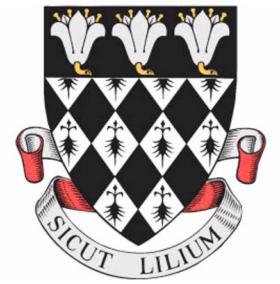


Magdalen College School

Waynflete Project

Object Detection in Thermal Imagery via Convolutional Neural Networks

Euan Ong

(Supervisor: Professor Niki Trigoni)

May 2019

Abstract

During search and rescue missions, firefighters must look for potential victims in burning buildings and bring them to safety as quickly as possible. Lack of visibility can significantly hinder this procedure, forcing firefighters to use thermal cameras to look for possible airways and exits. In this project, we explore the theory behind deep learning and object detection and develop deep learning methods for object detection from forward-looking infrared (FLIR) thermal images, with applications in the development of navigational aids for search and rescue operations. We demonstrate that a Faster R-CNN object detection network can be trained to detect specific objects, such as doors and windows, in RGB images, and then fine tuned to detect the same objects in thermal images. We also determine that performing cross-modality transfer learning on a Faster R-CNN via the fine tuning of only the weights of the feature extractor does not yield a performance increase over the more conventional method of fine-tuning the weights of all layers of the network. However, data augmentation and using multiple datasets in combination provide significant performance improvements on test data.

Contents

1 Introduction

1.1 Machine Learning

Since antiquity, humanity has dreamt of creating machines that think: one could say that artificial intelligence "began with an ancient wish to forge the Gods" [McCorduck, 2004]. From the ancient Greek legends of Hephaestus' golden servant Talos, to Mary Shelley's *Frankenstein*, stories of artificial life abound in our cultural mythos.

Now, with the development of programmable computers, we have been closer than ever before to realising this dream. In the 1950s and 1960s, the burgeoning field of artificial intelligence quickly tackled many problems that human beings find difficult but are comparatively easy for a computer to solve – problems whose solution can be described with a series of mathematical rules [Goodfellow et al., 2016]. The true challenge for this field, however, would be to solve tasks that are intuitive to humans but very difficult to describe in a set of rules or heuristics[1]: tasks such as common-sense reasoning, or recognising objects in images.

Over the last few decades, an explosion in the amount of data available to researchers, combined with the exponential growth of computational power, has given rise to new and immensely powerful methods of automated data analysis, and has given us a solution to these previously intractable problems – that of *machine learning*. The key to this solution is automating the process of finding patterns in data: given a series of examples, by learning the patterns and regularities inherent in these, a machine learning algorithm is able to improve its ability to take actions such as classifying this data into different categories [Bishop, 2007]. A brief overview of some key concepts in machine learning can be found in Appendix A.

1.2 Firefighters and Thermal Imaging

During search and rescue missions, firefighters must look for potential victims in burning buildings and bring them to safety as quickly as possible. Lack of visibility, chiefly due to thick smoke, can significantly hinder navigation.[2] Moreover, the operation controller must be aware of the location of hazards, exits and other points of interest within the building: this process of *semantic mapping* is currently done by memory, with firefighters marking the positions of objects on a floorplan after each search of the building.

To aid their navigation, firefighters use thermal imaging cameras[3]: these are used to find casualties, to locate dangers such as superheated fire gases, wires or furniture, and to look for possible airways and exits (such as doors and windows), both to help clear out the smoke and to provide a safe passage out of the building if needed. Although these are effective in allowing firefighters to search a building more efficiently, objects can still be difficult to discern, particularly those at a similar temperature to their surroundings.

[1] This is known as *Moravec's paradox*: Moravec [1988] writes that "it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility".

[2] In order for firefighters to search buildings for casualties, they perform what is known as the Breathing Apparatus (BA) shuffle [DFRMO, 2014]: they follow a wall with their right hand, swipe their leg to check where to step, move their hand in front of them to check for wires, and circle the room in this way. During this procedure, it is easy for firefighters to become disorientated; it is also difficult for operation controllers to keep track of the locations of firefighters within the building in order to coordinate the rescue.

[3] Thermal imaging cameras detect infrared radiation and render this as a visible light image, as demonstrated in Figure 1.

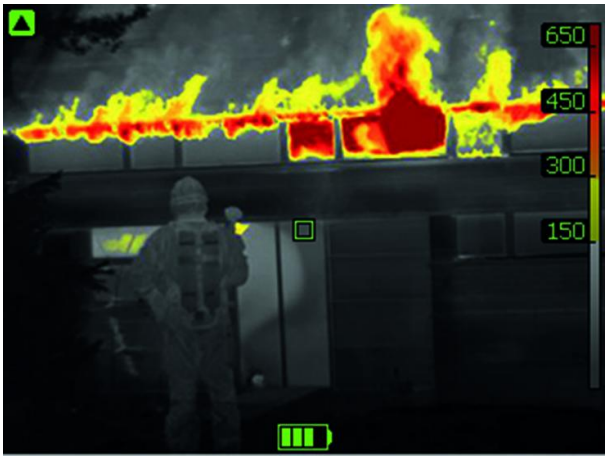


Figure 1: An example of a thermal image, taken with an FLIR thermal camera. Individual luminance values of each pixel are converted to temperatures and rendered as a pseudocolour image. Credit: FLIR

1.3 Contributions

The **aim of this project** is to *develop deep learning methods for object detection from forward-looking infrared (FLIR) thermal images*, with the ultimate goal being to help firefighters, who are equipped with a thermal camera, to locate and navigate to exits in a building, as well as (partially) automating the process of semantic mapping.

We develop a proof-of-concept pipeline demonstrating that an object detection neural network originally trained on RGB images can successfully be trained to detect the same objects in thermal images. For this proof-of-concept, we focus on the problem of detecting objects of the classes `door`, `window`, `table` and `chair` – the first two being relevant for exit detection, and the latter two in hazard avoidance and search and rescue missions. Using two different RGB object detection datasets, we train a Faster R-CNN object detection network [Ren et al., 2017] to label doors, windows, tables and chairs in RGB images. We then demonstrate that such a network can be fine-tuned on a dataset of thermal images labelled by running the previously trained network on corresponding RGB images.

We analyse the benefits of using cross-modality transfer learning, or *modality tuning* [Pratt, 1992, Castrejón et al., 2016], when training the network on thermal images (see Section 4.2 for more details) and determine that this does not yield a significant performance increase over the more conventional method of fine-tuning the whole network.

We also explore various methods of data augmentation, alongside ways to merge annotations from object detection networks trained on different datasets.

1.4 Outline

Section 2 provides a summary of developments in deep learning, neural networks and convolutional neural networks. Section 3 poses the problem of *object detection* and discusses deep learning approaches to object detection. Section 4 provides an outline of the experimental method used. Section 5 presents the the results obtained and our analysis thereof. Finally, Section 6 draws conclusions from the experimental work, and briefly explores directions for further research.

2 Basics of Deep Learning

This chapter introduces key concepts in machine learning relevant to object detection. We discuss the neural network, and explain how problems in computer vision can be approached with the use of convolutional neural networks.

2.1 Neural Networks

The concept of *deep learning*, in contrast to machine learning methods such as logistic regression, refers to the idea of allowing computers to represent the world as a hierarchy of concepts (or levels of abstraction), with higher level concepts being built from combinations of simpler ones. Specifically, if we represent this framework as a graph, showing how concepts are built from one another, this graph has many layers, and is said to be *deep*. [Goodfellow et al., 2016]

The quintessential deep learning model is the *neural network*, otherwise known as a *multilayer perceptron* (MLP).[4] A neural network can be visualised as a *directed acyclic graph* (DAG) like that in Figure 2 – this graph defines how data flows through the network.

