This is a repository copy of *Convolutional neural networks for the detection of damaged fasteners in engineering structures*.

White Rose Research Online URL for this paper:
http://eprints.whiterose.ac.uk/138024/

Version: Published Version

Proceedings Paper:
Gibbons, T.J. orcid.org/0000-0002-5041-7053, Pierce, S., Worden, K. orcid.org/0000-0002-1035-238X et al. (1 more author) (2018) Convolutional neural networks for the detection of damaged fasteners in engineering structures. In: Proceedings of the 9th European workshop on structural health monitoring (EWSHM 2019). 9th European Workshop on Structural Health Monitoring, 10-13 Jul 2018, Manchester, UK. NDT.net .

Reuse
This article is distributed under the terms of the Creative Commons Attribution-NonCommercial (CC BY-NC) licence. This licence allows you to remix, tweak, and build upon this work non-commercially, and any new works must also acknowledge the authors and be non-commercial. You don’t have to license any derivative works on the same terms. More information and the full terms of the licence here: https://creativecommons.org/licenses/

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

9th European Workshop on Structural Health Monitoring
July 10-13, 2018, Manchester, United Kingdom

Convolutional neural networks for the detection of damaged fasteners in engineering structures

Tom J Gibbons[1], Gareth Pierce[2], Keith Worden[1] and Ifigeneia Antoniadou[1]
1 Dynamics Research Group (DRG), The University of Sheffield, The Department of Mechanical Engineering, UK, t.gibbons@sheffield.ac.uk
2 Centre for Ultrasound Engineering (CUE), University of Strathclyde, Glasgow, UK

More info about this article: http://www.ndt.net/?id=23276

Abstract

Locating and classifying damaged fasteners, such as bolts, in large engineering structures is vital in many health monitoring applications. Whilst traditional signal processing methods are often used to identify the presence of such fasteners, accurately estimating their location remains an ongoing challenge. In recent years, image detection (or the location of objects within images) using deep learning algorithms, such as convolutional neural networks (CNNs), has seen substantial improvements. This is largely due to the abundant database of images provided by internet search engines, as well as significant advances in computing power. Moreover, advances in digital imaging technology mean that affordable computer vision systems are now more readily available than ever before.

In this paper, a CNN architecture is proposed for the task of detecting damaged bolts in engineering structures. The new architecture forms part of a regional convolutional neural network (R-CNN), which applies a bounding box regression algorithm for bolt location alongside a softmax classifier for damage classification. A dedicated training set is also developed, which combines internet search engine data with images of a specifically-designed bolt rig. The new images extend the current dataset with the purpose of developing a bolt detector that is invariant to camera angle and location, as well as environmental factors such as lighting and shadows.

1. Introduction

Although small in size, fasteners, such as bolts and rivets, play a vital role in the performance of many engineering structures. Damage occurs due to a variety of factors such as corrosion, general wear and tear and external loading, which can lead to an overall reduction in performance or even complete failure. It is for this reason that visual inspection of fasteners is an essential operation in many engineering applications. Most commonly, visual inspection is performed by human experts, which can be expensive, dangerous and subjective. Therefore, automated visual inspection of fasteners, using image recognition algorithms, is of great interest to both academic and industrial engineers alike.

1.1 Image recognition

Image recognition is a collective term used to refer to several image processing and machine learning tasks, including but not limited to: image classification, where a single label is assigned to an image from a fixed set of classes, object localisation, where a bounding box is fitted around a single object within in image, and object detection, where bounding boxes are fitted around several objects from various classes within an image.

Creative Commons CC-BY-NC licence https://creativecommons.org/licenses/by-nc/4.0/

In 2010, The Stanford Vision Lab launched the ImageNet Largescale Visual Recognition Challenge (ILSVRC) with the intention of improving current capabilities in image recognition. As part of ILSVRC, the organisers released the largest publically available dataset for image recognition, which includes over 10 million labelled images with 1 million hand annotations (bounding boxes) split over 1000 classes. Since then, there has been dramatic improvements in image recognition accuracy, largely due to the reintroduction of convolutional neural networks (CNNs).

A CNN is a type of deep, feed-forward artificial neural network, that has been adapted specifically for use with large three-dimensional (colour) images. Similarly to traditional neural networks, CNNs extract features, that were traditionally hand engineered, using a set of learnable parameters. However, the networks use convolution to reduce the number of parameters compared to a fully connected network, a characteristic which also makes them shift (or location) invariant i.e. the same features are extracted over the whole image. Many variations on the basic CNN have been submitted to ILSVRC, however, AlexNet [1] was the first to receive substantial attention.

