amplitude of the training and validation errors means that the error is heavily changing all the time.

A number of potential explanations for the high amplitude of the error rates were considered:

- The learning rates are too high: the minimum keeps getting 'overshot', the weights move endlessly around the rims of a bowl on the error surface[9].
- The dropout rate is too high: since neurons are randomly dropped at every iteration, a different model is tested every time. The errors correspond to partial derivatives of different models every time, so the amplitude is high.
- The number of parameters is too high: AlexNet contains 60 million parameters, far more than than the number of training cases, so collinearities between the parameters cannot even be broken, and most of them are rendered useless.
- The error rates are not computed correctly.
- Class imbalance: with 90% of the data belonging to the same class, there is not enough information about the 'No Clamps Used' and 'Photo Does Not Show Enough of Clamps' classes to be able to learn features for them.
- Mislabelled data: the images were not tagged correctly, too many members of one class appear in the other and vice versa, so nothing can be learned.

The learning and dropout rates were modified to no avail. (Jarrett et al 2009) report successful training of networks for which 'the number of parameters greatly outstrips the number of samples'.

5.2.2 Increase Validation Error Precision

If the set of images that the validation error is computed against varies from one iteration to the next, then the variations in validation error are not solely explained by the changes in the model parameters; they may also be due to changes in the validation set. Secondly, precision increases with the size of the set. Therefore, by computing the validation error on a larger and unchanging sample of images, one obtains stable, more precise measures that truly reflect what the model is learning.

[9]This interpretation was supported by the Google+ community when the plots were posted.

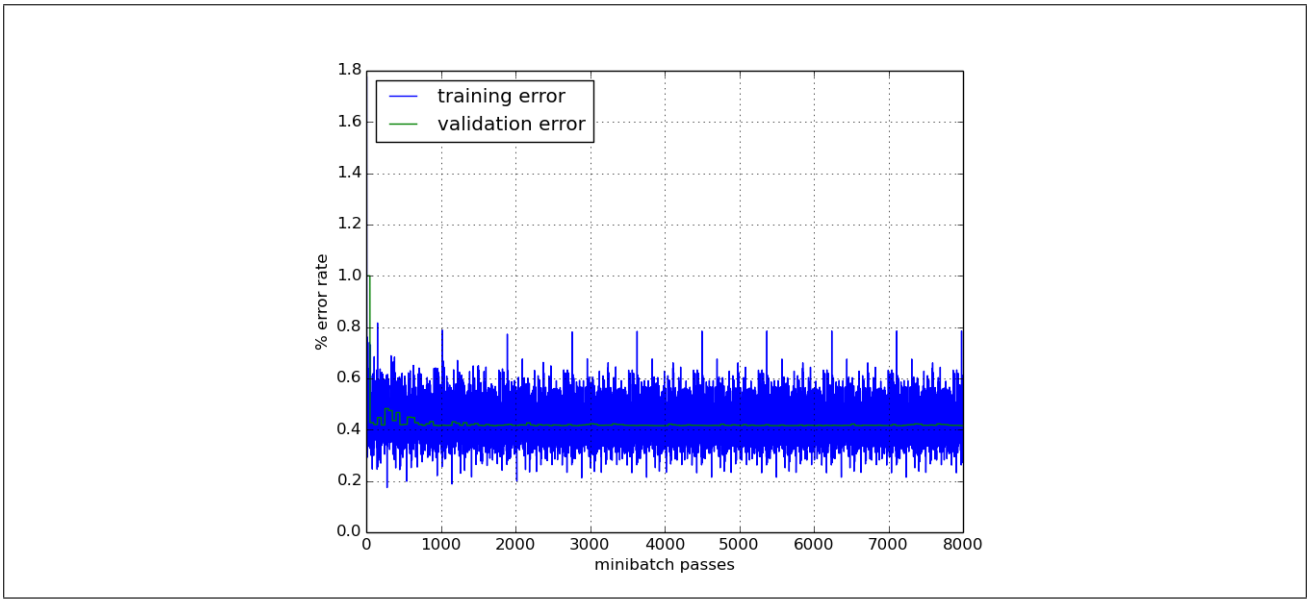


Figure 25: Test Run with validation errors computed over an unchanging set

With better validation error measures, one notices that the validation error is flat throughout most of training, indicating that little is learned. There is some movement during the first thousand iterations, suggesting in fact that backpropagation converged within 1000 iterations, which is highly rapid compared to similar tasks [9] [10] citetransfer-learning [28]. One may be tempted to conclude that the model is able to learn only very little.

5.2.3 Periodicity of the training error

What remains striking and unexplained is the high amplitude of the training error, and its refusal to converge to zero. Another intriguing aspect is its periodicity: its period corresponds to the number of mini-batches in an epoch. Since the same error is obtained on a mini-batch from one period to the next, the network parameters are therefore not changing from an epoch to the next. Therefore, backpropagation does indeed converge: it does so within a mere 1000 mini-batch passes, at which it attains a local minimum. At this point, the training error has not reached zero; therefore, the model is stuck in a poor local minimum. This is unusual for deep convolutional neural nets, which unlike deep fully connected neural networks are known for being easy to train i.e. not getting stuck in poor local minima [17][10].

5.2.4 Poor, Sampling-Induced Corner Minima

An intriguing aspect is that the 0.419515 validation (logprob) error that the model converges to corresponds to a 10.791% error rate, which is the proportion $\frac{12{,}287}{113{,}865}$ of Redbox images that belong to the majority class 'Clamp Detected'. Furthermore, the error rate on each batch was verified to correspond exactly the proportion of non majority class images within the batch. For example, batch 280 is the best performing batch of the entire set, obtaining 0.315% error rate at every epoch after model convergence, which is the proportion of 'No Clamp' or 'Photo Does Not Show Enough Of Clamps' images within; batch 152 is the worst performing and obtains 0.23% error rate, which is the proportion of non majority class images within.

Therefore, the reason for why the network converges relatively fast to logprob 0.4 minimum, and stays there, without ever overfitting, is that class imbalance has introduced a 'fake' minimum in a 'corner' of the error-parameter space. This location in the parameter space corresponds to the model being a constant function that always outputs $(1, 0, 0)$, where the first entry is the probability score for the majority class 'Clamp Detected'. This minimum is very hard to get out of because it is deep:

[10]Members of the Google+ Deep Learning community who were interacting on the thread were reluctant to believe that AlexNet could be stuck in a poor local minimum

it enables the model to score a 10% error rate, which might be better than a number of other local minima. It is in 'corner' in the sense that it is far away from where the 'real' minima are because the 'real' minima are in a region of the parameter space that corresponds to the network having learned visually meaningful features rather than being a constant function.

The high amplitude of the training error despite parameter convergence is therefore explained by the variation in proportion of 'Clamp Detected' images across batches.

It can also be interesting to notice that no meaningful features are learned: for example, the filters learned at the lowest convolutional layer in optimised networks usually resemble edge or contrast detectors: in this case, they are noisy, and bear no resemblance with the Gabor filters that are learned by a successfully trained CNN.

