Classification of Pipe Weld Images with Deep Neural Networks
— Final Report —

Dalyac Alexandre
ad6813@ic.ac.uk

Supervisors: Professor Murray Shanahan and Mr Jack Kelly
Course: CO541, Imperial College London

June 23, 2014

Abstract

Automatic image classification experienced a breakthrough in 2012 with the advent of GPU implementations of deep neural networks. Since then, state-of-the-art has centred around improving these deep neural networks. The following is a literature survey of papers relevant to the task of learning to automatically multi-tag images of pipe welds, from a restrictive number of training cases, and with high-level knowledge of some abstract features. It is therefore divided into 5 sections: foundations of machine learning with neural networks, deep convolutional neural networks (including deep belief networks), multi-tag learning, learning with few training examples, and incorporating knowledge of the structure of the data into the network architecture to optimise learning.

In terms of progress, several instances of large-scale neural networks have been trained on a simplifed task: that of detecting clamps in Redbox images. None of them have shown signs of parameter convergence, suggesting that the classification task is particularly challenging. Potential reasons are discussed; the next steps of the project are to test them.

Contents

1 Introduction

The background goes through the essential material regarding machine learning with feed-forward neural networks. Since this project was experimental from an early phase, the rest of the report is divided into chapters each of which go over the conceptual motivations, the design and the implementation of a main experiment.

2 Background

This project aims to automate the classification of pipe weld images with deep neural networks. After explaining and formalising the problem, we will explain fundamental concepts in machine learning, then go on to explain the architecture of a deep convolutional neural network with restricted linear units, and finally explain how the network is trained with stochastic gradient descent, backpropagation and dropout. The last sections focus on three challenges specific to the pipe weld image classification task: multi-tagging, learning features from a restricted training set, and class imbalance.

2.1 Defining the Problem

The problem consists in building a classifier of pipe weld images capable of detecting the presence of multiple characteristics in each image.

2.1.1 Explaining the Problem

Practically speaking, the reason for why this task involves multiple tags per image is because the quality of a pipe weld is assessed not on one, but 17 characteristics, as shown below.

Characteristic	Penalty Value
No Ground Sheet	5
No Insertion Depth Markings	5
No Visible Hatch Markings	5
Other	5
Photo Does Not Show Enough Of Clamps	5
Photo Does Not Show Enough Of Scrape Zones	5
Fitting Proximity	15
Soil Contamination Low Risk	15
Unsuitable Scraping Or Peeling	15
Water Contamination Low Risk	15
Joint Misaligned	35
Inadequate Or Incorrect Clamping	50
No Clamp Used	50
No Visible Evidence Of Scraping Or Peeling	50
Soil Contamination High Risk	50
Water Contamination High Risk	50
Unsuitable Photo	100

Table 1: Code Coverage for Request Server

At this point, it may help to explain the procedure through which these welds are made, and how pictures of them are taken. The situation is that of fitting two disjoint polyethylene pipes with electrofusion joints [4], in the context of gas or water infrastructure. Since the jointing is done by hand, in an industry affected with alleged "poor quality workmanship", and is most often followed by burial of the pipe under the ground, poor joints occur with relative frequency [4]. Since a contamination can cost up to 100,000 [4], there exists a strong case for putting in place protocols to reduce the likelihood

of such an event. ControlPoint currently has one in place in which, following the welding of a joint, the on-site worker sends one or more photos, at arm's length, of the completed joint.

Figure 1: soil contamination risk (left), water contamination risk (centre), no risk (right)

These images are then manually inspected at the ControlPoint headquarters and checked for the presence of the adverse characteristics listed above. The joint is accepted and counted as finished if the number of penalty points is sufficiently low (the threshold varies from an installation contractor to the next, but 50 and above is generally considered as unacceptable). Although these characteristics are all outer observations of the pipe fitting, they have shown to be very good indicators of the quality of the weld [4]. Manual inspection of the pipes is not only expensive, but also delaying: as images are queued for inspection, so is the completion of a pipe fitting. Contractors are often under tight operational time constraints in order to keep the shutting off of gas or water access to a minimum, so the protocol can be a significant impediment. Automated, immediate classification would therefore bring strong benefits.

2.1.2 Formalising the problem: Multi-Instance Multi-Label Supervised Learning

The problem of learning to classify pipe weld images from a labelled dataset is a Multi-Instance Multi-Label (MIML) supervised learning classification problem [2]:

> Given an instance space $\mathcal{X}$, a set of class labels $\mathcal{Y}$, a dataset $\{(X_1, Y_1), (X_2, Y_2), ..., (X_n, Y_n)\}$, learn a function $f : 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ where
>
> $X_i \subseteq \mathcal{X}$ is a set of instances $\{x_1^{(i)}, x_2^{(i)}, ..., x_{p_i}^{(i)}\}$
>
> $Y_i \subseteq \mathcal{Y}$ is the set of classes $\{y_1^{(i)}, y_2^{(i)}, ..., y_{p_i}^{(i)}\}$ such that $x_j^{(i)}$ is an instance of class $y_j^{(i)}$
>
> p_i is the number of class instances (i.e. labels) present in X_i.