AlexNet was submitted to ILSVRC in 2012 and achieved a top-5 classification accuracy rate of 84.7% (top-5 classification accuracy is where the correct class label is one of 5 highest classification probabilities), a considerable increase in accuracy when compared to previous state-of-the-art classifiers. The network also achieved a top-1 accuracy of 60.3%. The improvement was largely a result of the introduction of several non-conventional machine learning methods. AlexNet was the first CNN to use the rectified linear units (ReLU) activation function to introduce non-linearity, moreover, it was the first CNN to be trained using dropout, where connections between layers are randomly set to zero during training, in order to reduce overfitting. More recent submissions to ILSVRC have achieved top-5 classification accuracy rates as high as 97% [2].

CNNs have also been applied to image detection, most notably in the form of the regional convolutional neural network (R-CNN) [3]. An R-CNN contains three separate modules. Firstly, a fixed number of regions of interest (ROIs), or areas that are most likely to contain a single object, are proposed using the edge boxes algorithm [4]. Each ROI is then passed through a CNN to transform it into a fixed length feature vector. The feature vector is then passed into a classifier, which assigns a label to each ROI, and a parallel bounding box regressor, which fits a box around the object in each ROI. The R-CNN has since been adapted to improve training and test times [5].

2.2 Image recognition for (damaged) bolt detection

One of the main applications of automated damaged fastener detection is in railway track inspection. Marino et al. [6] used a multi-layer perception neural network to classify missing hexagonal bolts in railway tracks, whilst Yang et al. [7] used the principal components of a wavelet transform as features, before applying a linear discriminant analysis to classify bolts as present or missing. More recently, Feng et al. [8] used the line-segment detection algorithm to locate railway tracks and sleepers, and indirectly locate the fasteners, before applying a probabilistic damage classification method.

The current paper looks to apply the recent advances in CNNs, and more specifically R-CNNs, to the problem of detecting damaged fasteners in engineering structures, and is organised as follows. Firstly, a new, specifically-designed, dataset is presented in Section 2, before a more detailed introduction to R-CNN theory in Section 3. In Section 4, a new CNN architecture for damaged bolt classification is discussed, and in Section 5, this is adapted for damaged bolt detection. Finally, conclusions from this work are drawn in Section 6.

2. A new dataset

The dataset used in this paper is a combination of ILSVRC data, and a newly-developed dataset for the specific purpose of damage detection in hexagonal bolt heads. The ILSVRC dataset contains a bolt subset consisting of 1177 labelled images from internet search engines. The images ae all in the red-green-blue (RGB) colour space, and range in size from 60 ×60×3 pixels to 2048×1456×3 pixels. The images range from photographs of bolts in-use to bolts for sale online, they also include a small subset of corroded bolts. A selection of the images can be seen in Figure 1.

Figure 1: A selection of images from the ILSVRC dataset.

Whilst the ILSVRC dataset is extensive, it lacks variance in environmental conditions such as lighting conditions and shadows, as well as camera angle and depth. Moreover, the dataset contains very few damaged bolts. Therefore, a new dataset was collected to extend the ILSVRC dataset. A steel plate was used to house 50 M8 bolts in a regular grid pattern. Images were then taken at a range of depths and angles, before changing the lighting and repeating the same images.

Damage was then induced into the bolts by grinding flat edges of different sizes onto the heads at random locations. The image acquisition process was then repeated. The images were annotated by drawing ground truth bounding boxes around the bolt heads, which later act as target variables for bounding box regression. A selection of the annotated images can be seen in Figure 2. A separate test set was also captured that contains a mixture of damaged and undamaged bolts in random locations.

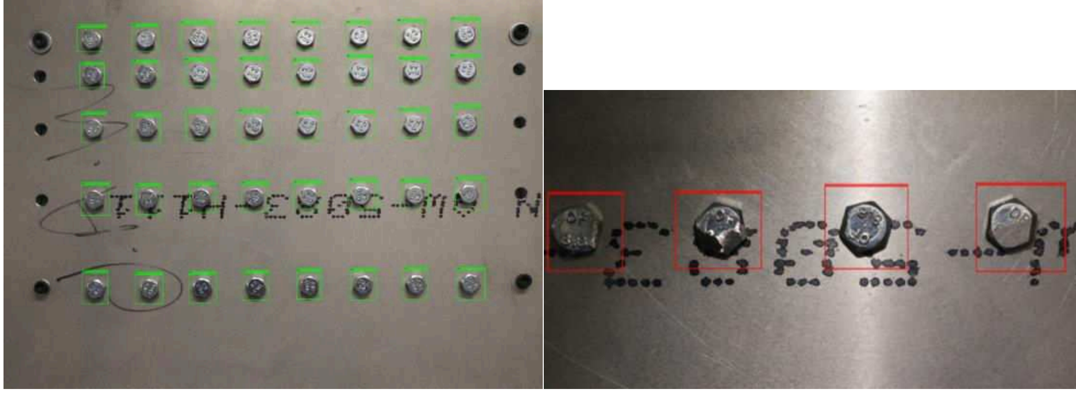

Figure 2: A selection of image from new dataset: green bounding boxes indicate undamaged bolts whilst red indicate damage.