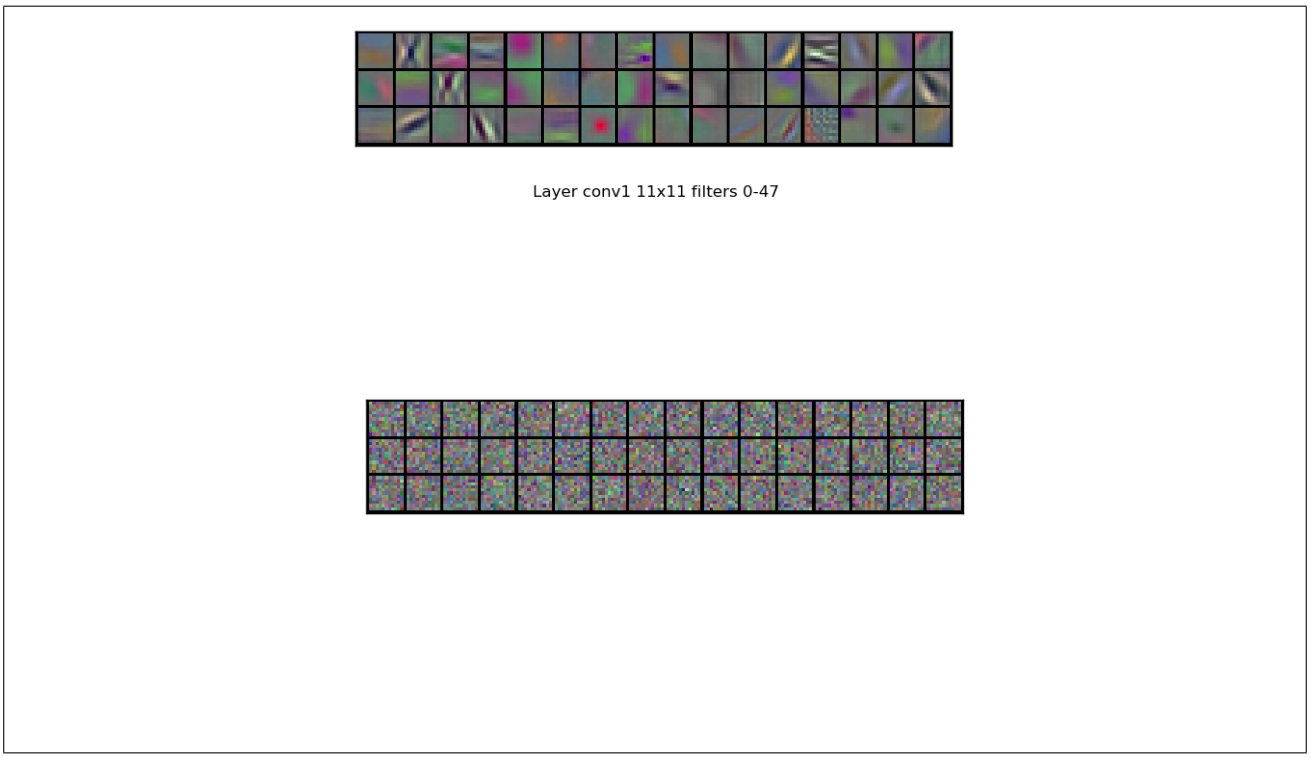


Figure 27: filters learned at lowest convolutional layer of this network

This local minimum is 'fake' because, with perfect class balance, it would not exist. Or at least, it would correspond to a $\frac{1}{K}$ error rate (where K is the number of classes). One may even call such a minimum 'lazy': it would correspond to a human not wanting to make the effort of learning, using instead a strategy of calling out the same label every time without inspecting the image, and still scoring 90% accuracy in this case. Note that accuracy refers to the percentage of correctly classified cases in the validation set.

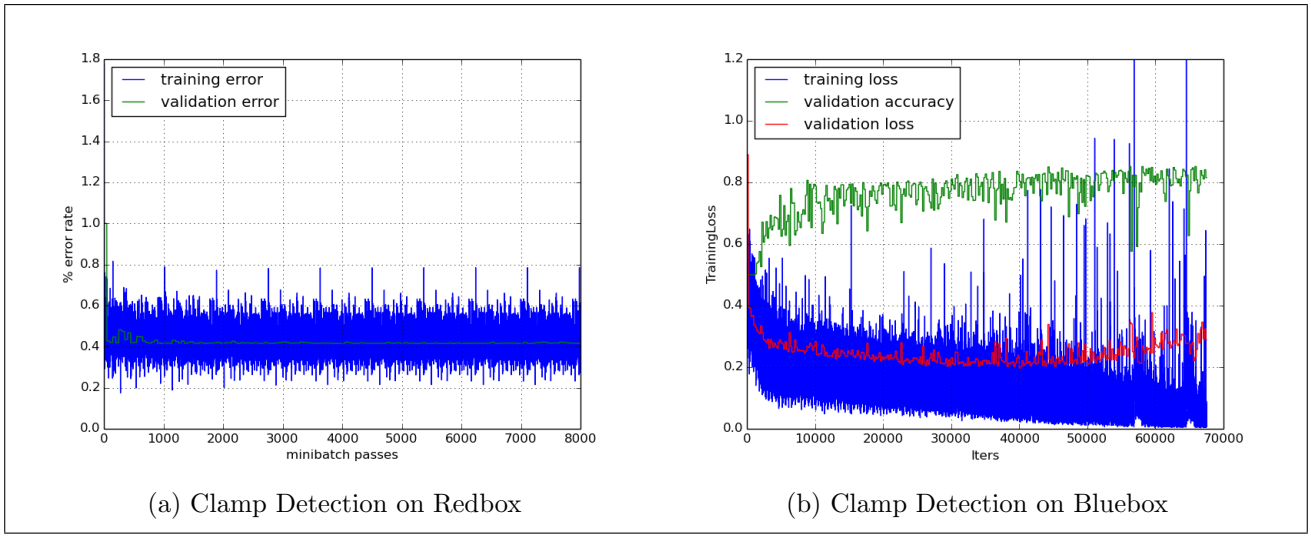
With this experiment, it becomes clear that class imbalance is a danger not sufficiently when there exists a class that is far more present than another, but sufficiently when there exists a class that takes up a vast majority of the training set. Indeed, the value of the 'fake' minimum in terms of error rate is $1 - p_K$, where p_K is the proportion of majority class cases.

The two situations are only equivalent in the 2-class case; otherwise, the latter is less likely to occur. This may shed light on why class imbalance is not a richly documented issue in the deep learning literature, since benchmark image classification datasets (such as CIFAR, MNIST and ImageNet) involve 10-1000 classes.

This experiment provides a sufficient condition for class imbalance to be dangerous; necessary and sufficient conditions are discussed in the Experiments section dedicated to class imbalance.

5.2.5 Mislabelling

It came as a surprise that, when the same clamp detection task was trained on the tenfold smaller 13,790 Bluebox dataset for which the majority class represents 88% of images, the model succeeded in learning, as the downward trend in training error and convex trend in validation error show. The achieved performance was 83.9% validation accuracy.



(a) Clamp Detection on Redbox

(b) Clamp Detection on Bluebox

A smaller training set is supposed to deliver weaker performance, because it provides the model with less information to learn from, and is more sensitive to overfit. However, after visually inspecting random samples from the Redbox training set, it was discovered that it suffered from heavy mislabelling. For example, out of 50 images randomly sampled from those Redbox images for which no flags were recorded - which are therefore supposed to be images of perfect weld installations - visual inspection performed with the help of a trained ControlPoint employee revealed that 22 of them should have received at least one flag. Three of them are given on figure 29.

Figure 29: Redbox images with no flags

As can be seen, the image to the left should have received the 'Unsuitable Photo' flag (it would have been merged into the 'Photo Does Not Show Enough of Clamps' class for training). However, since this one bears no flags, for the clamp detection task it is shown to the network as an example of a weld installation with clamps; for the scrape zone detection task, it is an example of a weld installation with visible scrape zones; etc. The middle case is almost comical: it too is shown to the network as an example of a weld installation with clamps, despite it being *literally* obvious that clamps are not present. Finally, the image to the right is zoomed in too closely for clamps to be visible.

Discussion elucidated that the beginnings of ControlPoint's pipe weld monitoring services were met with human and technical difficulties which may have resulted in data being misclassified. This would explain why learning is more successful on the Bluebox data: the Bluebox apparatus was shipped to clients later, at a stage when these difficulties had been overcome.

Deep neural networks have been reported to be robust to mislabelled data [17], but if the proportion of correctly labelled examples is lower than the proportion of majority class examples, then

bad minima are not just likely to occur, they are inevitable. Consider a hypothetical classifier with perfect understanding of the classes (e.g. a human expert from ControlPoint) which always assigns the true label to any image, and a 'lazy' classifier which always assigns the majority class label to any image. Consider also a dataset with majority class proportion k and random assignment of incorrect class labels to a proportion r of that dataset. The perfect classifier will obtain a $1 - r$ error rate, whereas the lazy classifier will obtain a k error rate. If $r > 1 - k$ then the lazy classifier will beat the perfect classifier. For clamp detection on the Redbox dataset, this is the case if more than 10.8% of images are missing a 'No Clamps Used' or a 'Photo Does Not Show Enough Of Clamps' flag. 8/50 of the randomly sampled flagless images were failed images akin to the unsuitable image shown above, and 4/50 were of the type shown in the middle or on the right. Since flagless images make up 43% of the training set, this estimate[11] suggests that approximately $0.43 \cdot \frac{8+4}{50} = 10.3\%$ of Redbox images are mislabelled with regards to clamp detection. This would also imply that none of six other, less frequent flags can be learned using the Redbox dataset.

In order to limit the impact of mislabelling, the rest of the project was conducted solely with the use of Bluebox data. This heavily reduced the amount of training data, motivating the use of transfer learning. It is worth noting that there is probably scope for making use of some of the Redbox data by exploring ways of filtering out some of mislabelled data; this is discussed in the conclusions of the report. This path was not chosen because making use of transfer learning seemed to offer more focus on deep learning techniques than on characteristics specific to the datasets.

[11] One must concede that the estimate is imprecise due to the small size of the sample.

6 Experiments 2: Tackling Class Imbalance

6.1 Definition

Before detailing the motivations for the experiments that took place, we properly return to the definition of class imbalance mentioned in the literature review and during the discussion of results on simple clamp detection. An extension to this definition is proposed to encompass *dangerous* class imbalance. Having such a definition provides a framework for reasoning about the problem.

Class imbalance has been defined as the situation where the sample distribution of classes is significantly non-uniform. For example, (Pastor-Pellicer et al 2013) [33] define it as when 'the number of patterns of one class is significantly lower than other classes'. However, consider a 12-class training set where 11 classes each take up 9% of the dataset and the 12-th takes up 1%. Outputting the same class all the time cannot provide more than 9% accuracy, which is only slightly better than 'completely random guessing', which would achieve 8.3% accuracy[12].

Class imbalance is defined in this report by the situation where **a classifier that constantly outputs the class priors performs significantly better than randomly outputting each class with the same probability**. To better understand this definition, one can take note of the property that outputting the prior classifies all input into the majority class. What follows are examples that justify this non-trivial definition.