This differs from the traditional supervised learning classification task, formally given by:

> Given an instance space $\mathcal{X}$, a set of class labels $\mathcal{Y}$, a dataset $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, learn a function $f : \mathcal{X} \to \mathcal{Y}$ where
>
> $x_i \in \mathcal{X}$ is an instance
>
> $y_i \in \mathcal{Y}$ is the class of which x_i is an instance.

In the case of MIML, not only are there multiple instances present in each case, but the number of instances is unknown. MIML has been used in the image classification literature when one wishes to identify all objects which are present in the image [2]. Although in this case, the motivation is to look out for a specific set of pipe weld visual characteristics, the problem is actually conceptually the same; the number of identifiable classes is simply lower.

2.1.3 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 $f^* : 2^{\mathcal{X}} \to 2^{\mathcal{Y}}$ 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 .

2.1.4 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 [5]. 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. In other words, approximation and generalisation are the two optimisation criteria for supervised learning, and both need to be incorporated into the cost function.

2.2 Architecture of a Deep Convolutional Neural Network with Rectified Linear 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).

2.2.1 Models of Neurons

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.

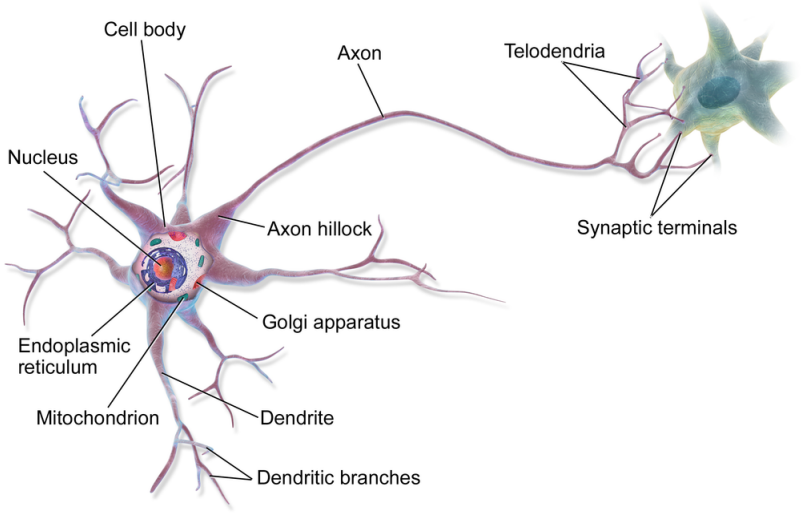

Figure 2: a multipolar biological neuron

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 [6]. 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 below, 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 sophistication

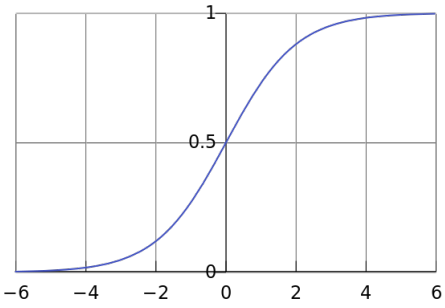

Figure 3: single-input logistic sigmoid neuron

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 [8]. Until now, this has more than offset the per-neuron information loss - and saturation risks - of the rectifier versus the sigmoid unit [9].

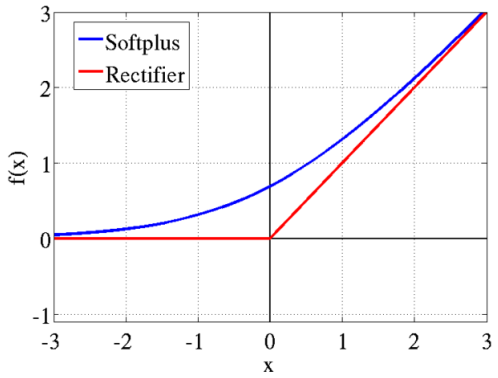

Figure 4: single-input rectified linear neuron

Softmax Neuron

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

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 $\frac{dy}{dz} = \frac{y}{1-y}$.

2.2.2 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. The figure below (taken from [6]) illustrates this.