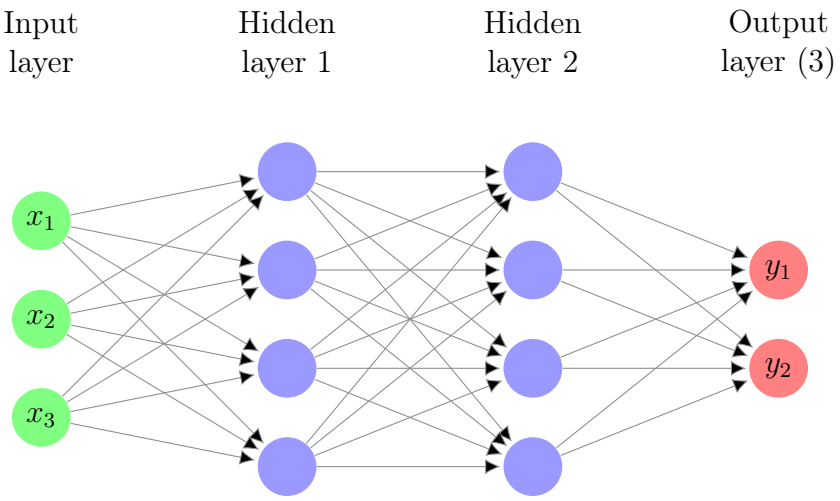


Figure 2: A neural network taking as input a 3-dimensional feature vector $\mathbf{x}$ and returning a 2-dimensional label vector $\mathbf{y}$ as output, with two hidden layers each containing 4 neurons.

2.1.1 Neurons

As can be seen from Figure 2, a neural network consists of a number of layers of *neurons* (here, represented by nodes), with the outputs of each neuron in each layer being con-

[4]The development of the neural network was inspired by attempts to mathematically model the workings of systems of neurons within the brain [Mcculloch and Pitts, 1943, Rosenblatt, 1958], lending it its somewhat biological moniker; it would be incorrect, however, to think of the neural network as an attempt to perfectly model the brain, but rather as a function approximator drawing loose insights from the way in which the brain organises and processes information. Feedforward networks with arbitrarily many neurons can be used to approximate any function with arbitrarily high precision (the *universal approximation theorem* [Hornik, 1991, Lu et al., 2017]); "[they] can be seen as efficient nonlinear function approximators based on using gradient descent to minimize the error in a function approximation." [Goodfellow et al., 2016]

nected to the inputs of all neurons in the subsequent layer (as indicated by the arrows)[5].

For a network with L layers, we will number these layers from 1 to L as in Figure 2. Note that both hidden and output layers contain neurons – the only difference being that the output layer is the last layer of neurons in the network (and so has as many neurons as there are elements in $\mathbf{y}$).

Each of these neurons, the basic building block of a neural network, is in effect a single decision-making unit, and, generally speaking, is responsible for learning a feature representation of some aspect of the data.

One can think of each neuron as a non-linear regressor (see Appendix A.2) defined as follows:

$$y(\mathbf{x},\mathbf{w},b) = f\Big(\sum_{i=1}^{D} w_i\, x_i + b\Big) \tag{1}$$

where

- $\mathbf{x} = (x_i)_{i=1}^{D}$ is a *feature vector* of dimension D whose elements (or *features*) are inputs to the neuron.
- $\mathbf{w} = (w_i)_{i=1}^{D}$ are the parameters (or 'weights') of the neuron, so called as they define how much weight is given to each feature in $\mathbf{x}$ when deciding whether or not the neuron fires.
- The value b is called the *bias*.
- The value of $y(\mathbf{x},\mathbf{w},b)$ is called the *activation* of the neuron (its output) – in other words, whether it fires and, if so, how much.

The non-linear function f is called the *activation function*. (A discussion of activation functions, along with a few examples, can be found in Appendix B.1.)

2.1.2 Inference and Forward Propagation

Inference (see Appendix A) on a neural net is carried out through a procedure called *forward propagation* – setting the value of the input vector and then updating the activations of all neurons based on this. Due to the nature of the graph, this process happens one layer at a time – a propagation of information from the back to the front of the network – hence the name.[6]

[5]Note that the green nodes are not actually neurons but elements of the feature vector $\mathbf{x}$ – this simply indicates that all neurons in the first layer receive as input each of the elements in $\mathbf{x}$. The outputs of the red neurons are the elements of the label vector $\mathbf{y}$, as indicated in the diagram.

[6] Using the above function, and following a standard notational convention (*Stanford Deep Learning Notation* where the superscript $[i]$ denotes the ith layer – https://cs230.stanford.edu/files/Notation.pdf), we can define **update rules** for the activation of a given neuron (the jth neuron in the ith layer):

$$a_j^{[i]} = y(\mathbf{a}^{[i-1]},\mathbf{W}_j^{[i]},b_j^{[i]}) = f\Big(\sum_{k=1}^{D} w_{jk}^{[i]}\, a_k^{[i-1]} + b_j^{[i]}\Big) = f\Big(\mathbf{W}_j^{[i]}\,\mathbf{a}^{[i-1]} + b_j^{[i]}\Big) \tag{2}$$

where

- $\mathbf{a}^{[i]}$ is a vector of length $n_h^{[i]}$ (the number of neurons in the ith layer); the jth element, $a_j^{[i]}$, is the activation of the jth neuron in the ith layer.
- $\mathbf{W}^{[i]}$ is a matrix of dimensions $n_h^{[i]} \times n_h^{[i-1]}$, where the (j,k)-element of the matrix, $w_{jk}^{[i]}$, is the

2.1.3 Training and Backpropagation

As explained in Appendix A.2.1, the goal of training is to minimise some loss function $\ell = \ell\big(\mathbf{X}, \mathbf{y}; (\mathbf{W}^{[i]})_{i=1}^{L}, (\mathbf{b}^{[i]})_{i=1}^{L}\big)$, a measure of how close the network's predictions $\mathbf{y}*$ on each training example $\mathbf{x}$ in the training set $\mathbf{X}$ are to the actual labels (*ground truth*) $\mathbf{y}$. As there is in general no analytical / closed form solution to this (and even if there were, finding the global minimum of the loss function often leads to overfitting [Choromanska et al., 2015]), we can numerically find a (local) minimum using *stochastic gradient descent* (see Appendix A.2.2).

In order to use stochastic gradient descent to train a neural network, we need to be able to compute the gradient vector $\nabla \ell_{\mathbf{x}}$ – in other words, the partial derivatives of ℓ with respect to all the weights and biases in the network – for a given training example $\mathbf{x}$.

The *de facto* method for doing this is known as *backpropagation* [Rumelhart et al., 1988]. This algorithm makes use of two key insights, the first of which being that we can represent the process of forward propagation, the update rule in (3), as a series of nested functions – for instance (in a network with 4 layers):

$$\mathbf{y} = \mathbf{a}^{[4]} = f^{[4]}(\mathbf{W}^{[4]} f^{[3]}(\mathbf{W}^{[3]} f^{[2]}(\mathbf{W}^{[2]} f^{[1]}(\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]}) + \mathbf{b}^{[3]}) + \mathbf{b}^{[4]}) \tag{4}$$

Now, finding the partial derivatives of ℓ with respect to $\{(\mathbf{W}^{[i]})_{i=1}^{L}, (\mathbf{b}^{[i]})_{i=1}^{L}\}$ is a matter of repeated applications of the chain rule to the above functions.

The second insight is that of using *memoisation*: in other words, computing derivatives of weights from the front of the network to the back and storing these derivatives in the process, so they do not have to be calculated multiple times. This allows us to achieve a time complexity of $O(W)$, where W is the number of weights in the network. This is favourable compared to alternative methods of differentiation, such as that of *finite differences* (perturbing each weight and monitoring how the error changes to approximate the gradient), with time complexity $O(W^2)$ [Bishop, 2007].

