

Classification of Pipe Weld Images with Deep Neural Networks

— Literature Survey —

Dalyac Alexandre
ad6813@ic.ac.uk

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

June 6, 2014

Abstract

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, a deep convolutional neural network, pretrained on 10 million images from ImageNet, has been trained on 150,000 pipe weld images for the simplest possible task of discriminating 'good' pipe weld from 'bad' pipe welds. A classification error rate of ??% has been attained. Although this figure is encouraging, it corresponds to a much simpler formulation of the task: the objective of this project is to achieve multi-tagging for 23 characteristics, some of which require considerable human training. Include a separate section on progress that describes: the activities and accomplishments of the project to date; any problems or obstacles that might have cropped up and how those problems or obstacles are being dealt with; and plans for the next phases of the project.

Contents

1	Background	3
1.1	Defining the Problem	3
1.1.1	Explaining the Problem	3
1.1.2	Formalising the problem: Multi-Instance Multi-Label Supervised Learning . . .	4
1.1.3	Supervised Learning	4
1.1.4	Approximation vs Generalisation	5
1.2	Architecture of a Deep Convolutional Neural Network with Rectified Linear Units . .	5
1.3	Training	6
1.4	Challenges specific to the Pipe Weld Classification Task	6
1.4.1	Multi-Instance Multi-Label Learning	6
1.4.2	Transfer Learning: learning with a restricted training set	6
1.4.3	Class Imbalance	6
2	Progress	7
2.1	Data: pipe weld images	7
2.1.1	Visual Inspection	7
2.1.2	Analysis	7
2.2	Implementation: Cuda-Convnet	7
2.2.1	Test Run	7
2.3	Further Work	8
2.3.1	Improve Current Model	8
2.3.2	Modify Model	9
2.3.3	Improve Implementation	9

1 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.

1.1 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.

1.1.1 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 [6]. Since a contamination can cost up to 100,000 [5], 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.

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 [7]. 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



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

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.

1.1.2 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 $\{(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)\}$, learn a function $f : 2^{\mathcal{X}} \rightarrow 2^{\mathcal{Y}}$ where

$X_i \subseteq \mathcal{X}$ is a set of instances $\{x_1^{(i)}, x_2^{(i)}, \dots, x_{p_i}^{(i)}\}$

$Y_i \subseteq \mathcal{Y}$ is the set of classes $\{y_1^{(i)}, y_2^{(i)}, \dots, y_{p_i}^{(i)}\}$ 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 $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ 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.

1.1.3 Supervised Learning

Learning in the case of classification consists in using the dataset \mathcal{D} to find a hypothesis function f^h which 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 $\{(x_1, f^*(x_1)), (x_2, f^*(x_2)), \dots, (x_n, f^*(x_n))\}$. 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

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

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

1.1.4 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) make this a relatively simple task. 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.

1.2 Architecture of a Deep Convolutional Neural Network with Rectified Linear Units

Learning a hypothesis function f^h comes down to searching a function space for the function which Choosing a deep convolutional neural network with rectified linear units as a classifier means restricting the space of hypothesis functions

Feed-Forward Architecture A feed-forward neural network is a graph where nodes represent neurons, and edges represent synapses. Data travels from one end of the network to the other via the edges. Each neuron's output is the result of some operation on the data it receives.

Deep Neural Networks: the Multilayer Peceptron The graph is made up of several layers. The top layer is the output layer, where each neuron represents a class known by the classifier. The output value of neuron k is the probability that the candidate being fed into the network is a member of class k . The other layers are called hidden layers, because they are not entirely under the programmer's control: the programmer doesn't hard-code what they do, but rather leaves it under the learning algorithm's control. What often ends up happening is that the hidden layers become feature detectors. Moreover, a higher hidden layer ends up detecting the presence of features which combine, build upon and complexify the features detected in the layer below. The top layer can be viewed as doing the same, where a class specification is a specific combination of very complex features.