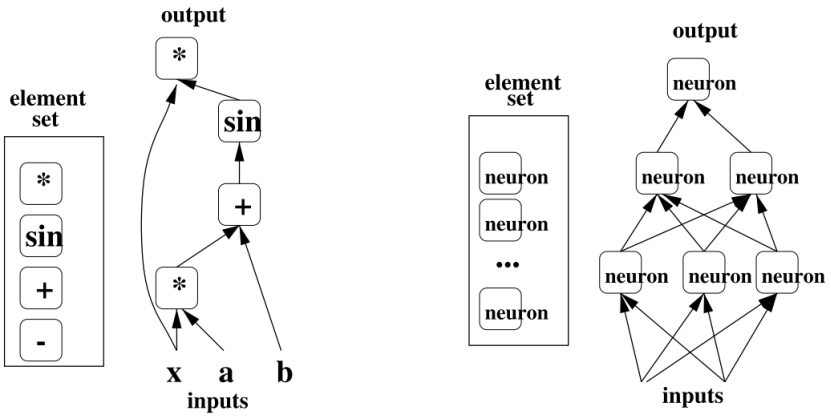

Figure 5: graphical representation of $y = x * sin(a * x + b)$ and of a feed-forward neural network

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 [6], 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 [6]. 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 [6]. 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.

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 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 certain class membership.

Mathematically, it was proved in 1989 that MLPs are universal approximators [10]; hidden layers therefore increase the size of the function space, and solve the initial limitation faced by perceptrons.

Deep Convolutional Neural Networks: for translation invariance A convolutional neural network uses a specific network topology that is inspired by the biological visual cortex and tailored

for computer vision tasks, because it achieves translation invariance of the features. Consider the following image of a geranium: a good feature to classify this image would be the blue flower. This feature appears all across the image; therefore, if the network can learn it, it should then sweep the entire image to look for it. A convolutional layer implements this: it is divided into groups of neurons (called *kernels*), where all of the neurons in a kernel are set to the same parameter values, but are 'wired' to different pixel windows across the image. As a result, one feature can be detected anywhere on the image, and the information of where on the image this feature was detected is contained in the output of the kernel. Below is a representation of LeNet5, a deep convolutional neural network used to classify handwritten characters.

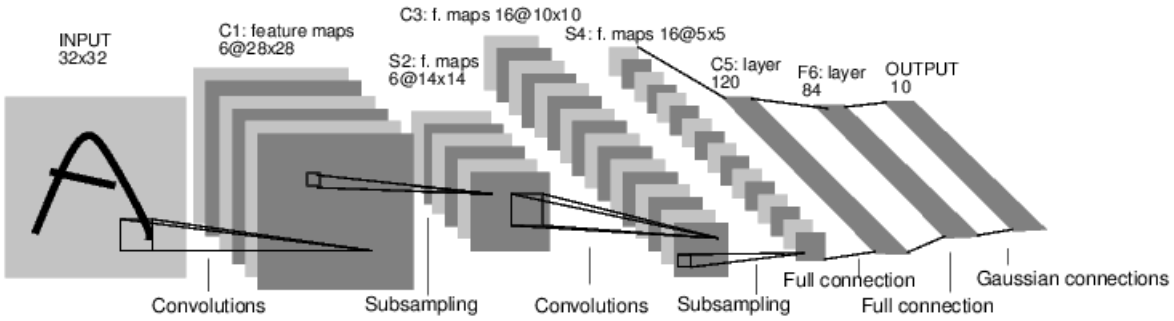

Figure 6: LeNet7 architecture: each square is a kernel

2.3 Training: Backpropagation

Now that the architecture of a deep CNN has been explained, 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.

2.3.1 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 (note that the bound on weight perturbations is no longer tight if we drop the assumption of fully connected layers).

2.3.2 Update Weight Values (with Gradient Descent)

The learning rule is given by $w_{i,t+1} = w_{i,t+1} + \tau \cdot \frac{\partial E}{\partial w_{i,t}}$

Visually, this means that weight values move in the direction the will reduce the error quickest, i.e. the direction of steepest 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 the figure below, two potential training sessions are illustrated. The minima attained in each cases are not the same. This illustrates a strong shortcoming with backpropagation: parameter values can get stuck in poor local minima.