2.2 Convolutional Neural Networks

Convolutional neural networks, or CNNs, were first popularised by LeCun et al. [1989], and were designed to fix a number of issues with standard feedforward neural networks when used on a specific type (*modality*) of data – that of (large) images. To provide some motivation for the architecture of the convolutional neural network, Appendix B.2

---

weight in the jth neuron in the ith layer associated with the activation of the kth neuron in the $(i-1)$th layer. (In the above we write $\mathbf{W}_j^{[i]}$ as the jth row of the matrix $\mathbf{W}^{[i]}$.)

- $\mathbf{b}^{[i]}$ is a vector of length $n_h^{[i]}$, where the jth element, $b_j^{[i]}$, contains the bias in the jth neuron in the ith layer.

We can naturally extend this to an update rule for $\mathbf{a}^{[i]}$ as a vector:

$$\mathbf{a}^{[i]} = f^{[i]}(\mathbf{W}^{[i]}\,\mathbf{a}^{[i-1]} + \mathbf{b}^{[i]}) \tag{3}$$

So to carry out inference on the network, one need only set $\mathbf{a}^{[0]}$ to the feature vector $\mathbf{x}$ and compute $\{\mathbf{a}^{[i]}\}_{i=1}^{L}$ using (3), and the label vector $\mathbf{y}$ is $\mathbf{a}^{[L]}$.

contains a brief example of training a feedforward neural network[7] on a well-known image dataset.

2.2.1 Convolutional Layer

The convolutional layer is the key building block of the convolutional neural network. It takes as input a 3-dimensional input volume (width × height × depth) – this is either an image or a set of stacked *feature maps* (as explained below).

Each convolutional layer consists of a series of *kernels*, or learnable filters. These are rank-3 tensors that are small spatially but have a depth equal to the depth of the input: for instance, for a 3-channel RGB image, a kernel might have dimensions $[5 \times 5 \times 3]$. The size of a kernel is otherwise known as its *receptive field*, and is a hyperparameter we can modify.

In forward propagation, we slide the kernel across the width and height of the input volume[8], at each position taking the dot product of the values in the kernel and the corresponding values in the input, as shown in Figure 3. We apply an activation function f, most commonly ReLU (see Appendix B.1) element-wise to the resulting matrix. This creates what is known as a *feature map*, containing $f(S(i,j))$ for $1 \leq i \leq H$, $1 \leq j \leq W$. From an intuitive perspective, these filters generally learn to detect a given feature in the image, such as an edge, or a contrast in colour, hence the name. A demonstration of this is given in Figure 4.

Notice that the properties of the kernel (shared weights and locality) are designed to exploit the spatial structure of images – specifically, the fact that features in images can be found anywhere within the image (translation equivariance) and are usually connected (i.e. comparing pixels that are close together is more relevant than comparing pixels that are far apart).

There are a few more hyperparameters[9] associated with a convolutional layer, which can determine the dimensions of its output.

[7] While convolutional neural networks are themselves a subset of feedforward neural networks, in that they are acyclic, we will use the term to refer to networks consisting only of 'standard' neurons, as described in Section 2.1.

[8] This is equivalent to performing a 2D discrete convolution of the image and the kernel, as follows: [Goodfellow et al., 2016]

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n K(m,n)\, I(i-m, j-n) \tag{5}$$

where $K(m,n)$, $I(m,n)$ are defined as K_{mn}, I_{mn} for valid / 'in-range' values of m, n and 0 otherwise.

[9] The depth of the output layer is determined by the *number of filters* in the layer – the output effectively consists of the feature maps of each filter stacked on top of each other. The width and height of the output are also determined by what is called the *stride* of the layer. As, particularly for large images and large networks, calculating so many dot products of kernels can become computationally expensive, we may want to skip over some possible positions of the kernel, reducing computational cost at the expense of less detailed feature extraction. In effect, this is equivalent to downsampling our feature map. The stride, therefore, is the number of pixels we skip (in either direction) between each placement of the kernel.

A further modification that is sometimes used is *zero-padding*: as in its current form the convolution layer will always return a feature map of smaller dimensions than its input, to retain the dimensions of the input one might choose to 'pad' the input with zeroes, such that the top left corner of the kernel can be placed outside the image. The width of this padding forms another hyperparameter of the layer.

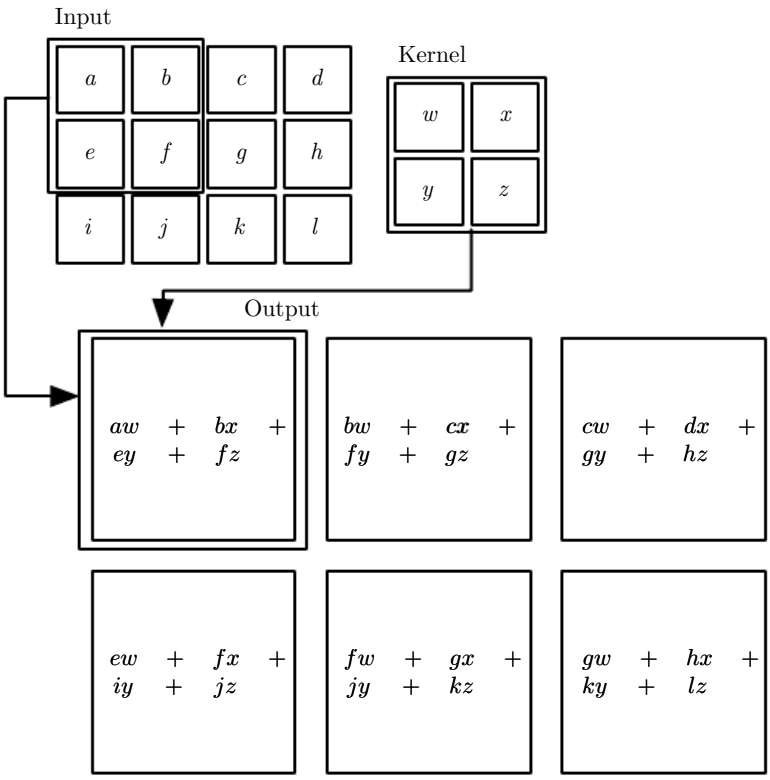


Figure 3: An example of convolution of a 2×2 kernel over a 3×4 matrix; at each position we take the dot product of the kernel and the corresponding values in the input to yield a value in a 2×3 feature map, as shown above. [Goodfellow et al., 2016]

2.2.2 Pooling Layer

The purpose of the pooling layer is to *downsample* and reduce the spatial dimensionality of the input volume, which results in a reduction of the number of parameters in the network. The pooling layer can prevent overfitting; it can also act to aggregate multiple low-level features occurring in a small neighbourhood [Scherer et al., 2010].

Although there are a few variants of the pooling layer, the most common is the *max-pooling* layer. This takes a filter of a given dimension $F \times F$ (most often 2×2), and a stride normally of the same length, and for each possible subregion the filter convolves around, returns the maximum value of all elements in that subregion (see Figure 5).

The max-pooling layer acts on each depth slice independently, producing an output with the same depth as the input, but with smaller width and height. Note that the max-pooling layer has no trainable parameters – it computes a fixed function of its input.[10]

2.2.3 Structure

A typical convolutional neural network consists of a sequence of three types of layer: the two described above (*convolutional* and *pooling* layers), and the standard feedforward, or *fully connected* layer as detailed in section 2.1.1.