1. Given a 2-class dataset with distribution of classes (0.9,0.1), a classifier which outputs (0.9,0.1) all the time will classify all cases into the majority class, and obtain 90% accuracy. On the other hand, randomly outputting them uniformly obtains only 50%.

2. It is not necessary for a class to occupy the vast majority of the training set: consider a 1000-class training set where one class takes up 20%. A classifier which outputs the prior all the time will obtain 20% top-1 accuracy, compared to 0.1% for random uniform.

3. It is not necessary for a class to occupy far more space than any other. Consider a 1000-class training set where 5 classes each take up 17%. Outputting the prior all the time obtains 85% top-5 accuracy, which for example is higher than the score of (Krizhevsky et al 2012) at ILSVRC 2012 [1].

It would be tempting to make the conjecture that outputting class priors is the best performing constant function classifier not just with respect to percentage accuracy, but also Maximum Likelihood Estimation (i.e. cross entropy when using a softmax layer) and Mean Square Error. However, due to time constraints there was insufficient time to attempt to prove this; it is left for further research.

With a clear definition in mind, we now return to motivations for the class imbalance related experiments that were carried out.

6.2 Motivations

The first challenge posed by the Bluebox dataset was its small size, the second was class imbalance. Although transfer learning by itself delivered strong results for clamp detection and other tasks, models failed to learn (in the sense that average per class accuracy remained at 0.5) for 9 tasks on which class imbalance was more pronounced.

Therefore, additional approaches for dealing with class imbalance were sought out. However, instead of trying them out on the unsatisfactory tasks, they were tried on a subset of the clamp detection dataset where class imbalance was increased (simply by throwing out training cases without

[12]To the best of our knowledge, in this case there is no classification strategy that would score higher than 9% without making use of information contained in the input. It would be preferable to prove it, but due to time constraints, this has been left for further research.

clamps). This way, one knows that the only thing preventing performance on the task is class imbalance, and the impact of an approach on classification performance is a good measure of the impact of the approach on class imbalance.

Were this tactic not adopted, it would not be possible to ascertain whether an approach failing to deliver a performance increase were due to its inability to tackle class imbalance, or to the fact that the task is difficult for other reasons. For example, water contamination risk is defined by the presence of droplets on the pipe fitting. However, when images are downsized to 256x256 pixels for AlexNet, droplets become invisible to the naked eye. This task could therefore be impossible to learn with networks the size of AlexNet, for a reason not related to the strong class imbalance that the task also presents (92.5%).

6.3 Implementation

Several class imbalance-tackling techniques that were experimented with required development. A Python program was written to under-sample or over-sample from a class. Two new Caffe layers were written in C++: a 'Bayesian softmax loss layer' for training a CNN on a novel cost function and a 'Bayesian per class accuracy' layer for computing validation error with the same cost function. Specific motivations and formulae for each are detailed in the experimentation subsection of this section. Following their success in this project's experiments, they were contributed to the Caffe open source project.

6.4 Experimentation

This section begins with a test run, intended to serve as a benchmark against which to subsequently evaluate six different approaches to tackling class imbalance. The first approach discussed below is transfer learning, so as to form a transition from the previous set of experiments. The second is the simplest, because it merely consists in tuning model hyperparameters. The three that follow are those explored by (Zhou and Liu 2005) [32] to specifically tackle class imbalance. The sixth is a novel cost function. A final subsection branches off from an observation concerning training results for this cost function, providing a credible explanation to the error-accuracy mismatch mentioned in the test run of the previous set of experiments.

The amount of additional class imbalance to impose on the clamp detection task was chosen in order to trap the model in a bad, sampling-induced local minimum on the Bluebox dataset. This brought the imbalance ratio from 88% to 98%, and corresponds to going from 1,323 minority class training cases to 193.

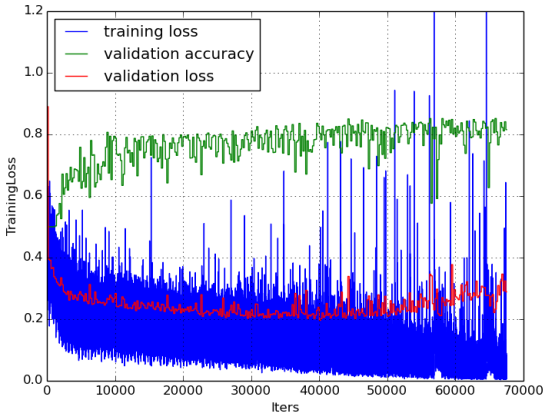
6.4.1 Test Run

To evaluate the difficulty of the clamp detection task with increased class imbalance, a model was trained from scratch without transfer learning, to be benchmarked with when it was trained on the entire dataset.

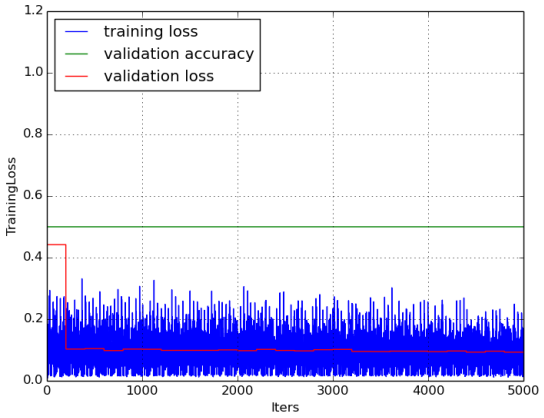
Whereas with 88% imbalance it is possible to learn without transfer learning, with 98% imbalance it is not. The model converges within one iteration to a constant function[13] and remains there.

Note that for lower levels of class imbalance, the model is able to escape the sample-induced bad minimum. The plot on figure 31 shows that with 90% class imbalance for example, the model becomes a constant function at first, and eventually is able to 'creep out' of the sample-induced minimum.

[13]Note that the reason for why the validation error seems to take 200 iterations to suddenly drop is because the validation frequency was set to a large interval of 200 training mini-batch passes, in order to speed up training.



(a) 88% class imbalance



(b) 98% class imbalance

Figure 30: no transfer learning

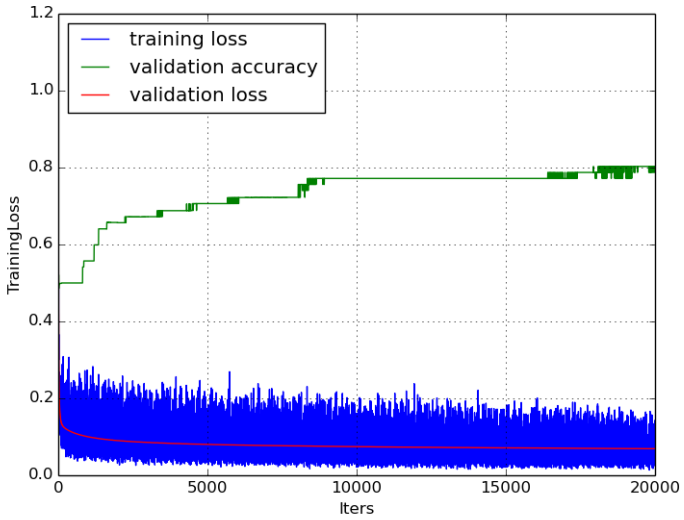


Figure 31: Clamp Detection, Full Backpropagation

This provides clues about the shape of the error surface in the neighbourhood of this bad minimum, and how varying the level of class imbalance affects it. It suggests that the bad minimum could in fact be a saddle point: a number of weights are at local optimality i.e. error-weight partial derivatives are zero, but not all; there is still room for improvement for some of them i.e. the error surface remains locally monotonous along some dimensions. This slowly but eventually moves the network away from the saddle point. Training deep neural networks along saddle points is known to exhibit this type of behaviour: training is slowed down but not stopped [17].

One would be tempted to conjecture that the steepness of the error surface along these dimensions decreases with the extent of class imbalance, and approaches 0 as class imbalance approaches 100%. This would explain why at some point class imbalance is too high for the model to escape.

Further analysis and experimentation, especially by seeing whether individual weights in the softmax layer do indeed cease to update, would perhaps make for interesting further work. In the meantime, in order to maintain generality, these points on the error surface will now be referred to as sampling-induced critical points, or simply bad critical points.