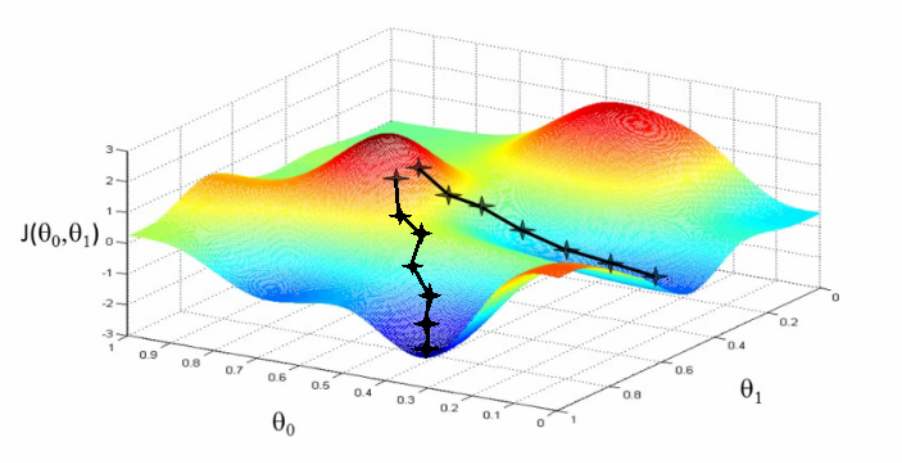

Figure 7: an error surface with poor local minima

Training, Validation and Test Sets As mentioned previously, learning is not a mere approximation problem because the hypothesis function must generalise well to any subset of the instance space. Approximation and generalisation are incorporated into the training of deep neural networks (as well as other models [**?**]) by 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 learned is the one that minimises the error over the validation set, and its performance is measured on the test set. The distinction between training and validation sets is what prevents the function from overfitting the training data: if the function begins to overfit, the error on the validation set will increase and training will be stopped. 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.

2.4 Challenges specific to the Pipe Weld Classification Task

A number of significant challenges have arisen from this task: multi-tagging, domain change, small dataset size (by deep learning standards) and class imbalance. Before going into them, an overview of the data is given below.

2.4.1 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 given below.

2.4.2 Multi-Tagging

:

Characteristic	Count
No Ground Sheet	30,015
No Insertion Depth Markings	17,667
No Visible Hatch Markings	28,155
Other	251
Photo Does Not Show Enough Of Clamps	5,059
Photo Does Not Show Enough Of Scrape Zones	21,272
Fitting Proximity	1,233
Soil Contamination Low Risk	10
Unsuitable Scraping Or Peeling	2,125
Water Contamination Low Risk	3
Joint Misaligned	391
Inadequate Or Incorrect Clamping	1,401
No Clamp Used	8,041
No Visible Evidence Of Scraping Or Peeling	25,499
Soil Contamination High Risk	6,541
Water Contamination High Risk	1,927
Unsuitable Photo	2
Perfect (no labels)	49,039

Table 2: Count of Redbox images with given label

As mentioned earlier, training images contain varying numbers of class instances; this uncertainty complexifies the training task.

2.4.3 Domain Change

:

Domain change can be lethal to machine 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, more worrying is the difference in image sharpness between Redbox and Bluebox images, as can be seen below. It remains to be seen how a CNN could be made to deal with this type of domain change.

Figure 8: left: a Redbox photo - right: a Bluebox photo

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 [11], 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 [12].

2.4.4 Small Dataset Size

Alex Krizhevsky's record-breaking CNN was trained on 1 million images [8]. Such a large dataset enabled the training of a 60-million parameter neural network, without leading to overfit. In this case, there are 'only' 127,000, 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 [13], 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 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.

2.4.5 Class Imbalance

The dataset suffers from a similar adverse characteristic to that of medical datasets: pathology 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 [8], [13], [1]), 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?

One possible solution would be to use a different cost function: the F-measure [14], which is known to deal with these circumstances. Although the F-measure has been adapted to into a fully differentiable cost function (which is mandatory for gradient descent), there currently exists no generalisation of it for the case of $n \geq 2$ classes. Moreover, one would need to ensure that the range of the error is as large as possible for a softmax activation unit, whose own output range is merely $[0; 1]$.

3 Task 1: Generic Clamp Detection

3.1 Motivations: Assumed Simplicity

Clamp detection was suggested by ControlPoint as a simple test run for training a CNN on their dataset. It was a standard single-tag task with few classes, so implementation was not as difficult as multi-tag, and the detection task in itself was deemed simple since clamps are large objects which would be easy to see. Moreover, four significant obstacles arose: non-converging error rates, class imbalance, data complexity, and mis-labelling.

3.2 Design: AlexNet

3.2.1 Overview

Winner of ILSVRC 2012 by far. Decent out-of-the-box benchmark for rapid evaluation of data complexity.
Trained on a bigger dataset, so expected overfit to occur quickly.

3.2.2 Network Architecture: Large-Scale Convolutional Neural Network

find the image for AlexNet with the maps and kernels and stuff.

Convolutional Layer

Convolution

Pooling Why do conv layers 3 and 4 not have it?

Normalisation Why do conv layers 3 and 4 not have it?

Fully Connected Layer

Softmax Output Layer

3.3 Implementation: Out-of-the-box API

3.3.1 Cuda-Convnet

Cuda-Convnet is a GPU implementation of a deep convolutional neural network implemented in CUDA/C++. It was written by Alex Krizhevsky to train the net that set ground-breaking records at the ILSVRC 2012 competition, and subsequently open-sourced. However, parts of his work are missing and his open source repository is not sufficient to reproduce his network. In order to do so, data batching scripts were written for turning raw .jpg images and separate .dat files into pickled python objects each containing 128 stacked jpg values in matrix format with RGB values split across, and a dictionary of labels and metadata for the network.