The most common structure for a convolutional neural network is to use a number

[10]Some studies [Springenberg et al., 2015], however, have suggested that pooling layers are not necessary for a successful convolutional neural network, suggesting that the occasional *Conv* layer with a larger stride be used instead for the purpose of spatial dimensionality reduction.

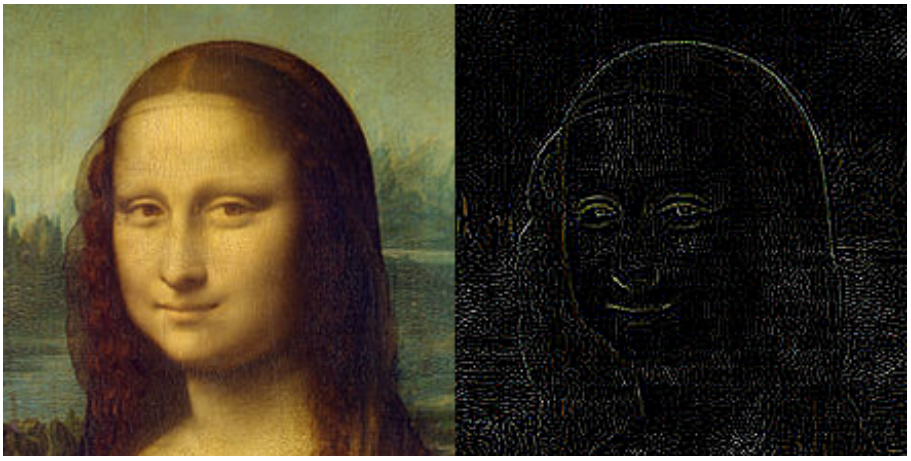


Figure 4: An example of such a filter is the following kernel:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Otherwise known as the *discrete Laplace operator*, it is commonly used in image processing to create an edge detection feature map like that shown in Figure 4.
This figure demonstrates the effect of the discrete Laplace operator (Laplacian kernel) when applied to an image, produced using Mathematica. Notice that regions of high change in intensity (high spatial second derivative) are highlighted in the processed image.

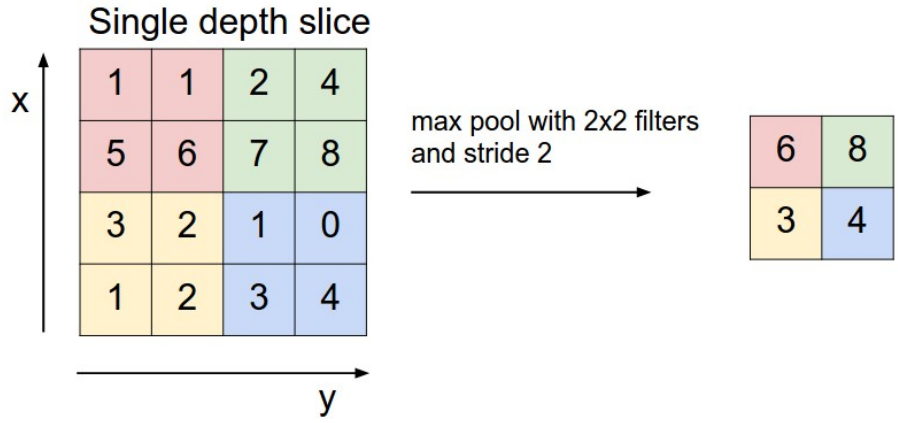


Figure 5: A max-pooling layer with 2×2 filters and stride 2, acting on a $4 \times 4 \times 1$ input. This discards 75% of the input. [Karpathy, 2019]

of 'stacks' consisting of a few convolutional layers connected in sequence for feature extraction, followed by a max-pooling layer for spatial dimensionality reduction, until the dimensions of the feature maps have been significantly reduced. At this point, one tends to transition to a few fully connected layers, with decreasing numbers of neurons – the output of the last fully connected layer forms the output of the network (e.g. class probabilities). In cases where the network is used for classification, the last layer is often a *softmax* layer, which has the effect of mapping the non-normalised output of a network to a probability distribution over the predicted output classes.

A well-known example of a CNN is the *VGG-16* network (see Figure 6).

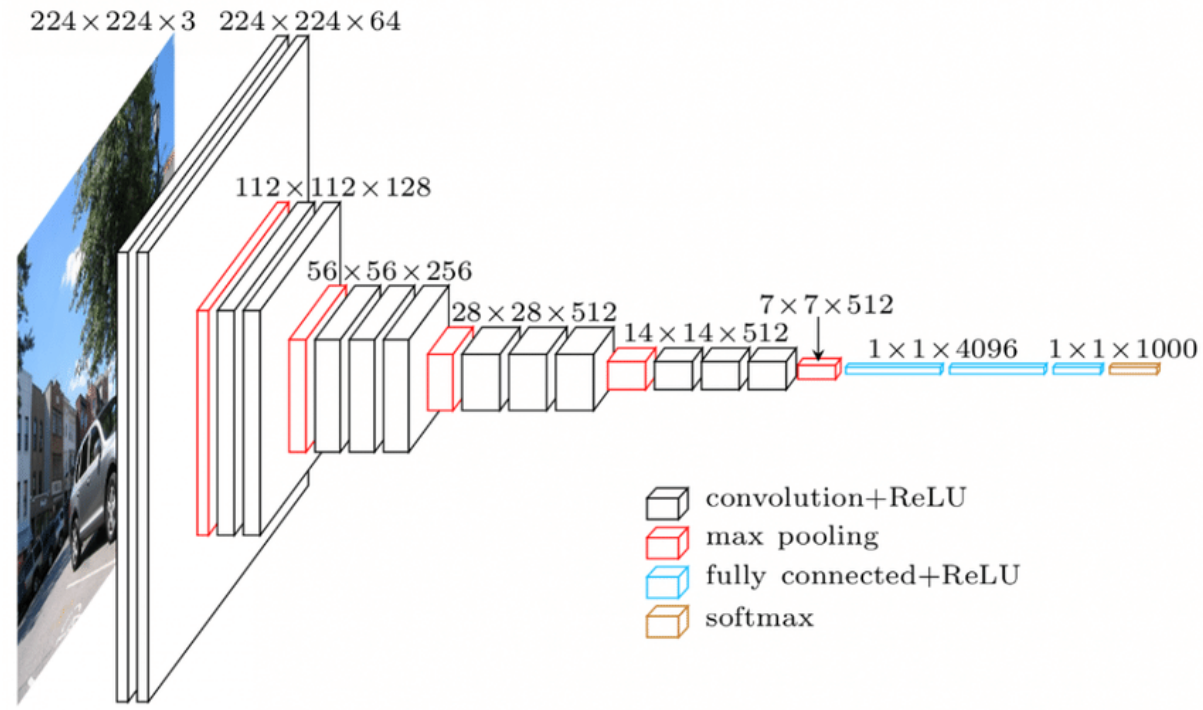


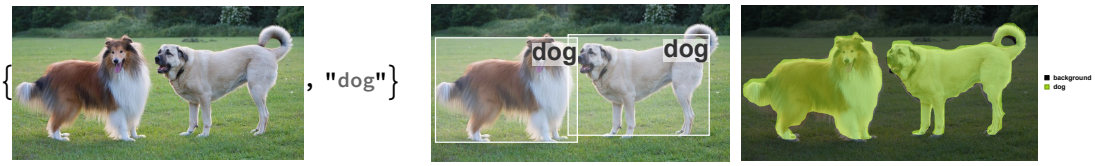
Figure 6: A diagram of the architecture of the VGG-16 convolutional neural network [Loukadakis et al., 2018]. Dimensions are specified as *width*×*height*×*depth* (number of feature maps in the layer).