3. Regional convolutional neural networks - theory

The input to any CNN is a 3-dimensional image of size $\mathbf{X} \in \mathbb{R}^{w^{[0]} \times h^{[0]} \times d^{[0]}}$, where $w^{[0]}$ is the width of the image, $h^{[0]}$ is its height, and $d^{[0]} = 3$ is its depth. Throughout this paper, superscripts with square brackets are used to denote the layer number. A CNN maps the input image to some output $\hat{y}$, in the case of classification this is a vector of class probabilities, and in object detection this is a vector describing a bounding box. An R-CNN involves both classification and bounding box regression. The classifier must be able to classify an input as positive (i.e. undamaged bolt or damaged bolt) or negative (background). Therefore, when training the classifier for an R-CNN, negative examples must be included in the training set. Instead of passing the whole image through the CNN, small regions of the image which may or may not contain an object are used as individual training examples. Therefore, a single training image x, is associated with a set of ground truth bounding boxes (hand annotations) g, and their corresponding class labels y,

$$g \in \mathbb{R}^{G \times 4} = \begin{bmatrix} g_1^x & g_1^y & g_1^w & g_1^h \\ \vdots & \vdots & \vdots & \vdots \\ g_G^x & g_G^y & g_G^w & g_G^h \end{bmatrix} \quad \text{and} \quad y \in \mathbb{R}^{G \times 1} \begin{bmatrix} c_i \\ \vdots \\ c_G \end{bmatrix} \tag{1}$$

Where G is the total number of bounding boxes in image x, g_i^x is the x-coordinate of the centre of bounding box i, g_i^y is the y-coordinate of the centre of bounding box i, g_i^w is the width of bounding box i, g_i^h is the height of bounding box i, and $c_i \in \{1, \dots C\}$ is the class label of the ith bounding box. In the case of a damaged bolt detector, $C = 2$.

The image is passed through a region proposal algorithm such as the edge boxes algorithm, which outputs a fixed number of regions of interest (ROIs) p_j. Each of these regions of interest must then be assigned a class label (c_j) and a single bounding box ground truth label (g_j). This is achieved by calculating the intersection over union (IOU) with all ground truth bounding boxes in g. The intersection of union (IOU) is given by,

$$\text{IOU}(p_j, g_i) = \frac{\text{Area of overlap}}{\text{Area of union}} \tag{2}$$

Then, p_j is assigned the target class c_i and target bounding box g_i with which it has the highest IOU, so long as the IOU is greater than 0.7. Also, if the IOU of p_j with all g_i is less than 0.3, p_j is labelled as a negative example ($c_i = 0$) and given no bounding box. All other ROIs are discarded. Therefore, each training example for an R-CNN consists of $p_j(x)$, the pixel values in the region p_j, a class label c_j (which may contain an object $c_j > 0$ or may not $c_j = 0$), and a ground truth bounding box $g_j = \left[g_j^x, g_j^y, g_j^w, g_j^h\right]$. Each of the layers that make up an R-CNN is now discussed in turn.

3.1 Convolutional layers

A convolutional layer replaces the traditional hidden layer in a neural network; it takes as input the activation from the previous layer $\mathbf{A}^{[l-1]} \in \mathbb{R}^{w^{[l-1]} \times h^{[l-1]} \times K^{[l-1]}}$ and transforms this into a new activation map $\mathbf{A}^{[l]} \in \mathbb{R}^{w^{[l]} \times h^{[l]} \times K^{[l]}}$. The layer consists of a set of $K^{[l]}$ learnable filters $\mathbf{W}_k^{[l]} \in \mathbb{R}^{f^{[l]} \times f^{[l]} \times K^{[l-1]}}$ and biases $b_k^{[l]} \in \mathbb{R}$. The spatial size of these filters is usually much smaller than the input ($f^{[l]} \ll w^{[l-1]}$); however, they must extend through the full depth of the input. This paper assumes that all filters are square, however, this is not necessary. A forward pass consists of convolving each of the filters with the input volume i.e. each filter is slid across the width and height of the input and the sum of the elementwise product calculated at each location. As a 3D filter is moved around the input volume, a 2D activation map is produced that gives the responses of the filter at every spatial position. The activation maps from each filter are then stacked on top of one another before applying an activation function, such as the ReLU function ($\mathrm{f}(x) = \max(0, x)$). Therefore, the output of any convolutional layer has a depth equal to the number of filters in that layer; this can be seen in Figure 3.