A final observation to explain is that although the validation error improves smoothly, the vali-

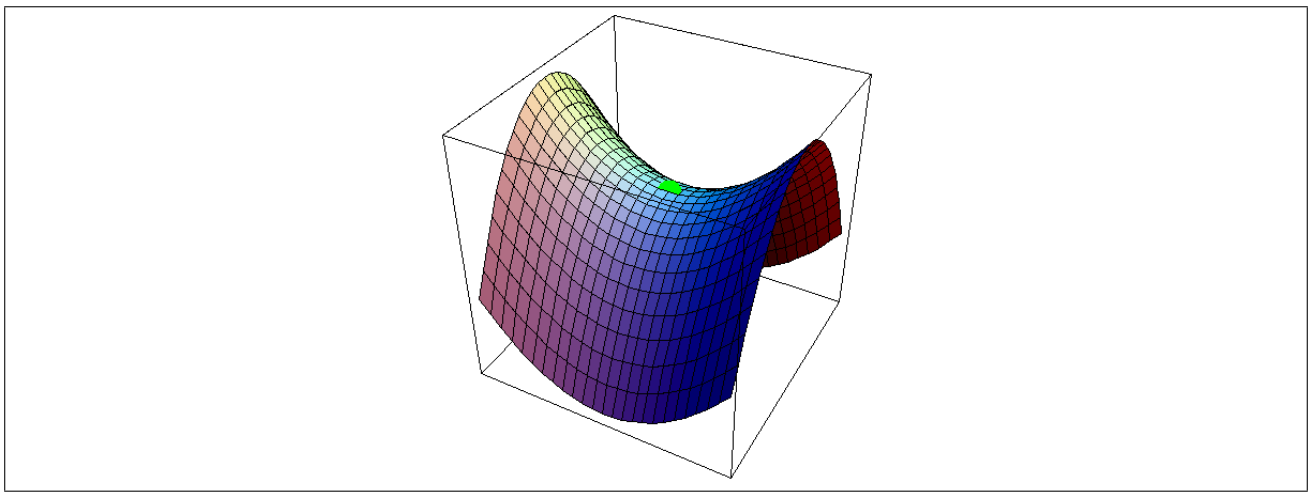


Figure 32: Example saddle point in 3D space

dation per class accuracy improves in sudden increments, i.e. discontinuously. This is likely a direct consequence of class imbalance in the validation set. Indeed, a weight adjustment that brings the predicted probability of a single minority class case in the validation set from below 0.5 to above 0.5 will decrease the validation error by $\frac{1}{n}$ and increase the validation per class accuracy by $\frac{1}{2n_{min}}$, where n is the validation set size and n_{min} is the number of minority class cases in the validation set. For sufficiently large n and sufficiently high class imbalance, the error change will be infinitesimal but the accuracy change will be discrete.

6.4.2 Transfer Learning

Transfer learning keeps the net's parameters from bad critical points, so the model is not a constant function. Notice this enables it to perfectly fit the training set. However, little is learned that generalises: maximum validation accuracy is 60.1%. Transferring the fully connected layer weights helps bring this up to 65.4%.

Note that validation error is very low because it is not a per class average: the transfer learning models quickly obtain high precision on the majority class, but struggle to learn the minority one.

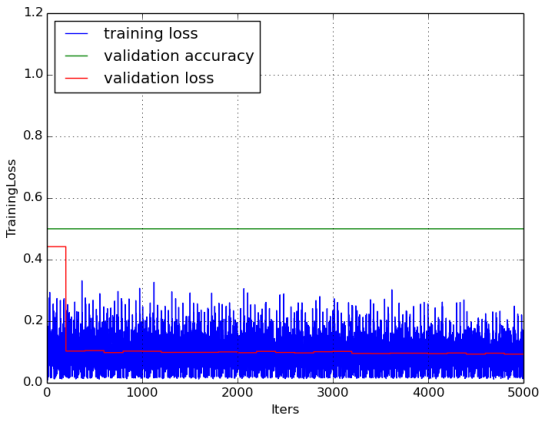
What is arguably a crucial question is whether the difficulty in learning the minority class is the inevitable consequence of having few training examples to learn from, or whether the imbalance ratio is preventing generalising features of the minority class from being learned. If the former, there is no use in searching for ways to tackle class imbalance without throwing away majority class data.

Following the evaluation of transfer learning, a fixed architecture and training strategy were kept throughout the experiments below: backpropagation on all layers, all weights transferred. Although freezing backpropagation on the first four convolutional layers was found to perform best during experiments of the previous section, full backpropagation was adopted in this section for the sake of clarity. The objective of this section is not to achieve the highest absolute accuracy, but rather to compare relative successes of different approaches to tackling class imbalance.

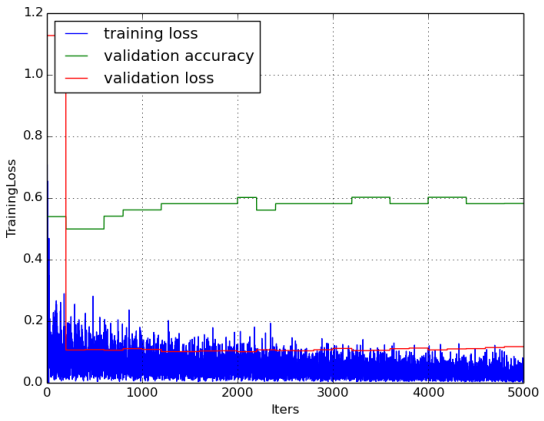
6.4.3 Hyperparameter Optimisation

Mini-batch Size Since class imbalance reduces the number of minority class cases, it could arguable make information relating to minority class misclassification contained in the gradient more noisy. If so, increasing the mini-batch size should improve performance.

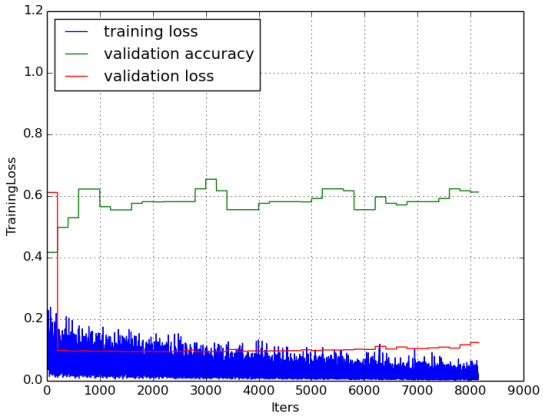
These results are surprising: they seem to suggest that increasing mini-batch size decreases performance. Since increasing mini-batch size only reduces noise in the gradient, this would attribute performance increase to greater noise in the gradient.



(a) no transfer

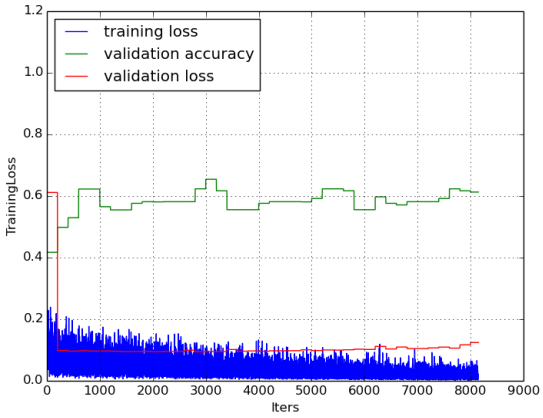


(b) convolutional layer transfer

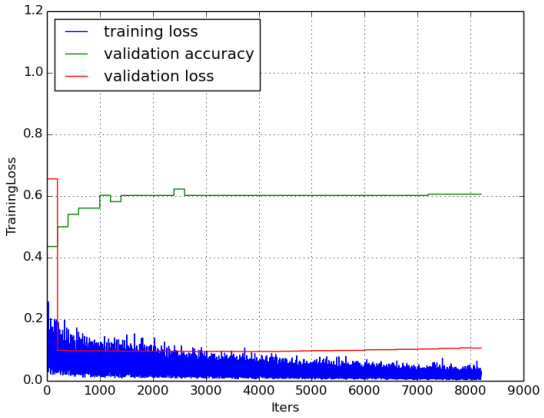


(c) full net transfer

Figure 33: 98% imbalance, varying levels of transfer learning



(a) mini-batch size 128



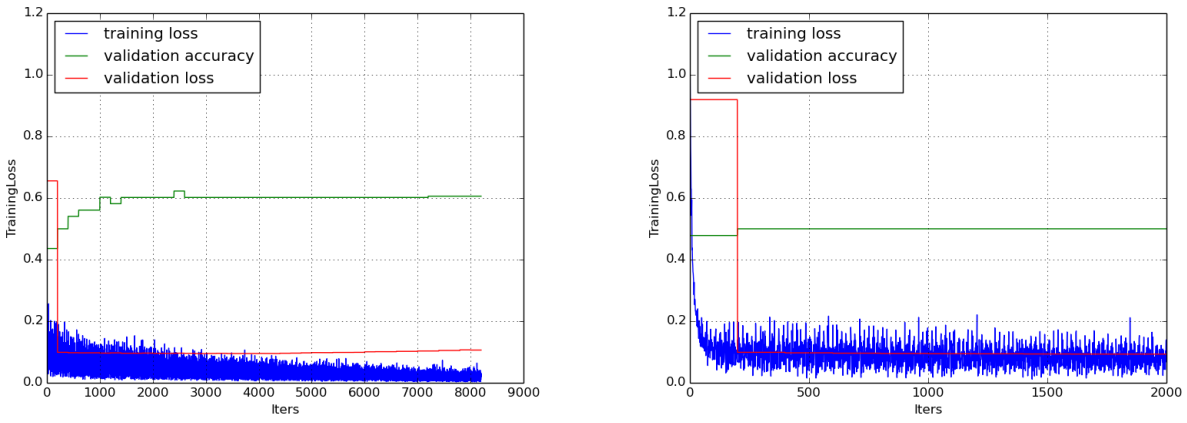
(b) mini-batch size 256

Figure 34: 98% imbalance, different mini-batch sizes

However, one notices that from the 1000th iteration onwards, in the 128 mini-batch size network, the time series of the per class validation accuracy (and if one looks closely, of the validation error too) is not convex. In fact, it appears to be random. The fact that per class validation accuracy will increase discontinuously for a single additional correct minority class classification in the validation set supports this claim. Indeed, random weight changes are bound to lead to small increases and decreases in the number of correct minority class classifications. The plot shows indeed as many increases as decreases. These seem substantial but this is only the result of per class accuracy being heavily weighted by minority class cases. If improvements are indeed random, they will not generalise to the test set.