With deep neural nets, the bottom layer is the input layer, where each neuron takes as input a component of the data vector. So the number of input neurons is the dimensionality of the data. Our input vector will contain at least as many dimensions as there are pixels in the image to recognise. It might also have neurons for GPS data, or for multiple images if the user is multi-querying. The dimensionality is going to be high!

Deep neural nets are a good strategy for learning features because they can learn non-linear patterns. Features are typically complex patterns, and complex patterns are typically non-linear. It was mathematically proven in 1969 that a peceptron cannot learn a non-linear pattern.

Models of Neurons Linear Neuron: $y = b + \sum_{i=1:n} x[i] * w[i]$ y : output b : bias x : inputs w : weights n : number of input neurons ie neurons sending their output to the given neuron

Binary Threshold Neuron: $y = 1$ if $M \leq b + \sum_{i=1:n} x[i] * w[i]$ 0 otherwise M is the threshold y takes a hard decision, just like biological neurons y produces spikes $x[i]$ is the truth value of a relevant proposition a way of combining information from many (possibly unreliable) sources of information n : number of input neurons ie neurons sending their output to the given neuron

Rectified Linear Neuron: $y = \max(0, b + \sum_{i=1:n} x[i] * w[i])$ b can be negative! semi soft, semi hard decision making below certain threshold I jump to certainty, but above that my

confidence is incremental -i doesn't produce spikes like a biological neuron -i may produce zero flow, and only ever increases flow gradually

Logistic aka Sigmoid Neuron: -i $y = 1 / (1 + \exp(-z))$ $z = b + \sum_{i=0; i \leq n; i++} x[i] * w[i]$ -i uses logistic function, $\text{logistic}(z)$ -i most commonly used -i fully differentiable -i bijection between $]-\infty; +\infty[$ and $[0;1]$ -i $y(0) = 0.5$ -i gets harder and harder to convince me further -i computationally expensive -i cuda-convnet uses rectified linear neurons instead for that reason

Stochastic Binary Neuron: -i $P(y=1) = 1 / (1 + \exp(-z))$ $P(y=0) = 1 - P(y=1)$ -i y is a Bernoulli random variable whose prob mass function is given by logistic value -i the number of spikes produced per time frame follows a Poisson distribution, the parameter of which is deterministic -i parameter is a function of logistic value

Softmax Neuron Equation $y[j] = \exp(z[j]) / (\sum_{k=1:n} \exp(z[k]))$ this enforces $\sum_{j=1:n} y[j] = 1$ conveniently, $dy/dz = y(1-y)$

Convolutions

Translation Invariance CITATION BELOW! *"Convolutional neural network is a specific artificial neural network topology that is inspired by biological visual cortex and tailored for computer vision tasks by Yann LeCun in early 1990s. See <http://deeplearning.net/tutorial/lenet.html> for introduction. The convolutional neural network implemented in ccv is based on Alex Krizhevskys ground-breaking work presented in: ImageNet Classification with Deep Convolutional Neural Networks, Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, NIPS 2012 The parameters are modified based on Matthew D. Zeilers work presented in: Visualizing and Understanding Convolutional Networks, Matthew D. Zeiler, and Rob Fergus, Arxiv 1311.2901 (Nov 2013)."*

Rectified Linear Units

1.3 Training

Gradient Descent

Backpropagation

Dropout

1.4 Challenges specific to the Pipe Weld Classification Task

1.4.1 Multi-Instance Multi-Label Learning

1.4.2 Transfer Learning: learning with a restricted training set

1.4.3 Class Imbalance

2 Progress

2.1 Data: pipe weld images

There are 227,730 640x480 'RedBox' images. There are 1280x960 'BlueBox' images. They all have xx tags.