A 2D convolutional process can be seen in Figure 4, where the red matrix is the input, blue is the filter and green is the resultant activation map. The input is usually padded with zeros so that each neuron (pixel) in the input is connected to the same number of neurons in the output, but it is also used to preserve the spatial size of the input i.e. the hyperparameter $p^{[l]}$ is usually chosen so that the width and height of the output matches that of the input. Another hyperparameter that controls the spatial size of the output is the stride $s^{[l]}$ or the step size with which the filter is moved around the input (in practice this value is usually either 1 or 2). Therefore, it can be shown that the activation map of a convolutional layer has spatial dimensions,

$$w^{[l]} = \frac{w^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \text{ and } h^{[l]} = \frac{h^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \tag{2}$$

The complete forward pass of a convolutional layer is given by,

$$\mathbf{Z}^{[l]}(x, y, k) = \sum_{a}^{f^{[l]}} \sum_{b}^{f^{[l]}} \sum_{c}^{K^{[l-1]}} \mathbf{W}_k^{[l]} \times \mathbf{A}^{[l-1]}(\bar{x} + a, \bar{y} + b, c) + b_k^{[l]} \tag{3}$$

$$\mathbf{A}^{[l]} = g^{[l]}\left(\mathbf{Z}^{[l]}\right) \tag{4}$$

Where $\times$ denotes the elementwise multiplication and $g^{[l]}$ is the activation function of layer l, $\bar{x} = (x-1)f^{[l]} + s^{[l]}$ and $\bar{y} = (y-1)f^{[l]} + s^{[l]}$ The backward pass is then given by,

$$\partial \mathbf{Z}^{[l]} = \partial \mathbf{A}^{[l]} \times g'^{[l]}\left(\mathbf{Z}^{[l]}\right) \tag{5}$$

$$\partial \mathbf{W}_k^{[l]} = \sum_x^{w^{[l]}} \sum_y^{h^{[l]}} \mathbf{A}^{[l-1]}\left(\bar{x}:\bar{x} + f^{[l]}, \bar{y}:\bar{y} + f^{[l]}, :\right) \times \partial \mathbf{Z}^{[l]}(x, y, k) \tag{6}$$

$$b_k^{[l]} = \sum_i^{w^{[l]}} \sum_j^{h^{[l]}} \partial \mathbf{Z}^{[l]}(i, j, k) \tag{7}$$

$$\partial \mathbf{A}^{[l-1]}\left(\bar{x}:\bar{x} + f^{[l]}, \bar{y}:\bar{y} + f^{[l]}, :\right) = \sum_k^{K^{[l]}} \mathbf{W}_k^{[l]} \times \partial \mathbf{Z}^{[l]}(x, y, k) \tag{8}$$

Where the superscript $'$ denotes the first derivative.

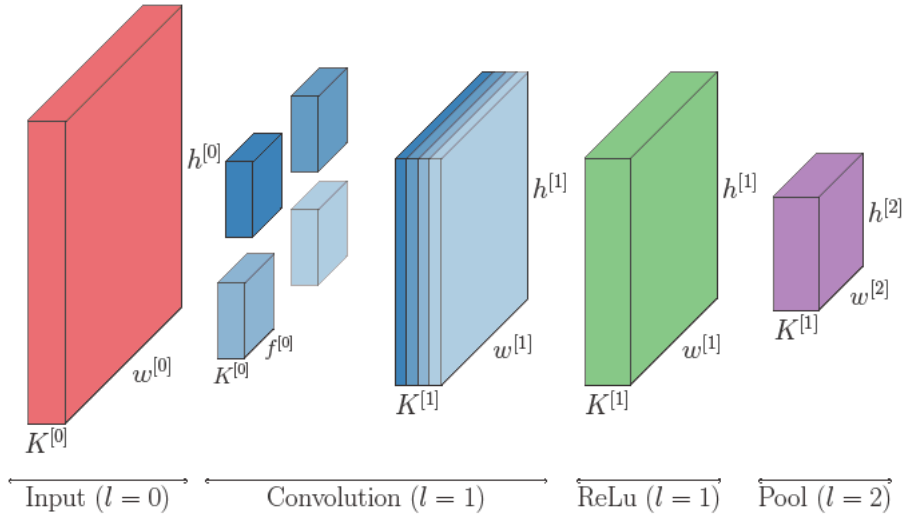

Figure 3: Diagram detailing the dimensions of a simple convolutional neural network with a single convolutional layer followed by a single pooling layer.