3.3.2 Hardware

CUDA-enabled GPU. How many teraflops? Which operations are parallelised? First a GTX GeForce 780 with 4GB RAM.

3.3.3 Summary

DOES THIS SUBSUBSECTION REALLY BELONG HERE? yes I would say so, but may need to be edited so that it just mentions how tough the task turned out to be, and that the obstacles which were discovered lead to devising a new task, that of joint-type classification. Perhaps also mention that unsupervised learning was considered to deal with the mis-labelled data, the reasons for why this was put aside (discussions with ControlPoint), but that it would make for interesting further research? Maybe it should be renamed to summary, and be included as final subsubsection of every implementation subsection?

Several weeks into training, it was discovered by visual inspection and extended communication with ControlPoint staff that there existed 6 completely different types of clamps – one for each different type of pipe joint [insert photos of different clamp types here]. It was also discovered that in a significant number of cases the glint of a clamp was sufficient to judge it present, and that in other cases the presence of clamps in certain areas of the pipes did not count as clamp presence [have images to illustrate this point]. This makes learning very difficult, as a class can be pictured as a disjoint set of subclasses, some of which fine-grained, and that one subclass could trigger a false positive of another class.

The solution was to train a classifier capable of recognising the type of joint having been welded, using the small subset of data which contained joint-type labels, in order to try and tag the remaining dataset. The advantage of knowing the joint-type is that we can isolate clamp types, and train a classifier on s single type of clamp. That way, the learning task is simplified, and one is able to focus on the other challenges.

3.4 Experimentation: Non-Converging Error Rates and Class Imbalance

3.4.1 Non-Converging Error Rates

The training produced confusing results. LOOK THROUGH ENTIRE DOCUMENT, when talking about this training, the classification validation error achieved was 11%, not 40%! It may also be good to get a test error.

They were trained on the clamp detection task using the Redbox data only. The purpose of these models is to set a benchmark for the difficulty of the classification task, by taking several out-of-the-box network configurations: namely, those found on open source projects or detailed in papers [8], [13], [1].

The classification task is a challenging one: indeed, none of the training sessions have shown any signs of parameter convergence, as the below plot of the training and validation errors over batch iterations can show. These results are very confusing: backpropagation is guaranteed to converge [cite! go into more detail about this! include the proof in your background!], and yet both training and validation errors seem stuck in volatile periodicity.

You should also justify why you are not bothering with the test error!

The error being shown is the negative of the log probability of the likelihood function:

$$\frac{1}{n}\sum_{i=1}^{n} ln(f(W|x_i)) \tag{8}$$

Where f is the cost function I think. So in this case it is the cross entropy I think.

A number of potential explanations for the volatile periodic error rates were considered:

- The learning rates are too high (the minimum gets 'overshot' even when the direction of partial derivatives is correct)

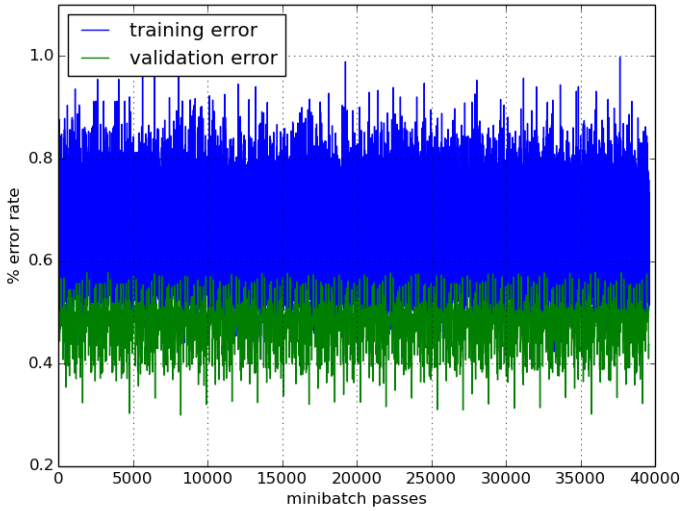

Figure 9: Test Run Training Results

- The dropout rate is too high (dropout is a technique which consists in randomly dropping out neurons from the net at every iteration to prevent overfit - but it also means that a different model is tested every time)
- The number of parameters is too high (the out-of-the-box implementation contains millions of parameters, which is more than than the number of training cases, so collinearities between the parameters cannot even be broken, and most of them are rendered useless)
- Class imbalance (not enough information about the clampless classes to be able to learn feature for them)
- Mis-labeled data (at the time of human tagging of these images, tags were left out) ...
- The error rates are not computed correctly ...