The VGG-16 network secured second place in the classification track for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014. Developed with the aim of exploring the impact of *network depth* on accuracy in classification problems using convolutional networks, Simonyan and Zisserman [2015] discovered that a significant improvement on prior state of the art can be achieved simply by increasing network depth to 16-19 layers. This network remains popular today due to its uniform architecture – using only 3×3 and 1×1 convolutional filters, it consists of a series of convolutional layers of increasing depth and decreasing spatial dimensions, followed by three fully connected layers (as shown in Figure 6). Notice also the stacks of three 3×3 convolutional layers, followed by a max-pooling layer. These stacks have an effective receptive field of 7×7. They offer increased performance, however, compared to an equivalent layer with 7×7 convolutional kernels – this is due to a 45% reduction in the number of parameters and the fact that, by stacking three layers, we include three non-linear (ReLU) functions instead of just one, making the decision function more discriminative.

3 Object Detection

3.1 Classification, Detection and Segmentation

The previous section is about *image classification* (Figure 7a): given an image, sort it into one of a number of categories, or *classes*, based on, for instance, the main object(s) in the image. An example would be a network that classifies an image as either a dog or a cat. The problem we now need to solve, *object detection* (Figure 7b), is more complex, and involves two subtasks: object localisation (identifying regions in an image which could contain an object, and predicting the coordinates of *bounding boxes* surrounding these regions), and object classification (identifying the object within the bounding box). A more difficult task is that of *semantic segmentation* (Figure 7c) – this is the process of labelling each pixel in the image based on its context (for instance, if a pixel is part of a dog, label it as such).



(a) Image Classification (b) Object Detection (c) Semantic Segmentation

Figure 7: An example illustrating different problems in computer vision. Image taken from [Venzel, 2016].

3.2 Evaluation Metrics

One of the most popular sets of object detection evaluation metrics are what are known as *COCO metrics*: the evaluation metrics used in the COCO[11] object detection challenge. The primary metric used here is what is known as *mAP@IoU=.50:.05:.95* – the mean of average precision values with IoU threshold varying from 0.50 to 0.95 with interval 0.05. Other common metrics include *mAP@IoU=.50* (Pascal VOC metric) and *mAP@IoU=.75*.

For the COCO dataset, the current state of the art is an mAP of 0.526, achieved by the Chinese AI startup Megvii [Lin et al., 2018]. A study by Google [Huang et al., 2016] analysing speed/accuracy tradeoffs in various object detection architectures for the COCO dataset yields mAPs as shown in Figure 8; the TensorFlow object detection 'model zoo' [Huang et al., 2019] contains COCO-trained models with mAPs ranging from 0.16 to 0.43.

For a systematic treatment of evaluation metrics, see Appendix C.

3.3 Deep Learning Methods

3.3.1 Localisation and Regression

To understand deep learning approaches to object detection, let us first consider how we would solve a simpler problem: that of *classification and localisation* (i.e. given an image, detect the main object in it and surround it with a bounding box). This is both a classification (image as input, class as output, cross-entropy loss) and regression (image

[11] Common Objects in Context [Lin et al., 2014]

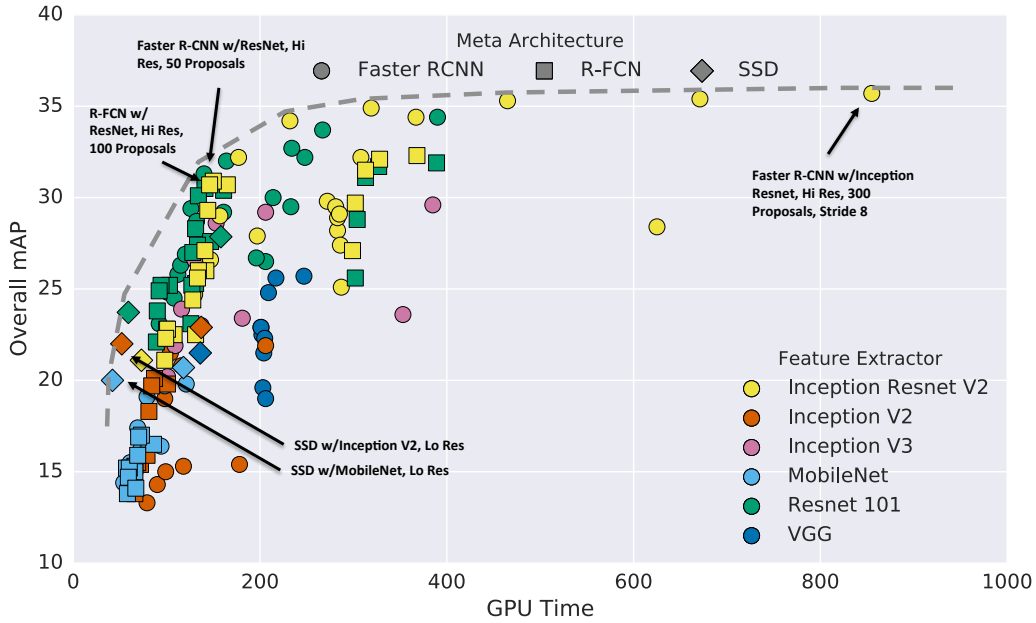


Figure 8: A graph of accuracy (percentage mAP) against per-image inference time. Colours indicate feature extractor; marker shapes indicate meta architecture. Each (meta architecture, feature extractor) pair can correspond to multiple points on the graph due to varying hyperparameters. [Huang et al., 2016]

as input, bounding box vector (x,y,w,h) as output, regression losses e.g. L_1[12], L_2[13]) task. Thus one way to tackle the problem would be to use a pre-existing convolutional neural network (e.g. VGG-16), to attach a 'regression head'[14] to the final convolutional layer and to train the regression head with L_1 or L_2 loss. This only works, however, in the case of single-object detection: in the case of multi-object detection, we cannot simply use regression as we could have arbitrarily many bounding boxes in our image, and such a network has only a fixed number of outputs.

3.3.2 Detection as Classification and R-CNN

Although this may seem counter-intuitive, we can instead try framing the problem of detection as one of *classification*: given a set of possible bounding boxes (bounding box priors), classify the object contained within the general region of each box as either one of a set of object classes or as a non-object (the 'background' class). Thus we can decompose detection into two subproblems:

1. **Region proposal**: Suggesting possible bounding boxes that could contain an object – *regions of interest*, or RoIs.

2. **Object identification**: Out of these bounding boxes:

   (a) Classifying each bounding box as containing one of a list of given objects, or as a non-object (background).

   (b) Performing further regression to give a more accurate prediction of the bounding box than the proposed RoI.

[12] L_1 loss: $E(\mathbf{X} \mid \mathbf{w}) = \frac{1}{n}\sum_{i=1}^{N}|y(x_i,\mathbf{w}) - y_i|$
[13] L_2 loss: $E(\mathbf{X} \mid \mathbf{w}) = \frac{1}{n}\sum_{i=1}^{N}(y(x_i,\mathbf{w}) - y_i)^2$
[14] a series of fully connected layers outputting four scalar values (x,y,w,h)

One way to approach (1) is to use classical image processing techniques to detect possible object candidates (e.g. 'blob-like' regions within the image) such as *selective search* [Uijlings et al., 2013]. This is the method used by the *Regions with CNN Features* (R-CNN) object detection network developed by Girshick et al. [2014]. This system consists of three modules, illustrated in Figure 9: a domain-independent region proposal generator, a convolutional neural network feature extractor (AlexNet), and a set of class-specific *support vector machines* (SVMs) (see e.g. [Bishop, 2007, Ch. 7]) and bounding box regressors.