A foolproof verification of this explanation would be to run both models on a test set, and see whether there is indeed no significant difference in true performance between the two models.

Learning Rate Decreasing the learning rate helps if it prevents a good minimum from being overshot.



(a) learning rate 0.0001

(b) learning rate 0.00001

Figure 35: 98% imbalance, different learning rates

With a lower learning rate, the validation accuracy seems to increase at a far slower pace. This suggests that there was no overshooting when using the original learning rate value, and that 0.0001 is a good learning rate for this task.

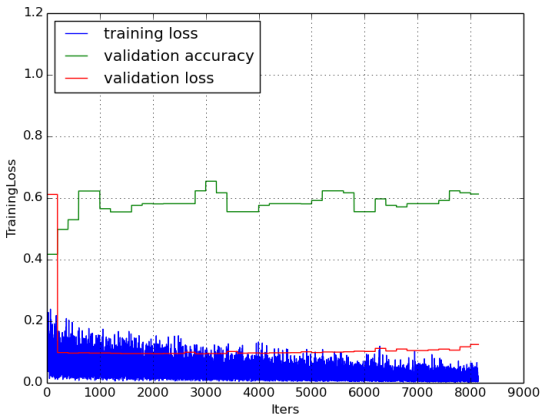
6.4.4 Over-Sampling

Over-sampling [32] [31] rebalances the training set to desired levels of imbalance by increasing the occurrence of minority class examples. This was implemented by creating duplicates in the dataset.

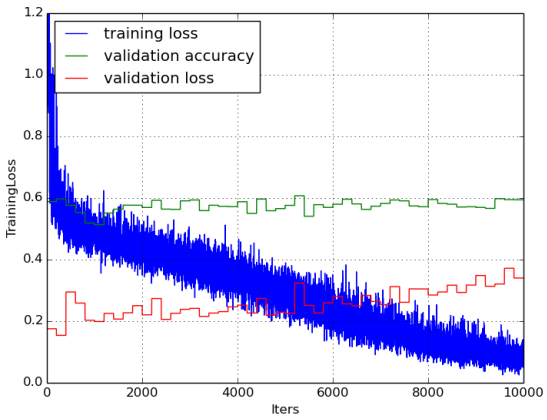
Oversampling does not produce good results. Even though the training error converges steadily to zero i.e. the model is able to fit the data, overfit occurs very quickly. This early overfit is consistent with the findings of [32] and [31], and can be explained by the fact that oversampled examples are seen highly frequently.

6.4.5 Under-Sampling

Under-sampling rebalances the training set to desired levels of imbalance by removing majority class examples. The brute force approach, implemented in (Maloof 2003) [31], consists in randomly removing these cases and train on the remaining set. However, (Zhou and Liu 2005) [32] describe an



(a) 98% imbalance, full backprop, full transfer



(b) Same with over-sampling to 88% imbalance

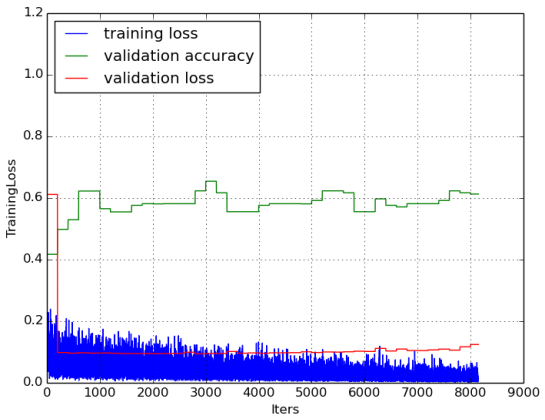
algorithm for removing selectively. It was not implemented during this project mainly due to its time complexity, which is analysed below.

Under-Sampling algorithm

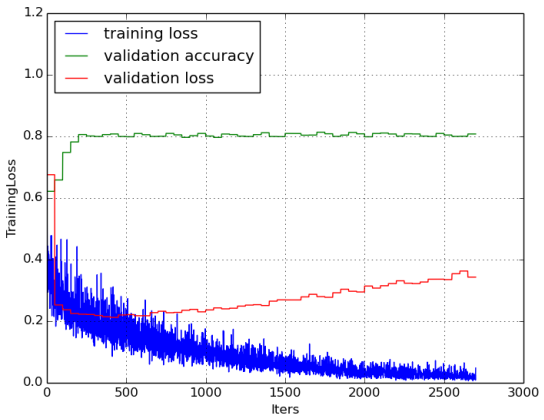
1. Choose desired imbalance ratios for each non-minority class.
2. Reach desired imbalance levels in training set by randomly setting aside non-minority class examples.
3. Train a net.
4. While there remains examples set aside:
   (a) Increase training set by bringing in 50% additional non minority class examples.
   (b) Don't train yet, but classify all new arrivals. Remove those correctly classified.
   (c) If training set remains unbalanced, remove non minority class examples whose nearest neighbour in feature space is of a different class.
   (d) If training set remains unbalanced, randomly remove non minority class examples.
   (e) Train the net on the training set.

Notice that the removal of majority class examples is selective based first on a 'redundant' criterion, then on a 'potentially noisy' criterion. The underlying assumption for the first criterion is that examples which are already classified will not contribute to learning. However, if 'redundant' is taken formally as stating that back-propagating the gradient from these examples will not modify the weights, then it assumes that the error on them is zero, i.e. that the output probabilities for these examples are binary vectors. This is highly unlikely to be the case in practice, so the classification test for rejection does not identify truly redundant examples.

The second criterion assumes that if an example's nearest neighbour is of a different class, then it is better for learning to discard this 'boundary example' because it is likely to be highly noisy. The impact of this selection criterion on learning is based on euclidean distance, it is therefore non-parametric and is difficult to intuit in the context of deep neural networks with softmax output, which are parametric models. On the other hand, the criterion can be intuited in the context of SVM learning. If we were training a Support Vector Machine, i.e. finding the decision boundary that leaves most gap between support vectors of different classes, then discarding non-minority class 'boundary examples' would have two effects: extend the minority class 'territory' into the 'territory' of non minority classes, and reduce the error on fitting linear boundaries to the training data. The implicit



(a) 98% imbalance, full backprop, full transfer



(b) Same with under-sampling to 88% imbalance

assumption of linear separability and the use of a non-parametric criterion in a parametric model may be considered difficult to justify.

To the best of our understanding, it is unclear from the paper whether training in 3(e) uses randomly initialised weights, or whether they are transferred from the previous model. From our experiments on transfer learning, the former would train significantly faster. However, it would also mean that examples which made it by luck early into the training set will be over-represented throughout overall training, which will increase overfit. Given that the authors reproach this over-representation in the case of over-sampling but not under-sampling, and that no mention of transfer learning is ever made [32], the latter is assumed.

Training a net has time complexity $T_{backprop} = \Theta(m(T_{forward\_pass} + T_{backward\_pass})) = O(m \cdot T_{backward\_pass})$, where m is the number of mini-batch iterations. We make the trivial assumption that 3a occurs in constant time. Classifying k examples is a single forward pass with mini-batch size k. Thanks to the empirically verified [30] assumption that $T_{forward\_pass} = O(T_{backward\_pass})$ holds none-tightly, we can comfortably say that 3b has asymptotic time complexity $O(T_{backprop})$. Computing the distance between two inputs in feature space amounts to two forward passes until the penultimate layer – because the coordinates of an input in feature space is the output of the penultimate layer) – followed by a constant number of vector operations between the two outputs to compute euclidean distance, which we assume to be computable in constant time. Therefore, 3c has time complexity $\Theta(T_{backward\_pass}) = O(T_{backprop})$. We trivially assume that 3d occurs in constant time. 3e is backpropagation, so its time complexity is $T_{backprop}$.

We now have that the body of the while loop has time complexity $T_{backprop}$. The body of the while loop will execute $log_{1.5}(\frac{n_1}{n_0})$ times, where n_1 is the size of the most heavily reduced class and n_0 its size after reduction. The 1.5 is because the number of new examples to examine increases by 50% at every iteration, so $log_{1.5}(\frac{n_1}{n_0})$ is the number of iterations required to get from the reduced set back to the original set with a 50% growth rate.