Characteristic	Count
No Ground Sheet	30015
No Insertion Depth Markings	17667
No Visible Hatch Markings	28155
Other	251
Photo Does Not Show Enough Of Clamps	5059
Photo Does Not Show Enough Of Scrape Zones	21272
Fitting Proximity	1233
Soil Contamination Low Risk	10
Unsuitable Scraping Or Peeling	2125
Water Contamination Low Risk	3
Joint Misaligned	391
Inadequate Or Incorrect Clamping	1401
No Clamp Used	8041
No Visible Evidence Of Scraping Or Peeling	25499
Soil Contamination High Risk	6541
Water Contamination High Risk	1927
Unsuitable Photo	2
Perfect (no flag)	49039

Table 2: Count of Redbox images with given label

2.1.1 Visual Inspection

2.1.2 Analysis

ANOVA

t-SNE

2.2 Implementation: 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 (see test run for how this was dealt with).

2.2.1 Test Run

Use Alex Krizhevsky's GPU implementation. Use Daniel Nouri's nocn scripts. Download the file containing serialized code for Yangqing's decaf [1] network parameters, reverse engineer it using Daniel Nouri's skynet configuration files. Write batching script for decaf-net's image dimensions. Write data augmentation scripts to reduce overfit. Write c++ code to import the decaf parameters to initialise the weights of a new network, to re-initialise the weights of the fully connected layers, to train only the fully connected layers, keeping the lower layers 'frozen'.

888 batches of 128 images each. Used 10% of the data for testing - so trained on 102,400 images.

DeCAF By Yangqing Jia and University of California, Berkeley. "As the underlying architecture for our feature we adopt the deep convolutional neural network architecture proposed by Krizhevsky et al. (2012), which won the ImageNet Large Scale Visual Recognition Challenge 2012 (Berg et al., 2012) with a top-1 validation error rate of 40.7%. We chose this model due to its performance on a difficult 1000-way classification task, hypothesizing that the activations of the neurons in its late hidden layers might serve as very strong features for a variety of object recognition tasks. Its inputs are the mean-centered raw RGB pixel intensity values of a 224×224 image. These values are forward propagated through 5 convolutional layers (with pooling and ReLU non-linearities applied along the way) and 3 fully-connected layers to determine its final neuron activities: a distribution over the tasks 1000 object categories. Our instance of the model attains an error rate of 42.9% on the ILSVRC-2012 validation set 2.2% shy of the 40.7% achieved by (Krizhevsky et al., 2012). We refer to Krizhevsky et al. (2012) for a detailed discussion of the architecture and training protocol, which we closely followed with the exception of two small differences in the input data. First, we ignore the images original aspect ratio and warp it to 256×256 , rather than re-sizing and cropping to preserve the proportions. Secondly, we did not perform the data augmentation trick of adding random multiples of the principle components of the RGB pixel values throughout the dataset, proposed as a way of capturing invariance to changes in illumination and color.

Network Architecture Don't want to saturate neurons. Need a standard deviation of weight initialisations proportional to the square root of the fan-in. Convolutional layers have a much smaller fan-in than, say, fully connected layers, so the standard deviation needs to be much lower.

Use dropout of 0.5 in each fully connected layer, so add twice as many outputs. Go from 4096 to 8192. Actually no, keep it at 4096. Furthermore, reduce the number of neurons in each fully connected layer, because don't have as many classes to classify. Just need a few abstract features. The last layer has number of neurons fixed to number of classes. So actually, maybe reduce the number of neurons only in the 2nd fully connected one, but not the first: this is like having a few very abstract features, which are combinations of a very high number of less abstract features.

2.3 Further Work

Directions for further work can be divided into three sections: improving the current model i.e. with small changes, and modifying the model i.e. with significant changes, and finally improving the implementation with more flexible, higher performing APIs.

2.3.1 Improve Current Model

Improving the current model consists in optimising the hyperparameters of the model.

Tweak Architecture Change proportion of data used as training data. Change fan-in for the fully connected layers. Change dropout rate (probably don't need to). Does number of classes to classify influences network hyper-parameters, other than number of neurons in top layer? Maybe reduce the number of neurons in each fully connected layer even more so, because don't have as many classes to classify. Just need a few abstract features. The last layer has number of neurons fixed to number of classes. So actually, maybe reduce the number of neurons only in the 2nd fully connected one, but not the first: this is like having a few very abstract features, which are combinations of a very high number of less abstract features.