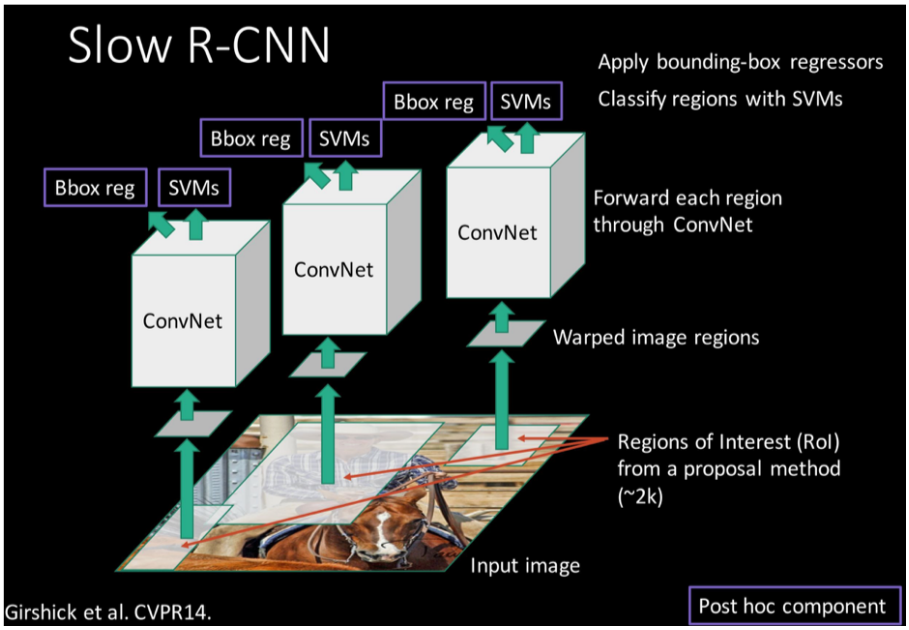


Figure 9: The R-CNN architecture: regions of interest are extracted using selective search, forwarded through a CNN (e.g. AlexNet) and classified with SVMs. Coordinates of bounding boxes are fine tuned with bounding box regressors. [Girshick et al., 2014]

The R-CNN model can be trained by performing the following steps:

1. Train an image classification CNN (or take a pre-trained network) on the target classes.
2. Reinitialise the final fully connected layer, add a 'background' class and fine-tune this network on positive (in a bounding box) and negative (outside a bounding box) regions from the ground truth.
3. For each image, run the region proposal generator to extract RoIs, and run each through the CNN – save each of the features generated (the 4096-dimensional feature vector from the last max-pooling layer) to disk.
4. Train a binary SVM for each class to classify the extracted features.
5. Train a regressor to predict normalised offsets in order to refine the rough bounding box predictions given by the proposed RoI.

At test time, for each image, around 2000 region proposals are extracted using selective search, each region is warped to 224×224 pixels and fed into the CNN in order to extract a 4096-dimensional feature vector. Each feature vector is then scored using the class-specific SVMs, and overlapping bounding boxes are suppressed using a technique called *non-maximum suppression*.

3.3.3 Optimisation 1: Fast R-CNN

A solution for some of the inefficiencies[15] in the R-CNN network was proposed by Girshick [2015], in the form of the aptly named *Fast R-CNN* object detection network. The main innovation of this architecture is that it swaps the order of the CNN feature extractor and region proposal: instead of proposing ∼2000 regions which each individually need to be passed through the CNN, the whole image is processed once by the CNN *before* features are extracted (see Figure 10).

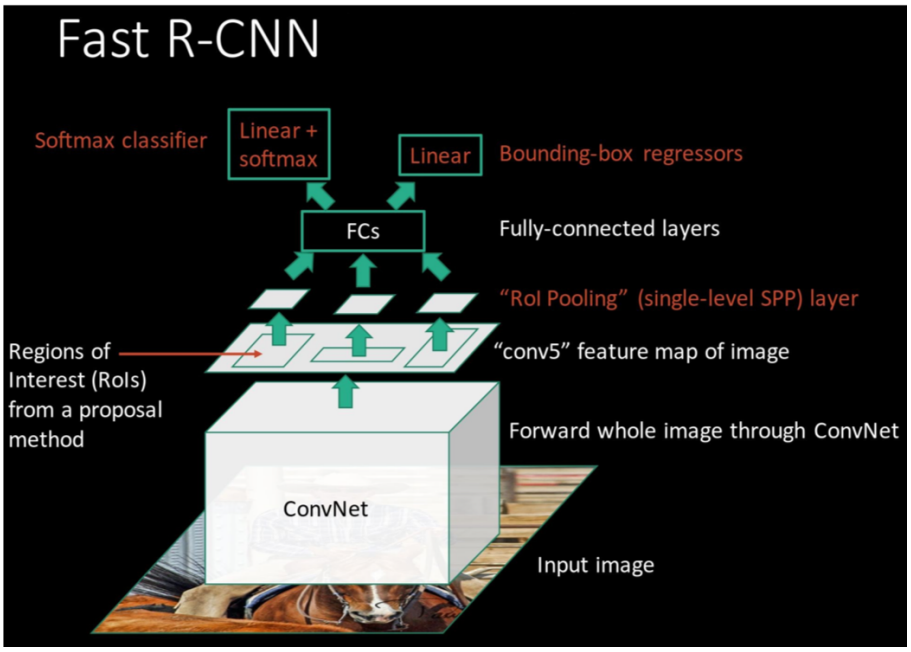


Figure 10: The Fast R-CNN architecture [Girshick, 2015]. The network pipeline is as follows:

1. The image is forwarded through a CNN (with all layers from the last max-pooling layer onwards removed) yielding a feature map of the image.
2. RoIs are generated using e.g. selective search.
3. An RoI pooling layer extracts fixed-size feature maps from the output of the CNN for each RoI. In this layer, each RoI acts as a 'window' onto a particular section of the feature map; this window then uses max-pooling to output a fixed dimension feature map for the given RoI.
4. These feature maps are passed through fully connected layers which return feature vectors.
5. Feature vectors are passed to a softmax classifier (for class prediction) and bounding box regressors (for fine tuning coordinates of bounding boxes).

[15] The two main drawbacks of the R-CNN network are as follows:

1. The network performs a CNN forward pass for each individual object proposal. This makes training and inference very expensive in terms of space and time: when training, for each image, ∼2000 RoIs must be extracted with selective search, run through the CNN and the corresponding 4096-dimensional feature vectors saved to disk, and when testing, features must be extracted from each RoI in the image in the same way and then individually passed to SVMs.
2. The network is a pipeline formed by three independent models (CNN, SVM, bounding box regressor), each of which must be trained independently.

3.3.4 Optimisation 2: Faster R-CNN

While this architecture is much faster than R-CNN (9× faster than R-CNN for training; 213× faster for inference), there is still one bottleneck in terms of efficiency: the generation of region proposals. While Fast R-CNN achieves near real time detection rates (∼0.2 seconds per image [Girshick, 2015]), this is only *when ignoring the time spent on region proposals*. Commonly used region proposal algorithms, such as selective search, take up to two seconds per image [Uijlings et al., 2013], an order of magnitude slower.

The logical solution to this problem is, yet again, computation sharing: the *Faster R-CNN* network developed by Ren et al. [2017] does exactly this. Instead of region proposal algorithms such as selective search, they use a *region proposal network* (RPN) that takes as input the feature map produced by the CNN and returns a series of region proposals (see Figure 11).