This algorithm therefore has time complexity $T_{backprop} \cdot log_{1.5}(\frac{n_1}{n_0})$. So in the case of clamp detection with 98% class imbalance, training with under-sampling at 88% target imbalance – which was the imbalance ratio at which clamp detection was successfully learned in previous sections – would have taken 5 times longer than training with simple brute force under-sampling. Considering on top of this the time that would have been spent in implementation, it was deemed impractical, especially since the authors report better performance with threshold.

Instead, brute force under-sampling was implemented as in (Maloof 2003) [31]. Note that in order to do so, minority class examples from the clamp detection dataset were first deleted in order to achieve 98% imbalance, followed by deletion of majority class examples in the training set to bring the dataset back to 88% imbalance. Results are shown on figure **??**.

The results show that under-sampling is very effective: the model does not get stuck in any seeming critical point, obtains an additional 15 percentage points in classification accuracy,to obtain 80.8% validation accuracy. This is substantial and shows how strikingly detrimental class imbalance is: we have eliminated 58.7% of the data to get rid of it, and yet this sacrifice is highly worthwhile. This sets a high benchmark for other techniques for tackling class imbalance. Since we are throwing data, it also implies that if there exists a technique for solving class imbalance without losing data, performance could be even higher. The assumption that underlies this is that providing the model with additional information i.e. data is in theory purely beneficial, therefore there must exist a way for doing so.

6.4.6 Threshold-Moving

(Maloof 2003) and (Zhou and Liu 2005) both find threshold-moving to be the best technique for tackling class imbalance. It allows one to set different costs to mis-classifications on different classes; in the 2-class case, this means that we can assign different costs to false positives than to false negatives. As with detecting the presence of a pathogen in a patient in the medical industry [34], one could argue that a false negative is more expensive to ControlPoint: it is worse to classify a bad joint as good than to classify a good joint as bad, since in the latter case, no potential damage is done to the installation. When the idea of assigning different costs was suggested to ControlPoint, the reply was positive.

Motivations Threshold-moving is a technique that operates on the model only at test time. It is implemented by taking the predictions of the model at test time, dividing them by the prior probabilities of the classes i.e. their proportions in the training set, and renormalising. This may remind one of Bayes' theorem; indeed, threshold-moving has an elegant probabilistic interpretation. Consider a model trained on a 2-class task with assignment of equal cost to both mis-classifications, and its output on an example c from the test set. We would want the model to predict the label $arg \max_{l \in \{l_1, l_2\}} \{p(c|l)\}$. However, the model will output the probabilities $(p(l_1|c), p(l_2|c)) = (p(c|l_1)p(l_1), p(c|l_2)p(l_2)$. The model will output exactly what we want when classes are distributed uniformly in the training set, since the $p(l_i)$ will all be the same. This fails if the priors are different, and can be corrected simply by exploiting Bayes' theorem: dividing the output by the prior and renormalising. The reason for why this is called threshold-moving can be explained as follows: assume the prior probability for class C_1 is $p(C_1) = 0.1$. The threshold for classifying in input as C_1 given its predicted probability for belonging to C_1 moves from 0.5 to 0.1.

Note that the mismatch between desired and outputted probabilities, exposed by this Bayesian conceptual framework, occurs as soon as classes are not uniformly distributed, which corresponds to the literature's definition of class imbalance rather than the definition of 'dangerous' class imbalance proposed by this report. This has one of two implications: either – as the difference in definitions suggest – probability mismatch and introduction of bad critical points in the error surface are two different problems, or they are the same one, and the problems observed with class imbalance until now have been wrongly interpreted, wrongly explained. This can be verified by seeing whether threshold-moving solves this hypothetical bad critical point problem, i.e. if it stops the model from outputting the same classification for all inputs at test time.

Implementation Threshold-moving can be relatively easily implemented by writing a Python script that runs a trained model on a test set, obtains the predicted probabilities, and classifies according to the modified threshold, given by the prior probabilities of the classes. Also, if we choose to make the threshold flexible rather than determined by the prior probabilities, then we have a way to increase the relative cost of a false negative at will. This can be used to set a threshold that guarantees, say, 95% accuracy on true positives, with a consequentially lower accuracy on true negatives depending on the overall performance of the model. Note that practically, this would enable the automation of

a portion of the work without any increased risk of letting false positives slip through.

Results The python threshold-moving script was run on a model trained on the 98% imbalance clamp detection task with no transfer learning, and another with transfer learning (full transfer of weights, backpropagation enabled on all layers).

Model	Per Class Accuracy	Automation
No transfer learning	50%	0%
No transfer learning, threshold	50%	0%
Transfer learning	59.3%	17.0%
Transfer learning, threshold	64.7%	19.6%

Table 2: Impact of threshold-moving

Since the threshold does not improve per class accuracy in the no transfer learning case, solving the probability mismatch does not solve the bad critical point problem; the two problems are therefore distinct. Indeed, inspection of the output probabilities for the no transfer learning model reveals that every output is the binary vector (0,1), where the 2nd entry corresponds to classification probability for the majority class. Therefore, no amount of threshold moving can prevent such a model from being a constant function. This suggests that the literature on class imbalance has until now not documented the occurrence of bad critical points when class imbalance is dangerous, since it has only focused on the probability mismatch.

In the context of trying different approaches to tackle class imbalance, threshold-moving was however found to improve average class imbalance, since it provides 5.4 additional percentage points in classification accuracy for the transfer learning model. This suggests that threshold-moving helps as long as the model is not stuck in a bad critical point.

6.4.7 Bayesian Cross Entropy Cost Function

Motivations It was wondered whether the best performing techniques discovered until now could also be combined with a modified cost function. Indeed, threshold re-normalisation does not change the fact that most of the errors computed throughout a SGD mini-batch pass come from majority class examples. Since the stochastic gradient is an average across the mini-batch, it carries more 'teaching material' for learning the majority class than the minority class. If we want the stochastic gradient to reflect as much of the C_1 errors as the C_2 errors on average, then perhaps we need to renormalise the errors based on class priors as well. This would result in modifying the cross entropy cost function from:

$$-\frac{1}{n}\sum_{i=1}^{n} log(f(x_i, l_i, W)) \tag{16}$$

where $f(x_i, l_i, W)$ is the model's predicted probability for input x_i's label to be l_i, W are its parameters, l_i is the true label for x_i, n is the mini-batch size, to:

$$-\frac{1}{K \cdot n}\sum_{i=1}^{n} \frac{log(f(x_i, l_i, W))}{p(class(x_i))} \tag{17}$$

where K is the number of classes in the classification task. There is a parametric justification for this cost function, which can be intuited when the cross entropy cost function is seen as a formula for Maximum Likelihood Estimation. This latter aspect is first shown.

As explained in the literature review, placing a softmax layer as the output layer of a deep neural network forces it to output a probability distribution for the label of the input. This makes deep

neural networks probabilistic inference models, since the weights are now parameters that define a probability distribution conditional on the training data for the discrete random variable that takes a value for each class. Each input defines a different random variable, and the model uses its parameters to output what it thinks is that random variable's probability distribution, conditional on the events it has been trained on and the information it has about the input.

The idea behind Maximum Likelihood Estimation is that the best parameters are the ones that make the model output that probability distribution for each training example's random variable which assigns as high a probability as possible to the training example's outcome. In the case of clamp detection, the training example is an image x_i, its random variable is the label L_i, L_i is a random variable defined over the event space $\Omega(L_i) = \{$'Clamp Detected', 'No Clamps'$\}$, and the training example's outcome is the value of the label l_i. The maximum likelihood estimates for the parameters of the model are therefore found by solving:

$$arg \max_{W \in \mathbb{R}^D} P\{ \bigcap_{1 \leq i \leq n} [L_i = l_i] | x_i, W\} = arg \max_{W \in \mathbb{R}^D} \prod_{1 \leq i \leq n} P\{[L_i = l_i] | x_i, W\}$$

assuming that the X_i are independently and identically distributed;

$$= arg \max_{W \in \mathbb{R}^D} \log(\prod_{1 \leq i \leq n} P\{[L_i = l_i] | x_i, W\})$$

since log is a monotonous strictly increasing function, the coordinates of W at which max is reached are the same;

$$= arg \min_{W \in \mathbb{R}^D} - \log(\prod_{1 \leq i \leq n} P\{[L_i = l_i] | x_i, W\})$$

since taking the opposite function turns maximum into minimum;

$$= arg \min_{W \in \mathbb{R}^D} - \sum_{1 \leq i \leq n} \log(P\{[L_i = l_i] | x_i, W\})$$

$$= arg \min_{W \in \mathbb{R}^D} -\frac{1}{n} \sum_{1 \leq i \leq n} \log(P\{[L_i = l_i] | x_i, W\})$$

since scaling the function by a constant preserves extrema;

$$= arg \min_{W \in \mathbb{R}^D} -\frac{1}{n} \sum_{1 \leq i \leq n} \sum_{l \in \Omega(L_i)} P\{L_i = l\} \cdot \log(P\{[L_i = l] | x_i, W\})$$

since $P\{L_i = l\} = 1$ if $l = l_i$ and 0 otherwise;

$$= arg \min_{W \in \mathbb{R}^D} -\frac{1}{n} \sum_{1 \leq i \leq n} KL(P\{L_i\}, P\{L_i | x_i, W\})$$