Increase Test Error Precision By computing the test error on a large and unchanging sample of images, one obtains more precise estimates, and convergence can clearly be seen.

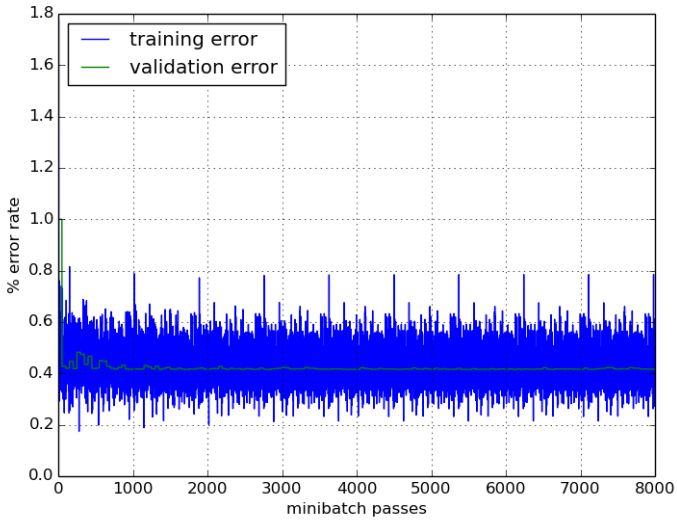

Figure 10: test and validation error rates for clamp detection after fixing a large test range

I trained AlexNet on 100k images of 3 different classes with class imbalance (1 class takes up 90% of data), 40% mis-labelling, and a very small learning rate (0.0001). The emerging periodicity of the training error corresponds to the number of batches, so it looks like the parameters have stopped changing, ie the network has reached a local minimum (flesh out the intuition for why periodicity implies convergence, just going around in the same circle). Why does no overfitting take place? There is not much data compared to capacity of the network, there is a weight decay of 0.0005, there is no dropout. Since overfitting hasn't been reached, maybe this is a poor local minimum?

Francis Quintal Lauzon: I would tend to think learning was indeed stuck, though I would not go so far as to suggest this is caused by a local minimum. Indeed, I have sometimes seen long plateaus followed by sudden steep drop in error. One way I see this is learning with a given learning rate and reducing the learning rate when validation error (or rather, a smoothed version of the validation error) stops going down. This suggest than rather being stuck at a local minima, learning is simply blundering around some minimum, may be because of a combination of the local shape of the cost surface and learning steps too large.

Plateau hypothesis: on the image I show only 10 epochs, but I trained it for 44 epochs and the periodicity extends all the way. When you were experiencing plateaus, did they stretch over more than 1 epoch? because to me, this periodicity suggests roughly the same gradients are repetitively being computed at the same locations on the error surface, so no good keeping on going.

Learning rate hypothesis: so you are suggesting that the minimum keeps getting overshot. But 0.0001 learning rate is already very small no?

I guess another possibility is that I'm zigzagging in a very tight bowl, but seems unlikely it would go on for as long as 40 epochs?

Francis Quintal Lauzon: I trained on much larger (though highly correlated) dataset so I never trained for 44 epochs. I did use learning rate of the same magnitude you are using and still, reducing it after a plateau does help (in my experience). If you are using momentum, you also might want to reset any momentum to zero after facing a plateau (this might helps as well).

Another striking aspect is how volatile the training error is, even when the test error is stable. That the training error is computed correctly was checked by training an out-of-the-box net on a well-documented task: MNIST. Therefore, it was established that in this setup, the training error is indeed volatile and periodic despite the validation error converging, and that an explanation must exist. This could be interpreted as the network's parameters having converged to values that make it good on certain batches, and bad on others. What is curious is that this would suggest that the contents of some batches are very different from one another (unless one considers the paper "Intriguing Properties of Neural Networks"). Therefore, it might be interesting to look at these batches in detail.

Alter Momentum What is momentum for?

the intuition (maybe wrong): if a smaller learning rate and zero momentum could help you deal with a plateau, it means that moving in very small steps is better. but if a plateau is a continuous flat surface, then surely a smaller step will take you longer to reach the other side? on the other hand, if you're in a very tight and stretched bowl, or if there's a very narrow ditch surrounded by a plateau, then the smaller step will help?

Increase Is that used to get past the local minimum? Or is it used to rush past a plateau?

Barely noticeable change in train error: still periodic, slightly different shape of period. Slight increase in test error though. Can interpret that weight updates are slightly less optimal since they don't follow the direction given by the gradients, since there's this extra momentum factor, which is bigger.