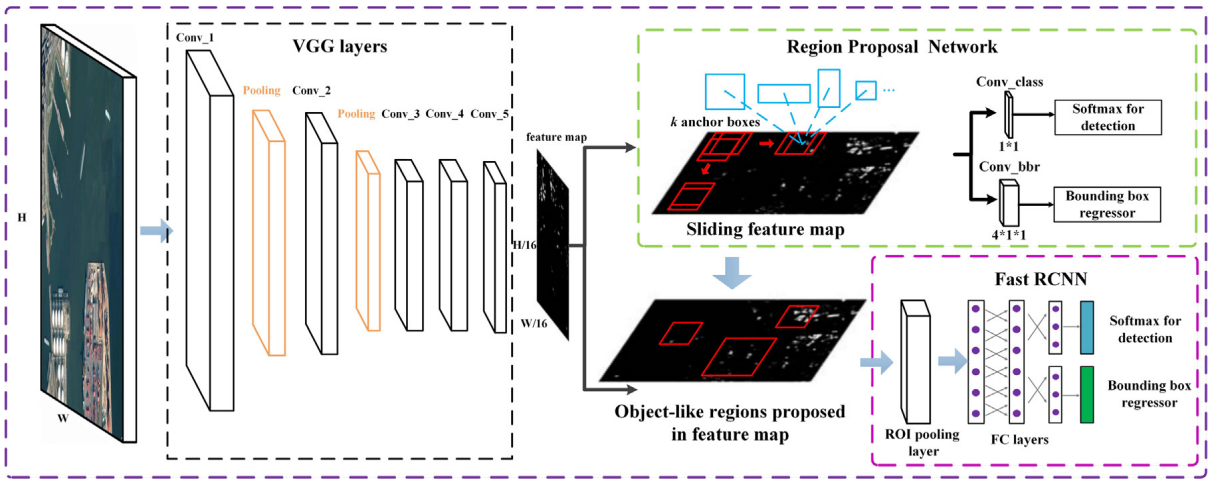


Figure 11: A diagram illustrating the architecture of the Faster R-CNN network (with a VGG feature extractor) [Deng et al., 2018]. The network pipeline is as follows:

1. As before, the image is forwarded through a CNN (with all layers from the last max-pooling layer onwards removed) yielding a feature map of the image.
2. RoIs are extracted using the region proposal network. A spatial window of fixed dimensions $n \times n$ is convolved over the feature map with a fixed stride, much like in a convolutional neural network. At each position, the sliding window is mapped to a lower dimensional feature with the use of a $n \times n$ convolutional layer. This feature is fed into two other fully connected layers, one of which being a classification layer (predicting the class of the object within the window – either one of the object classes, or 'background'), and the other being a regression layer (predicting coordinate offsets for the bounding box). For each position of the window, the RPN simultaneously predicts k multiple region proposals, with the coordinates of each proposal being relative to one of k given *anchor boxes* (depicted in blue). Each anchor box is associated with a different scale and aspect ratio. These help the network to detect objects at different scales and dimensions.
3. These RoIs, together with the feature map, are passed to the RoI pooling layer, at which point detection proceeds as in Fast R-CNN.

4 Methods for Detecting Objects in Thermal Images

This chapter discusses the problem of thermal image object detection, within the context of emergency response situations. A deep learning solution is proposed, and a general method is outlined.

4.1 Oxford Thermal Dataset (OxTD)

The dataset provided for this project, the *Oxford Thermal Dataset* (OxTD), consists of 158,517 pairs of corresponding RGB and thermal images, captured in 21 different locations ('runs') around the University of Oxford (Figure 12a).[16]

The RGB images each have dimensions 848×480, and are 3-channel images with 24-bit colour depth (the R, G and B values of each pixel range from 0 to 256). The thermal images each have dimensions 464×348, and are 1-channel images with 16-bit colour depth (each pixel in the image takes a single value between 0 and 65535).[17]

(a) An example image

(b) An image exhibiting significant motion blur

Figure 12: 848×480 images taken from the Oxford Thermal Dataset.

[16]Data for the **Oxford Thermal Dataset** were collected by University of Oxford staff as they walked through the buildings, using an RGB-D camera attached by a 3D-printed mount to an FLIR E95 thermal imaging device, at a rate of around 60 frames per second.

As the RGB images were not captured using the RGB camera on the FLIR E95, but instead on a separate camera (attached securely to FLIR) connected to a microcontroller, at times there is loss of synchronisation between the capturing of the thermal and RGB images leading to a lag of up to a few seconds in some cases. In addition, during recording, the camera was handheld; this introduced artifacts such as motion blur into the images (Figure 12b).

After some analysis, two of the 21 runs in the dataset took place in locations containing very few objects of interest besides empty shelves, and one run was duplicated. In addition, one run displayed a particularly high level of latency between RGB and thermal images. These runs were removed from the dataset, leaving 119,078 RGB-thermal pairs.

[17]These raw values are a measure of the amount of radiation hitting each pixel in the sensor, and can be converted to estimated temperatures using standard equations in infrared thermography. Accurate estimation of temperature requires certain calibration constants and other metadata captured by the thermal camera, which was unavailable in the dataset. Some brief testing suggests that using estimated temperature values would be no more effective than using the raw radiance values, as we are only interested in distinguishing objects from each other – a task which depends on the relative differences in value of pixels. [Minkina and Dudzik, 2009].

4.2 Problem Analysis and Transfer Learning

We shall use *transfer learning* to construct a network that can detect objects of the relevant classes in thermal images – specifically images from the OxTD.

Transfer learning is a machine learning method where a model trained to perform a particular task is repurposed, or *fine tuned*, in order to perform a similar task better than if a new model had been trained from scratch. In terms of computer vision, this often entails pre-training a network on a very large dataset (e.g. COCO or Pascal VOC) and either using this network as a feature extractor for the task of interest, or using its weights as the initial weights for a new network. The weights of this network can then be *fine-tuned*, by continuing the training process and backpropagation as normal on the new dataset. (For a fuller explanation of transfer learning, see Appendix D.)

A study by Castrejón et al. [2016] demonstrated the possibility of performing *cross-modal*[18] transfer learning with CNNs. In contrast to conventional transfer learning methods, where the *task-specific* last layers of the network are fine tuned and the rest frozen, they instead fine tune the earlier, *modality-specific* layers of the network and keep the last few layers frozen. In this way, they approached the task of scene recognition (given an image, determine the environment in which it was taken) across the modalities of natural images, sketches, clip art, spatial text[19] and descriptions, by learning low level representations specific to each modality, and a high level representation shared across all modalities (see Figure 13).

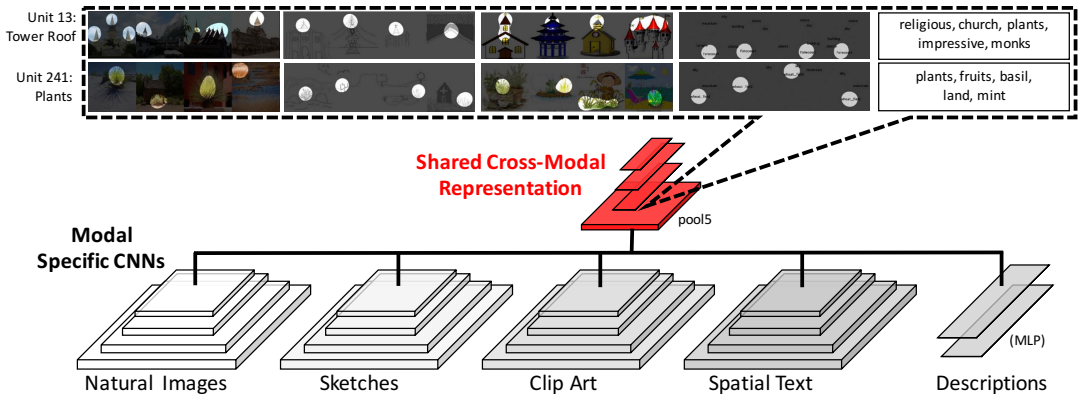


Figure 13: A diagram of the architecture used by Castrejón et al. [2016]. Above, objects and parts of the scene that activate specific neurons the most are highlighted – interestingly, there are neurons in the higher level layers which tend to fire on the same objects regardless of modality.