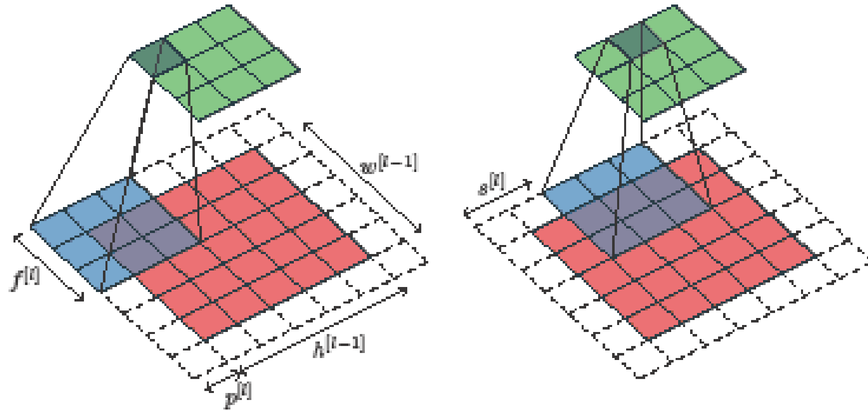

Figure 4: A 2D convolution process

3.2 Pooling layers

Convolutional layers are often followed a pooling layer, which is used to subsample the activation maps, and can dramatically reduce both training and test times. Since pooling layers reduce the number of parameters, they are also used to reduce overfitting. A pooling layer works by partitioning a 2D activation map into a set of non-overlapping rectangles, and applying a non-linear pooling function to each matrix. The pooling function is applied to each dimension separately so that the output of any pooling layer has the same dimension as the input (Figure 3). The most common pooling functions are the maximum or average functions, and this paper will only make use of the maximum,

$$f(x) = \max(x) \tag{9}$$

The size of the output is again controlled by a set of hyperparameters. This paper assumes that each rectangle is a square of size $f^{[l]} = 2$ and the pooling is applied with a stride of $s^{[l]} = 2$ i.e. the maximum function is applied to every 2x2 square in the input so that every neuron in the input is connected to exactly one neuron in the output, as is shown in Figure 5.

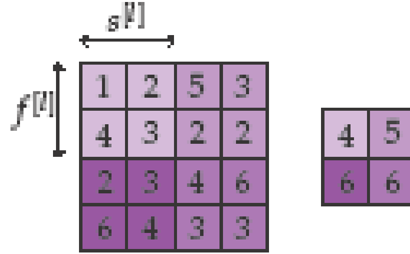

Figure 5: A 2D max pooling operation

A complete forward pass of a max pooling layer is given by,

$$\mathbf{Z}^{[l]}(x, y, k) = \mathbf{A}^{[l]}(x, y, k) = \max_{\forall i \in X, j \in Y} \mathbf{A}^{[l-1]}(i, j, k) \tag{10}$$

Where $X = \{ixs^{[l]}: ixs^{[l]} + f^{[l]}\}$ and $Y = \{jys^{[l]}: jys^{[l]} + f^{[l]}\}$. In practice, a mask is cached during the forward pass so as to remember which neurons in the input were selected by the max function, since only the neurons that pass through the pooling layer contribute to the loss, and only these neurons should be included in the parameter update. The backward pass is then given by,

$$\partial \mathbf{Z}^{[l]} = \partial \mathbf{A}^{[l]} \tag{11}$$

$$\partial \mathbf{A}^{[l-1]}_{\forall i \in X, j \in Y}(i, j, k) = \begin{cases} 0 & \text{if mask}^{[l]}(i, j, k) = 0 \\ \partial \mathbf{Z}^{[l]}(x, y, k) & \text{if mask}^{[l]}(i, j, k) = 1 \end{cases} \tag{12}$$

3.3 Dropout layers

Overfitting is a serious problem in deep learning, due to the vast number of learnable parameters, and an often limited amount of training data. Alongside the common regularisation methods such as L2 regularisation, data augmentation and early stopping,

a common method to deal with overfitting in CNNs is to include a number of dropout layers within a network. The idea behind dropout is to randomly drop (set equal to zero) neurons, with some probability $q^{[l]}$, during training. The purpose of this is to stop the neurons from co-adapting and becoming dependent on one another.