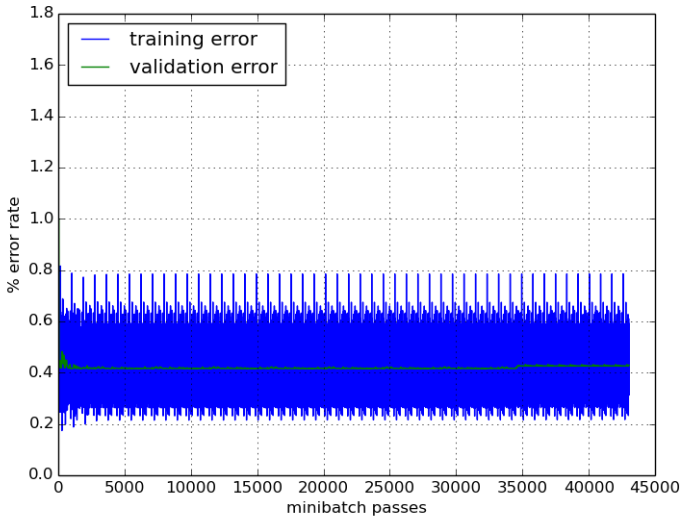

Figure 11: test and validation error rates for clamp detection after raising momentum

Decrease Because Francis Quintal Lauzon says so. But does he mean to reduce it once you're done with the plateau?

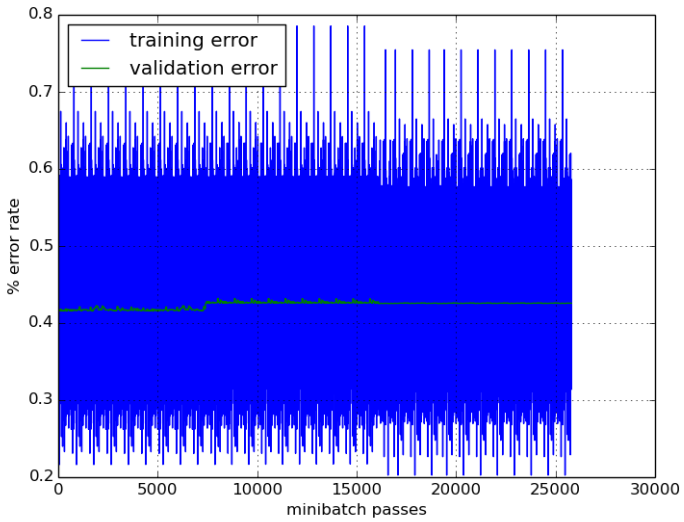

Figure 12: test and validation error rates for clamp detection after setting momentum to zero

The training error decreases a little on average (would be good to assert this statistically). Once again, because momentum is not distorting direction of descent.

The test error stops to jitter completely. So the tiny pulses observed during convergence (i.e. between 5000 batches and 35000) were caused by momentum. I guess it's the result of going upslope a bit just in case there's a better minimum nearby. Could be interesting to put a stupidly high momentum like 1.5 to double check that jittering indeed gets even bigger. Who knows, this might settle the network in another minimum.

Reduce Learning Rate $train_output.txt$ contains data to plot this. Good to put it in to show that you have checked everything meticulously, and to show that it's evidence for the hypothesis of stuck in corner solution.

Stuck in Sampling-Induced, "fake" Corner Minimum I trained a neural network to detect clamps in the images. It very quickly converged to logprob 0.4, i.e. 10% classification error, as can be seen in attachment.

But what is surprising is that the training error remains in the neighbourhood of the test error, without converging to zero and without the network overfitting (should converge to zero since neural networks are 'good' universal approximators).

I played around with momentum and the learning rate, which typically help to deal with plateaus, tight bowls, and poor local minima. (you can see it on the graph, the test error jitters a bit more, and then becomes completely constant, and the amplitude of the training error varies a bit). It didn't help.

The reason for why the network converges quickly to logprob 0.4 minimum, and stays there, without ever overfitting, is because severe class imbalance has introduced a "fake" minimum in a "corner" of the error surface. (By error surface, I mean error on the z axis, and parameter values on all the other axes). This local minimum is very hard to get out of because it's quite deep (you get 10% error rate - it might even be the global minimum), and it's far away from where the "real" minima are.

To find evidence to support this claim, we can use the periodicity in the training error to verify the claim that the network is stuck in this deep, fake, corner minimum: take the batches that consistently score very high train error, take those that consistently score very low, and look at the images. Between the top scoring ones and the low scoring ones, is there a significant different in the proportion of "clamp detected" images? Or is there a significant difference in something else (eg unsuitable photos, mislabelling)?

Intuitively, the network goes like "wait a second, if I output 'clamp detected' every time, then I get 10% error, that's awesome, let's just do that."

sutop is a list of the absolute best performing batches in terms of training error top is a list of top performing worst is a list of worst suworst is a list of absolute worst top performing batches: 280, 543, (195, 185, 177, 157, 156, 147, 143, 82, 67, 53, 19, 18 and others) worst performing batches: 152, (302, 162, 105, 87, 60, 39, 736, 710, 643, 631 and others) 280, 543 achieve 0.18-0.22, the others are in the 0.20s. 152 achieves 0.7-0.8 error, the others achieve in the 0.60s.