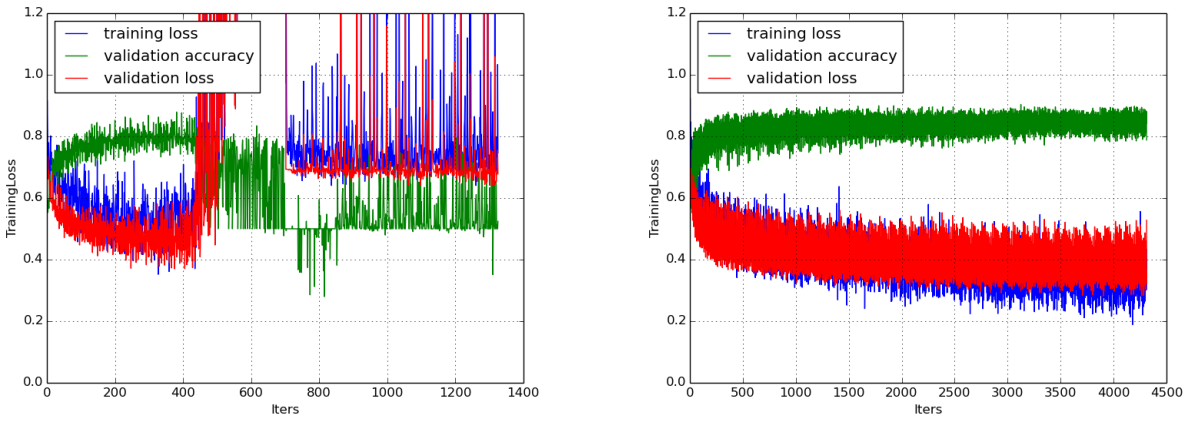
where KL is the Kullback-Leibler divergence;

$$= arg \min_{W \in \mathbb{R}^D} -\frac{1}{n} \sum_{1 \leq i \leq n} CE(P\{L_i\}, P\{L_i | x_i, W\})$$

where CE is the cross entropy;

$$= arg \min_{W \in \mathbb{R}^D} -\frac{1}{n} \sum_{1 \leq i \leq n} CE(P\{L_i\}, f(x_i, l_i, W))$$

$$= arg \min_{W \in \mathbb{R}^D} -\frac{1}{n} \sum_{i=1}^{n} log(f(x_i, l_i, W))$$



(a) Incorrect gradient propagation

(b) Correct gradient propagation

Figure 38: Ground Sheet detection, different backpropagation formulae

This optimisation problem is, as we know, tentatively solved by local optimisation with gradient descent.

Therefore, when viewed as MLE estimation, the Bayesian cross entropy can be interpreted as follows: instead of trying to maximise the joint probability of the data under the assumption that each observed event has the same probability, we are assigning higher probability to minority class events. In a way, we are supposing that the 'identically distributed' assumption is violated.

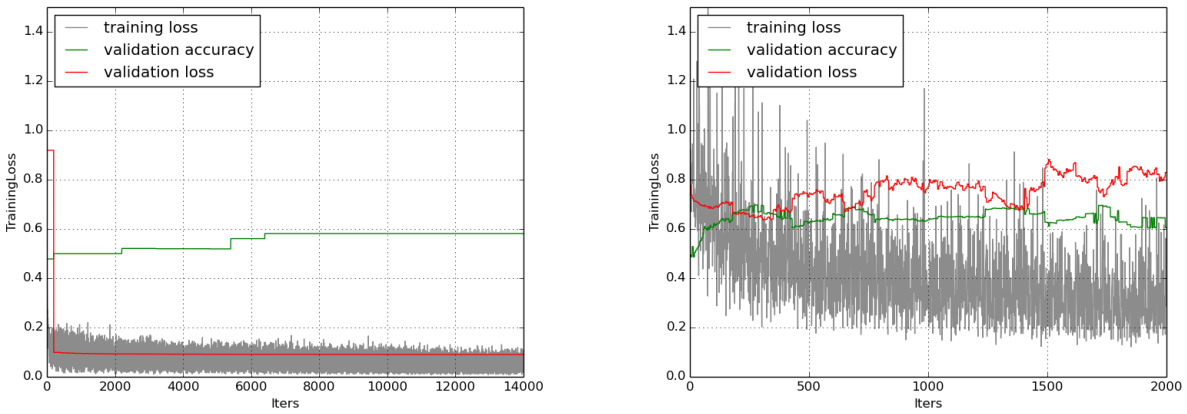
The Bayesian cross entropy has several advantages. The key one is that it should rebalance the gradient without requiring us to throw away any data or slow down training. Under-sampling throws away data, over-sampling slows down training, threshold does not rebalance the gradient. It can be combined with transfer learning and has a probabilistic interpretation in line with the Maximum Likelihood Estimation framework on which softmax deep neural networks are trained.

Implementation The Caffe framework does not separate output layer from cost function, so a softmax bayesian loss layer and a per class bayesian accuracy layer were written, in C++. The layer computes the prior class probabilities within the mini-batch so as to be more precise than by taking the priors over the training set. The implementation of backpropagation within the softmax layer also had to be updated since the Jacobian entries all had to be divided by the prior for the true label of the training case. One of the difficulties encountered during development was that if a class were absent from a single mini-batch, the prior for it would be zero, loss computed on it infinite, and weight updates would diverge.

For testing and debugging, the cost function was used on one of the simpler tasks with little imbalance: ground sheet detection. Once can notice that when the gradient is not correctly propagated, backpropagation does not converge. The plot on the left shows training results when only the softmax layer's Jacobian's diagonal entries are renormalised by the prior.

Results Full transfer learning, backpropagation on all layers, batch size 256.

With Bayesian cross entropy, training is far less clean, but the highest per class classification accuracy reached is nearly 10 percentage points higher. The training is less clean because the modulus of the gradient is strongly influenced by imbalance within the mini-batch, and this value can greatly vary from a mini batch to the next, since a mini batch with 3/256 minority class cases has minority class proportion 3 times that of a mini batch with 1/256 minority class cases, and both are likely



(a) Cross entropy

(b) Bayesian cross entropy

Figure 39: 98% imbalance, different cost functions

to occur in sampling. This could be improved by making sure every mini-batch has the same class proportions as the training set. This would also save us from having to compute the within-batch prior at every iteration, and deliver a slight speed increase.

One may get the impression that Bayesian cross entropy fails because the models trained on it do not hit as low a validation error minimum as training the same models trained on normal cross entropy. Firstly, such an interpretation is wrong: comparing two cost functions in absolute terms makes no sense; their relative impact on the performance of a model can only be assessed by measuring performance on both models by the same metric, such as per class accuracy. In this regard, for the clamp detection task with 98% imbalance, Bayesian cross entropy scores nearly 10 percentage points higher.

Nevertheless, because cross entropy and Bayesian cross entropy have similar formulae, it is still possible to explain why minimum validation cross entropy is below minimum validation Bayesian cross entropy. Scarcity of minority class cases makes this class harder to learn, so by using tricks to get the net to 'focus on it more', one is telling the net to forego opportunities to learn majority class patterns that have higher return across the training set, in favour of minority class patterns. Since Bayesian cross entropy is a kind of average per class cross entropy, it will almost *always* be higher in absolute terms than cross entropy when there is class imbalance, since the minority class will almost always be harder to learn due to scarcer information. This will occur even if the Bayesian cross entropy achieves in getting the net to learn the minority class significantly better, and per class accuracy improves!

As is often said in machine learning, there is no free lunch: if one makes minority class cases stand out disproportionately, one is telling the net 'never mind per case performance, maximise per class performance: seize opportunities to improve on the minority class even if it's at the smaller expense of performance on the majority class'. When taking a measure of performance weighted by class, this is beneficial; when taking a measure of performance weighted by case in an unbalanced dataset, such as cross entropy, this is detrimental. It all depends on what we wish to optimise. In the case of ControlPoint, per class accuracy is preferred.

Why does SBL perform better than threshold-moving? A potential explanation would be that, by acting only at test time, threshold-moving does not interfere with the mechanics of learning. This is good if one believes these mechanics to be set up optimally. However, we would argue from the experiments carried out in this project that when there is dangerous class imbalance, standard mechanics of learning are suboptimal because bad critical points are introduced on the error surface, and need to be removed. SBL is designed for this; the superior performance results support the claim that it achieves this.

Further Research It would be interesting to mathematically derive how imbalance affects the modulus of the gradient, and how the standard deviation of the modulus with with Bayesian cross entropy is much greater than with cross entropy. As mentioned, the variance of the gradient could also be reduced by making sure every mini-batch has the same class proportions as the training set. This would also save us from having to compute the within-batch prior at every iteration, and deliver a slight speed increase.

It would also be interesting to confirm that Bayesian cross entropy achieves something different to threshold-moving. This could be achieved with the threshold-moving Python script, which contains code that, given a model and target ratio of false positives to total positives, returns the proportion of predicted negatives, when the threshold level that is required to attain this target ratio is used. The extent to which this proportion is greater for the model trained with Bayesian cross entropy than for the same model trained with cross entropy would be a measure of the additional semantic content learned.