In practice, this is implemented by creating a random binary mask ($\text{mask}^{[l]}$) with the same size as the input $\mathbf{A}^{[l-1]}$. The number of ones and zeros in the mask is $q^{[l]}$ times the total size of $\mathbf{A}^{[l-1]}$. Then, a forward pass is completed by calculating,

$$\mathbf{A}^{[l]} = \mathbf{Z}^{[l]} = \text{mask}^{[l]} \cdot \mathbf{A}^{[l-1]} \tag{13}$$

With corresponding backward pass,

$$\partial \mathbf{Z}^{[l]} = \partial \mathbf{A}^{[l]} \tag{14}$$

$$\partial \mathbf{A}^{[l-1]}(x, y, k) = \begin{cases} 0 & \text{if } \text{mask}^{[l]}(x, y, k) = 0 \\ \partial \mathbf{Z}^{[l]}(x, y, k) & \text{if } \text{mask}^{[l]}(x, y, k) = 1 \end{cases} \tag{15}$$

3.4 Fully-connected layers

Towards the end of the network, the activation map is vectorised ($\mathbf{A}^{[l-1]} \rightarrow a^{[l-1]}$) and passed through one or more fully-connected layers. As with a standard neural network, each neuron in a fully connected layer is connected to every activation in the previous layer via a set of learnable weights $\mathrm{W}^{[l]}$ and biases $b^{[l]}$. The purpose of these layers is to reduce the size of the activation map to that required by the classification or regression output. The forward pass is given by,

$$\mathrm{z}^{[l]} = \mathrm{W}^{[l]}\mathrm{a}^{[l-1]} + b^{[l]} \tag{16}$$

$$\mathrm{a}^{[l]} = g^{[l]}(\mathrm{z}^{[l]}) \tag{17}$$

Whilst the backward pass is given by,

$$\partial \mathrm{z}^{[l]} = \partial \mathrm{a}^{[l]} g'^{[l]}(z^{[l]}) \tag{18}$$

$$\partial \mathrm{W}^{[l]} = \partial \mathrm{z}^{[l]}(\mathrm{a}^{[l-1]})^T \tag{19}$$

$$\partial b^{[l]} = \partial \mathrm{z}^{[l]} \tag{20}$$

3.5 Softmax Regression

The activation function applied to the final fully-connected layer of a multi-class classification R-CNN is usually the Softmax function, such that,

$$\hat{y} = a^{[L]} = \frac{e^{z^{[L]}}}{\sum_c e^{z_c^{[L]}}} \tag{21}$$

Where $\hat{y}$ are the normalised class probabilities. For training purposes the class label (c_j) for a single training example p_j is transformed into a binary vector y_j of length C+1

(where a 1 indicates the correct class). Therefore the the loss function for a single training example ($p_j(x)$,y_j) is given by,

$$\mathcal{L}(\hat{y}_j, y_j) = -\sum_{j=1}^{C} y_j \log \hat{y}_j \tag{22}$$

The corresponding cost function for all n ROIs in all m training examples is the given by,

$$J\left(\mathbf{W}_1^{[1]}, b_1^{[1]}, \ldots\right) = \frac{1}{m+n}\sum_{i}^{m}\sum_{j}^{n}\mathcal{L}\left(\hat{y}_j{}^{(i)}, y_j^{(i)}\right) + \frac{\lambda_c}{2(m+n)}\sum_{l=1}^{L}\sum_{k}^{K^{[l]}}\left\|\mathbf{W}_k^{[l]}\right\|_F^2 \tag{23}$$

Where λ_c is the classification regularisation parameter, and $\| \|_F$ is the Frobenius norm.

3.6 Bounding Box Regression

The bounding box regressor(s) acts separately to the classifier, and is usually connected to the final convolutional, pooling, or dropout layer via a series of fully-connected layers. The final fully-connected layer performs a parameterised transformation of the activation map from the previous layer $a^{[l-1]}$ into an output bounding box $\hat{g}$. Such that,

$$\hat{g}_i{}^x = p_i^w \mathrm{W}^{x^T} a^{[l-1]} + p_i^x \tag{24}$$
$$\hat{g}_i{}^y = p_i^h \mathrm{W}^{y^T} a^{[l-1]} + p_i^y \tag{25}$$
$$\hat{g}_i{}^w = p_i^w \exp\left(\mathrm{W}^{w^T} a^{[l-1]}\right) \tag{26}$$
$$\hat{g}_i{}^h = p_i^h \exp\left(\mathrm{W}^{h^T} a^{[l-1]}\right) \tag{27}$$