Training Error	"Clamps Detected"	"No Clamps"	"Clamps Not Sufficiently Visible"
18-22%	0.96484375	0.0234375	0.01171875
20-29%	0.94161184	0.03371711	0.02467105
60-69%	0.81534091	0.10795455	0.07670455
70-82%	0.765625	0.15625	0.078125

Table 3: Class Proportions Across Batches that Score Different Training Errors

Would be nice to add the plot of the errors and point to which batches the spikes correspond to.

This suggests that the only thing that determines error is proportion of non-"clamp detected" cases. this is very strong evidence that the net has learned to output "clamp detected" all the time.

Do you think I should try and find completely foolproof evidence? Like running dummy input through the network?

3.4.2 Class Imbalance

SHOULD I REALLY IMPLEMENT THIS? So much work just on the first task which was supposed to be preliminary...

Three approaches were taken to tackle class imbalance, the first ones being simplest to implement, the last ones most sophisticated and information-preserving.

Subset of Training Set CNNs were trained on all of the "No Clamps" or "Clamps Not Sufficiently Visible" images, and a random sample containing 15% of the "Clamps Detected" images, in order to obtain an acceptably balanced dataset. This fix was easiest to implement and served as a benchmark for the more sophisticated subsequent approaches. Naturally, one would also expect this approach to be least effective: 85% of the training data was lost, data which could intuitively serve not to teach the network to distinguish classes, but to learn features that are good encoders of pipe weld images, from which an effective discriminative setup of the parameters could ensue.

Partial Data Augmentation

F Measure

4 Task 2: Joint-Type Classification

4.1 Motivations

4.2 Design

4.3 Implementation

4.3.1 Training the Joint-Type Classifier

4.3.2 Applying the Joint-Type Classifier to Specific Clamp Detection

4.3.3 Summary

4.4 Experimentation

4.4.1 Overfit Problem: Small Dataset

4.4.2 Overfit Solution: Transfer Learning

Use transfer learning to make up for the lack of training examples.

5 Task 3: Full Multi-Tagging Pipe Weld Classification Task

5.1 Motivations

5.2 Design

5.3 Implementation

5.3.1 Theano

python wrappers for CUDA, LISA. Uses some of Krizhevsky's cuda-convnet code.

Greater flexibility than cuda-convnet, which is monolithic to some extent.

symbolic programming - completely new to me, took time.

5.3.2 Hardware

CUDA-enabled GPU. How many teraflops? Which operations are parallelised? Moving from the GTX 780 to a Titan Black with ...

5.4 Experimentation

5.5 Motivations

5.6 Design

5.7 Implementation

5.7.1 Summary

5.8 Experimentation

5.8.1 Tight Bowl Zig-zagging

5.8.2 Image Preprocessing

blurring if only colour matters, grey-scale if only shape matters, filter from ControlPoint.

6 Conclusions and Future Work

The realisation that weird sampling can heavily dent the approximation of the error surface has raised questions:

How does the training set provide an approximation of the true error surface? Since it can introduce dangerous minima, does that mean that, formally, it does not always provide an extrema-conserving approximation of the true error surface? Theoretically, what are the conditions for obtaining an extrema-conserving approximation (i.e. that doesn't introduce fake minima)? Practically, can we perform transformations on the cost function, or do stuff to our data (sampling), to limit the introduction of fake minima?

Could we answer these questions if, instead of facing the usual problem of having a training set sampled from an unknown distribution, we ran a completely artificial experiment where we start with a known distribution? That way, we know the true error surface, and as we draw samples from the distribution, we can look at how each one approximates the true error surface, how it introduces bad minima?

How does mis-labelling alter the error surface? Intuitively, it would seem that it lifts up the true minima only, making the false minima even more attractive (e.g. "labelling is so bad it's too confusing, there's just nothing to distinguish them, so I might as well go for the blind strategy of outputting clamp detected all the time").

Unsupervised Learning to correct mis-labelling, with encouragements to create clusters initialised by those resulting from supervised learning (maybe could only work well with many false negatives, few false positives, as is the case here?).

References

[1] 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

[2] 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

[3] Pastor-Pellicer, Joan; Zamora-Martinez, Francisco; Espana-Boquera, Salvador; Castro-Bleda, Maria Jose; *F-Measure as the Error Function to Train Neural Networks*

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

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

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

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

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

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

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

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

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

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

[14] 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

[15] URL: `https://code.google.com/p/cuda-convnet/`, last accessed 6th June 2014.