Overall, these results are encouraging and further experiments with benchmark multi-class datasets such as MNIST, CIFAR and ImageNet would allow to situate the performance gains from using SBL in a wider academic context.

6.4.8 Error-accuracy mismatch revisited

A surprising observation made on all previous training results was that per class accuracy tends to continue improving long after the validation error begins to deteriorate. A proposed explanation was that, during this period, the model increases its precision on the minority class at the slight expense of precision on the majority class. It was based on the mismatch between averaging the performance metric across the dataset and averaging across classes. In the training results for Bayesian cross entropy, the validation error is also averaged per class, and indeed, error and accuracy appear strongly negatively correlated. This is substantial evidence in favour of this explanation.

However, what remains a curiosity are the implications that this explanation has for the dynamics of learning: it would imply that when using cross entropy error, learning 'focuses' on the majority class first, and afterwards focuses on learning the minority class. This makes 'strategic' sense, since there are more gains from improving on the majority class when using an error that is averaged across the min-batch.

SGD mechanics One may wonder how this could translate from the mechanics of Stochastic Gradient Descent. In the two class case, for each softmax neuron, the gradient propagating through it in a class A instance is always of opposite sign to that from a class B instance. Therefore, when backpropagating a gradient that has been averaged over a mini-batch with class imbalance, assuming that the average per-case error on the majority class is sufficiently not smaller than on the minority class, then this averaged gradient will be of the same sign as a majority class gradient. Therefore, the weight corrections for the softmax layer will be in the direction that favours learning the majority class over the minority class. Assuming that class imbalance is present in all mini-batches, this will occur until the per-case error on the majority class is sufficiently reduced as to change the sign of the averaged 'stochastic' gradient, and invert the directions of learning in the softmax layer in favour of the minority class[14].

All of the above holds only for the softmax output layer and not for lower layers, since as shown in the section on the ReLU, the sign of a gradient propagating from one neuron to another will change if the edge it is propagating through has negative weight. Therefore, the above cannot explain how the entire model learns. Nevertheless, it provides some analytical grounding for what is being observed. Observing and analysing these mechanisms further and with greater rigour would perhaps make for interesting further research.

[14] All these affirmations can be shown analytically, but due to time constraints were not written up in this report.

7 Conclusions and Future Work

The third stage of the empirical research focused on formalising class imbalance, understanding how it impacts learning, and seeking ways to deal with it. Despite its dire effects, the topic is significantly undocumented in the field of deep learning, so the research was an opportunity to make new insights and discoveries. It seems that when the distribution of classes in the training set are such that a classifier which constantly outputs the class priors can obtain high performance, the error surface associated with standard cost functions is punctured with bad sampling-induced critical points, which slow down and often even prevent learning with stochastic gradient descent. After exploring simple techniques as well as some of those advocated by the literature, a novel cost function was proposed. Its ability to tackle class imbalance was found to be substantial, delivering up to 28 additional percentage points in average per class validation accuracy in imbalanced tasks. The performance gains from this Bayesian cross entropy are also likely to be increased by ensuring the same distribution of classes in every mini-batch of the training set, since this would reduce noisiness a.k.a variance in the modulus of the gradient. Further empirical research would also need to be conducted by trying out this cost function on benchmark datasets with multiple classes and varying levels of class imbalance.

References

[1] Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E.; *ImageNet Classification with Deep Convolutional Neural Networks*
2012

[2] Glorot, Xavier; Bordes, Antoine; Bengio, Yoshua; *Deep Sparse Rectifier Neural Networks*
2013

[3] Lowe, David *Object Recognition From Local Scale-Invariant Features*
The Proceedings of the Seventh IEEE International Conference on Computer Vision 1999

[4] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks*
ILSVRC 2013

[5] Tang, Yichuan; *Deep Learning using linear Support Vector Machines*
arXiv 2014

[6] Nair, Vinod; Hinton, Geoffrey; *Rectified Linear Units Improve Restricted Boltzmann Machines*
ICML 2010

[7] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov; *Improving Neural Networks By Preventing Co-Adaptation of Feature Detectors*
arXiv 2012

[8] Dan C. Ciresan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, Juergen Schmidhuber; *High-Performance Neural Networks for Visual Object Classification*
arXiv 2011

[9] Donahue, Jeff; Jia, Yangqing; Vinyals, Oriol; Hoffman, Judy; Zhang, Ning; Tzeng, Eric; Darrell, Trevor; *DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition*
arXiv preprint arXiv:1310.1531, 2013

[10] Fergus, Robert; *Tutorials Session A - Deep Learning for Computer Vision*
NIPS 2013

[11] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, Stefan Carlsson; *CNN Features off-the-shelf: an Astounding Baseline for Recognition*
arXiv 2014

[12] Olah, Christopher; *Understanding Convolutions*
URL: `http://colah.github.io/posts/2014-07-Understanding-Convolutions/`, last accessed 1st September 2014.

[13] Zhou, Zhi-Hua; Zhang, Min-Ling; *Multi-Instance Multi-Label Learning with Application to Scene Classification*
Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006

[14] Fusion Group - ControlPoint LLP, *Company Description*
URL: `http://www.fusionprovida.com/companies/control-point`
last accessed 5th June 2014.

[15] LeCun, Yann; *The Unreasonable Effectiveness of Deep Learning*
Journee du Labez Bexout 12 June 2014 URL: `https://www.youtube.com/watch?v=zPVHH7ZJi9Q&list=PLYTk9IwHBb2ANKHc2feIso7tL1slpuVPY&index=1`
last accessed 4th September 2014/

[16] Barron, Andrew R., *Universal Approximation Bounds for Superpositions of a Sigmoidal Function*
IEEE Transactions on Information Theory, Vol. 39, No. 3 May 1993

[17] Bengio, Yoshua; *Learning Deep Architectures for AI*
Foundations and Trends in Machine Learning Vol. 2, No. 1 (2009) 1-127 2009

[18] Bishop, Christopher; *Pattern Recognition and Machine Learning*
Springer 2010

[19] Zeiler, Matthew; Fergus, Robert; Visualizing and Understanding Convolutional Networks
arXiv 2013

[20] Russell, Stuart J; Norvig, Peter; *Artificial Intelligence: A Modern Approach*
2003

[21] Hornik, Kur; Stinchcombe, Maxwell; White, Halber; *Multilayer Feed-Forward Networks are Universal Approximators*
1989

[22] Saenko, K., Kulis, B., Fritz, M., and Darrell, T.; *Adapting visual category models to new domains*
ECCV, 2010

[23] Bengio, Yoshua; *Google+ Deep Learning community post*
URL: `https://plus.google.com/+YoshuaBengio/posts/GJY53aahqS8`, last accessed 1st September 2014.

[24] Bay, H., Tuytelaars, T., and Gool, L. Van; *SURF: Speeded up robust features*
ECCV, 2006

[25] Sermanet, Pierre; Eigen, David; Zhang, Xiang; Mathieu, Michael; Fergus, Rob; LeCun, Yann; *OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks*
arXiv:1312.6229

[26] Krizhevsky, Alex; *Cuda-ConvNet: High-performance C++/CUDA implementation of convolutional neural networks*
URL: `https://code.google.com/p/cuda-convnet/`, last accessed 6th June 2014.

[27] Deng, Li; Yu, Dong; *Deep Learning Methods and Applications*
Foundations and Trends in Signal Processing 7:3-4 2014

[28] Jia, Yangqing; *Caffe: a deep learning framework* URL: `http://caffe.berkeleyvision.org/getting_pretrained_models.html`
last accessed 6th June 2014.

[29] Fergus, Robert; *NIPS 2013 Tutorial - Deep Learning for Computer Vision*, URL: `http://research.microsoft.com/apps/video/default.aspx?id=206976&l=i`
last accessed 6th August 2014.

[30] URL: `https://github.com/soumith/convnet-benchmarks`
last accessed 2nd September 2014.

[31] Maloof, Marcus A; *Learning when data sets are imbalanced and when costs are unequal and unknown*
Workshop on Learning from Imbalanced Data Sets II, ICML 2003

[32] Zhi-Hua, Zhou; Xu-Ying, Liu; *Training Cost-Sensitive Neural Networks with Methods Addressing the Class Imbalance Problem*
IEEE Transactions on Knowledge and Data Engineering, 2005

[33] Joan Pastor-Pellicer, Francisco Zamora-Martnez, Salvador Espaa-Boquera, Mara Jos Castro-Bleda; *F-Measure as the Error Function to Train Neural Networks*
Advances in Computational Intelligence Volume 7902, 2013, pp 376-384

[34] He, Haibo; *Learning from imbalanced data*
IEEE Transactions on Knowledge and Data Engineering Volume 21, no 9, 2009

[35] Hinton, Geoffrey; *Neural Networks for Machine Learning*
Coursera, URL: `http://coursera.org/course/neuralnets`

[36] Theano, URL: `http://deeplearning.net/software/theano/`

[37] Torch7, URL: `http://torch.ch/`