Where W* are class specific vectors of length 4. The loss function for a single training example ($p_j(x)$,g_j) is then given by,

$$\mathcal{L}(\hat{g}_j, g_j) = \left\|g_j - \hat{g}_j\right\|_2^2 \tag{28}$$

Where $\| \|_2$ is the L^2 norm. The corresponding cost function for all n ROIs in all m training examples is then given by:

$$J\left(\mathbf{W}_1^{[1]}, b_1^{[1]}, \ldots\right) = \frac{1}{m+n}\sum_{i}^{m}\sum_{j}^{n}\mathcal{L}\left(\hat{y}_j{}^{(i)}, y_j^{(i)}\right) + \frac{\lambda_r}{2(m+n)}\sum_{l=1}^{L}\sum_{k}^{K^{[l]}}\left\|\mathbf{W}_k^{[l]}\right\|_F^2 \tag{29}$$

4. Classification of damage in bolt heads

The first step in designing a damaged bolt detector is to construct and train a convolutional neural network that can classify bolts as damaged or undamaged (as well as classify background images). The main body of the classification CNN is then used to train a bounding box regressor.

One of the difficulties in training a deep CNN, besides the time constraint, is the vast number of hyperparameters, in particular, the architectural hyperparameters. In order to reduce the number of architectural hyperparameters a CNN was constructed as follows. The main bulk of the CNN contains N_b blocks of convolutional layers, with each layer within a block having N_l convolutional layers. Standard practice is to have the number of filters in each convolutional layer increase throughout the network. Therefore, all convolutional layers in the first block contained $K^{[1]}$ filters, whilst all layers in the second block contained $2K^{[l]}$, and all layers in the final convolutional block contained $N_b K^{[l]}$ filters. The parameters N_b, N_l, and $K^{[l]}$ are left as hyperparameters to be tuned.

Within each of the convolutional layers, stride was set to one so that each neuron in the input is connected to the same number of neurons in the output. Moreover, as is common, padding in all convolutional layers is set so that the output volume has the same spatial dimensions as the input, this is commonly referred to as 'same' padding. Also, the filter size in all convolutional layers is set to a single hyperparameter $f^{[l]} = f$ to be tuned.

Each of the convolutional blocks was followed by a max-pooling layer with hyperparameters $f^{[l]} = 2$ and $s^{[l]} = 2$ fixed. Each of the max-pooling layers was followed by a dropout layer with a probability hyperparameter $p^{[l]} = p$ to be tuned. The final convolutional/pool/dropout block was followed by a single fully connected layer which reduced the activation map to a vector of size 3, before applying the softmax function to output the normalised class probability.

The classifier was trained using mini-batch gradient descent with momentum, which updates the parameters with the following update rule:

$$v_{\delta \mathbf{W}} \coloneqq \beta v_{\delta \mathbf{W}} + (1-\beta)\partial \mathbf{W} \quad \text{and} \quad v_{\delta b} \coloneqq \beta v_{\delta b} + (1-\beta)\partial b \tag{30}$$

$$\mathbf{W} \coloneqq \mathbf{W} - \alpha v_{\delta \mathbf{W}} \quad and \quad b \coloneqq b - \alpha v_{\delta b} \tag{31}$$

Where β, the momentum, and α, the learning rate, are also hyperparameters to be tuned. Therefore, the network has a total of 8 hyperparameters, 5 structural $(N_b, N_l, K^{[1]}, f, p)$ and 3 learning $(\lambda_c, \beta, \alpha)$, that were tuned using Bayesian optimisation.

4.1 Bayesian optimisation

In order to tune the set of hyperparameters, the test set, described in section 2, was split randomly into two subsets so as to produce a cross validation and test set. The hyperparameters can then be tuned by comparing the error on the cross validation set. Since the function that maps the hyperparameters to the validation error (or objective function) is unknown, a black box optimisation method is required to minimise the objective function.

Bayesian optimisation works by iteratively constructing a Gaussian function that approximates the mapping from hyperparameters to validation error. Initially, a small number of hyperparameter combinations are evaluated (i.e. the CNN is trained and validation error calculated) before constructing the Gaussian process. The optimum

configuration, based on the current model, is then evaluated before updating the Gaussian process.