Tweak Training I'm training a CNN to recognise 3 classes, but 91% of my data is of the same class. Is there anything I should alter in the way I train?

For example, should I make the proportions of each class in my validation batches even ($1/3$, $1/3$, $1/3$)? Because the thing is, weights are only saved when new validation error is lower than lowest validation error until now. But if this validation error is computed over hardly any examples of the other 2 classes, how can it be a good indicator that the network is on its way to learning good features?

See the plot for training: over 39 epochs, there is no progress, weight updates look random. This is a bad sign.

If one views the problem as picking a direction to walk along in the error surface, it looks like the algorithm is stuck at the stage of randomly trying directions, heading down one a little bit, but deciding that it is not a good direction and starting from scratch. This is either due to the correct direction not being computed, or the correct direction not being correctly recognised.

Maybe dropout is too high: dropout makes the logprob over time more spiky because it's like training lots of models concurrently, so there is never much focus on a particular model. Maybe the dropout rate should be reduced.

Maybe the 2 other classes don't appear often enough, so it's impossible to learn anything: this can apparently be solved by choosing a different error function to minimise: the F-score [3]. However, The F-Measure is proposed as an alternative to the MSE - but can it be used as an alternative to cross entropy? I have a softmax output layer, so all output neuron values are between 0 and 1, which drastically reduces the range of the MSE. Would that be the same for F-Measure? Is there a way of making the range of the F-Measure wide for a softmax output layer?

Starting from 39.415, the test frequency was raised from 20 batches to 200 batches. The idea here is to let the weights run along a longer path before evaluating whether this is a good direction (more depth in the search).

2.3.2 Modify Model

Transfer Learning

Hidden Markov Model *It might be interesting to explore hidden markov models because the data has obvious hidden states which humans benefit from learning: pipe welds can be T welds and standard welds, and this alters where scratch marks need to be seen.*

Hidden Markov for pipe type. Whether the joint is a T-joint or not alters the way the image has to be assessed for each of the flags. In other words, whether or not the joint is a T-joint modifies the visual features that have to be picked up by the neural network for classification. Unless I'm mistaken, we do not have data for each image about whether or not it is a T-joint. At first, I had found that a bit disappointing, But I've just been thinking, this might be a very exciting research opportunity: there's a (powerful and super cool) type of machine learning model called a Hidden Markov model, where unknown states can be learned. In this case, the unknown state would "whether or not joint is a T-joint". Knowing this state should in theory help the network to improve classification, because it could indicate to it that a slightly different set of visual features need to be used for classification. I don't think much work has been done on combining neural networks with hidden markov models - and I don't know whether it's possible. But I'd like to mention it in my first report as a potential research path (unless Jack thinks it's a bad idea!). Hence why I wanted to be sure about the terminology for these different joints!

2.3.3 Improve Implementation

Caffe

Theano

Torch

References

- [1] Donahue, Jeff and Jia, Yangqing and Vinyals, Oriol and Hoffman, Judy and Zhang, Ning and Tzeng, Eric and Darrell, Trevor, DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition
arXiv preprint arXiv:1310.1531, 2013
- [2] Zhou, Zhi-Hua and 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] Joan Pastor-Pellicer, Francisco Zamora-Martinez, Salvador Espana-Boquera and Maria Jose Castro-Bleda, 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] Fusion Group - ControlPoint LLP, Company Description
URL: <http://www.fusionprovida.com/companies/control-point>, last accessed 5th June 2014.
- [6] Fusion Group - ControlPoint LLP, Company Description
URL: <http://www.fusionprovida.com/companies/control-point>, last accessed 5th June 2014.
- [7] Fusion Group - ControlPoint LLP, Company Description
URL: <http://www.fusionprovida.com/companies/control-point>, last accessed 5th June 2014.