The optimisation algorithm was performed over the following hyperparameter space:

$$N_d = \{1:4\}, \quad N_1 = \{1:3\}, \quad f = \{3:7\}, \quad K^{[1]} = 20:60, \quad p = \{0:1\}, \quad \beta = \{0.8:1\}, \quad \alpha = \{0.01:0.001\}, \quad \lambda_c = \{1e^{-5}:1e^{-2}\}$$

Before finding an optimum configuration of:

$$N_d = 3, \quad N_1 = 2, \quad f = 3, \quad K^{[1]} = 38, \quad p = 0.071, \quad \beta = 0.886, \quad \alpha = 0.0043, \quad \lambda_c = 1.16e^{-5}$$

After optimisation, the CNN achieved a classification accuracy of 92.3% on the validation set, and an accuracy of 91.7% on the unseen test set. A selection of the test results can be seen in Figure 6.

Figure 6: A selection of classification test results

5. Detection of damaged bolt heads

The bolt detector shares the feature extraction layers (i.e. convolutional, pooling, and dropout) with the classifier. Therefore, in order to train the bounding box regressors for the damaged and undamaged classes, the learning rate α for these layers was set to zero. A randomly initialised fully connected layer was then attached to the feature extraction layers, which reduced the size of the activation map to 4. This was followed by the bounding box regressor. Therefore, there are no architectural hyperparameters to tune when training the bounding box regressors. However, the learning hyperparameters ($\beta, \alpha,$ and λ_r) must still be tuned using Bayesian optimisation.

The accuracy of an object detector is commonly measured by the mean average precision (mAP) metric, which is given by the mean of the average precision (AP) for each class. Where the average precision for a particular class is the area under the precision-recall curve, and

$$\text{precision}(g, \hat{g}) = \frac{\text{Area of intersecion}}{\text{Area of } \hat{g}} \quad \text{and} \quad \text{recall}(g, \hat{g}) = \frac{\text{Area of intersecion}}{\text{Area of } g} \quad (32)$$

Therefore, the hyperparameters were tuned using Bayesian optimisation to minimise the error in the mAP metric i.e. 1-mAP. Having used the same hyperparameter space for the learning hyperparameters in Eq. (32), the optimum parameters were found to be:

$$\beta = 0.886, \qquad \alpha = 0.0043, \qquad \lambda_r = 1.16e^{-5}$$

After optimisation, the R-CNN achieved a mAP on the validation set of 60.5%, whilst achieving 57.5% on the unseen test set. Whilst the classification results presented in this paper are particularly promising, the regression results require significant improvement. The R-CNN achieved a top-1 classification result of 60.3%, on the much more difficult ILSVRC test set. This is significantly lower than the R-CNN bolt detector. However, it also achieved a mAP of 53.3% on the same test set, a result comparable with the bolt detector. The regression results of the R-CNN damaged bolt detector may be improved by relaxing the constraint on the learning rate α during the regression training.

6. Conclusion

The automated detection of damaged fasteners, such as bolts, has application in many structural health monitoring problems. Recent advances in deep learning for image recognition have led to significant advances in the field. This paper has presented the theory and application of regional convolutional neural networks in a specific SHM context. An R-CNN has been trained and optimised for the purpose of detecting damaged bolts in engineering structures. Moreover, a novel deep dataset has been presented, which combines images from the ImageNet large scale visual recognition challenge with a set of new, specifically acquired, images.

ACKNOWLEDGEMENTS
This research was funded through the EPSRC grant (EP/N018427/1) Autonomous Inspection in Manufacturing and Remanufacturing (AIMaReM).

7. References

[1] A. Krizhevsky, I. Sutskever and G. Hinton, “ImageNet Classification with Deep Convolutional Neural Netowrks,” Advances in neural information processing systems, pp. 1097-1105, 2012.

[2] K. He, X. Zhang, S. Ren and J. & Sun, “Deep residual learning for image recognition,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.

[3] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014.

[4] C. L. Zitnick and P. Dollar, "Edge Boxes: Locating Object Proposals from Edges," Computer Vision-ECCV, pp. 391-405, 2014.

[5] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," Advances in Neural Information Processing Systems 28 , pp. 91-99, 2015.

[6] F. Marino, A. Distante, P. L. Mazzeo and E. Stella, "A real-time visual inspection system for railway maintenance: automatic hexagonal-headed bolts detection.," in IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2007.

[7] J. Yang, W. Tao, M. Liu, Y. Zhang, H. Zhang and H. Zhao, "An efficient direction field-based method for the detection of fasteners on high-speed railways," Sensors, pp. 7364-7381, 2011.

[8] H. Feng, Z. Jiang, F. Xie, P. Yang, J. Shi and L. Chen, "Automatic fastener classification and defect detection in vision-based railway inspection systems," IEEE transactions on instrumentation and measurement, pp. 877-888, 2014.