In this vein, as we have many data and networks trained on the RGB modality and comparatively little thermal data, we hypothesise that it is possible to perform what Castrejon et al. refer to as *modality tuning* on the thermal dataset: if we train a network to detect objects in RGB images, we should be able to freeze[20] the last few modality-agnostic layers and fine-tune the first few modality-specific layers to allow the network

[18] A *modality* refers to the way in which information is represented through data – 'the way in which something happens or is experienced' Baltrusaitis et al. [2017]. These include, for instance, natural language and vocal signals. In the context of computer vision, RGB images, thermal images and drawings are three distinct modalities through which visual information can be conveyed.

[19] Spatial text refers to words describing objects, with each word being associated with the coordinates of the object it refers to.

[20] *Freezing* layers refers to keeping the weights of these layers constant, and not modifying them during training.

to detect objects in thermal images.

4.3 Overview of the Four-stage Training Pipeline

The training pipeline consists of the following stages:

1. ***Test set creation***. Calculate mappings between RGB and thermal image coordinates for each run, and create a test set of around 100-200 images in both RGB and thermal modalities, hand annotated with bounding boxes.

2. ***Fine tuning on RGB images***. Identify a suitable network architecture to use, and fine tune this network on RGB images containing the categories: doors, windows, tables, and chairs.

3. ***Inference and labelling on training and validation sets***. Use this network to annotate (a subset of) the RGB images in the OxTD with bounding boxes for these categories, and map these bounding boxes to their thermal image equivalents.[21]

4. ***Modality tuning on thermal images***. Perform modality tuning on this network: freeze the higher-level layers and train it on the bounding boxes for the thermal images.

All experiments were performed in *TensorFlow* [Abadi et al., 2015]; specifically, the *TensorFlow Object Detection API*, a high-level framework for training, inference and evaluation of object detection neural networks. Networks were trained on an NVIDIA Titan V with 12GB memory. Pending approval, further implementation details and code may be available at `https://homegrownapps.tk/objectdetection`.

Figure 14: The 4-stage Training Pipeline

4.4 Stage 1: Test Set Creation

A flowchart for this stage is provided in Figure 15. A discussion of the data processing methods used in this stage can be found in Appendix E.1.

4.5 Stage 2: Fine Tuning on RGB Images

A flowchart for this stage is provided in Figure 16.

[21] In order to perform transfer learning on a network, we must not only prepare a network that can detect the relevant objects in RGB images, but we also require labelled thermal training data with which to perform modality tuning on the RGB network. We can solve both of these problems by training a variety of networks to detect objects in RGB images, and using these networks to annotate the RGB images of each RGB-thermal pair in the OxTD. Now, as each RGB image is paired with a thermal image, we can compute a translation mapping coordinates on an RGB image to the respective coordinates on a thermal image. Thus we can map the coordinates of the bounding boxes produced to their equivalent coordinates in the thermal images, allowing us to generate a labelled thermal dataset.

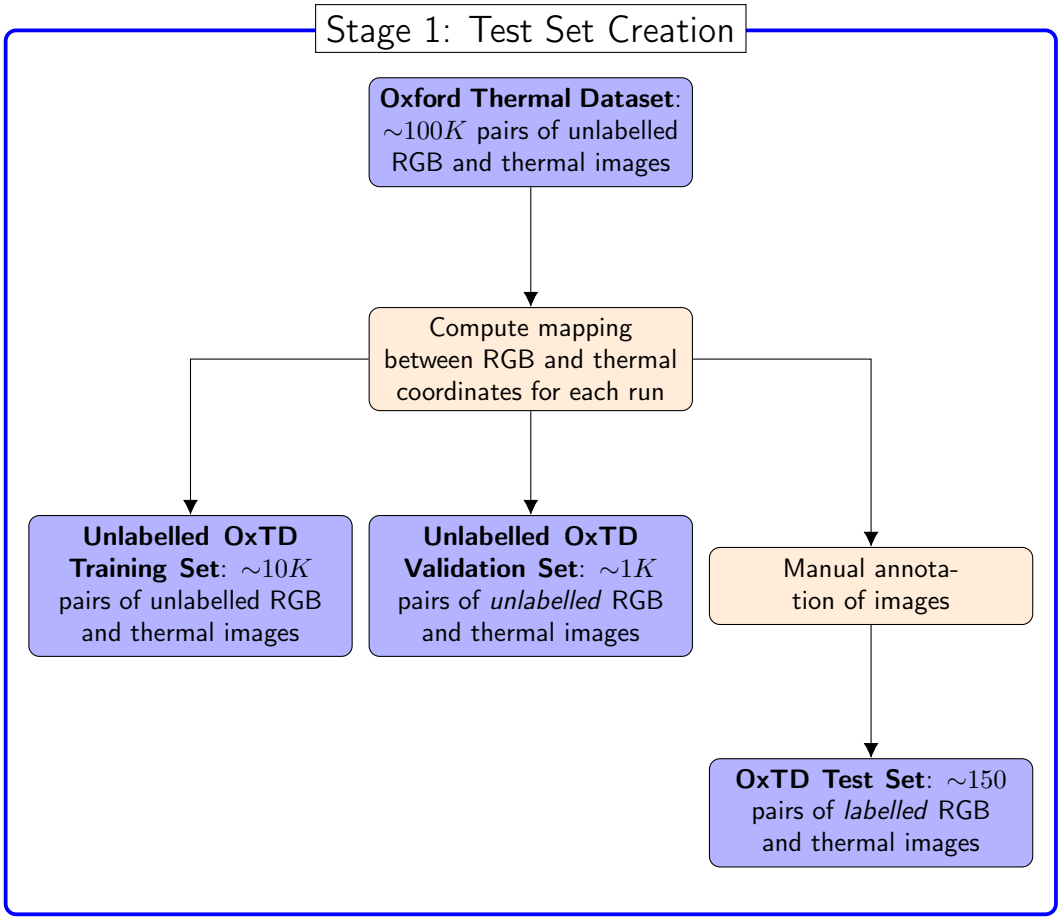


Figure 15: A flowchart illustrating Stage 1 of the training pipeline.

4.5.1 Network Architecture: Faster R-CNN

In order to perform fine tuning on RGB images, we first had to choose an object detection network to use. This would also be the network used for modality tuning on thermal images. For this project, we opt to use the Faster R-CNN meta-architecture, as for a proof of concept we prioritise accuracy over speed.[22]

The TensorFlow Object Detection API comes with what is known as a *model zoo*: a collection of detection models pre-trained on a variety of datasets. From this model zoo, as our 'base' network, we chose the Faster R-CNN with a Resnet-101 feature extractor, trained on the COCO dataset[23]. With an mAP of 0.32 on COCO test data and a runtime (per 600×600 image) of 106ms, this network yields high performance without a prohibitively slow speed.

4.5.2 RGB Datasets

Two readily-available datasets – the Indoor Scene Recognition (ISR) dataset [Quattoni and Torralba, 2009] and the Open Images Dataset (OID) [Krasin et al., 2017] – contain

[22]While Faster R-CNN tends to lead to slower models, these are often more accurate than one stage (or 'one shot') meta-architectures such as SSDs [Huang et al., 2016].

[23]Common Objects in Context (COCO) Dataset: $328K$ labelled images with annotations for 91 different classes of object, all of which 'would be easily recognisable by a 4 year old' [Lin et al., 2014]

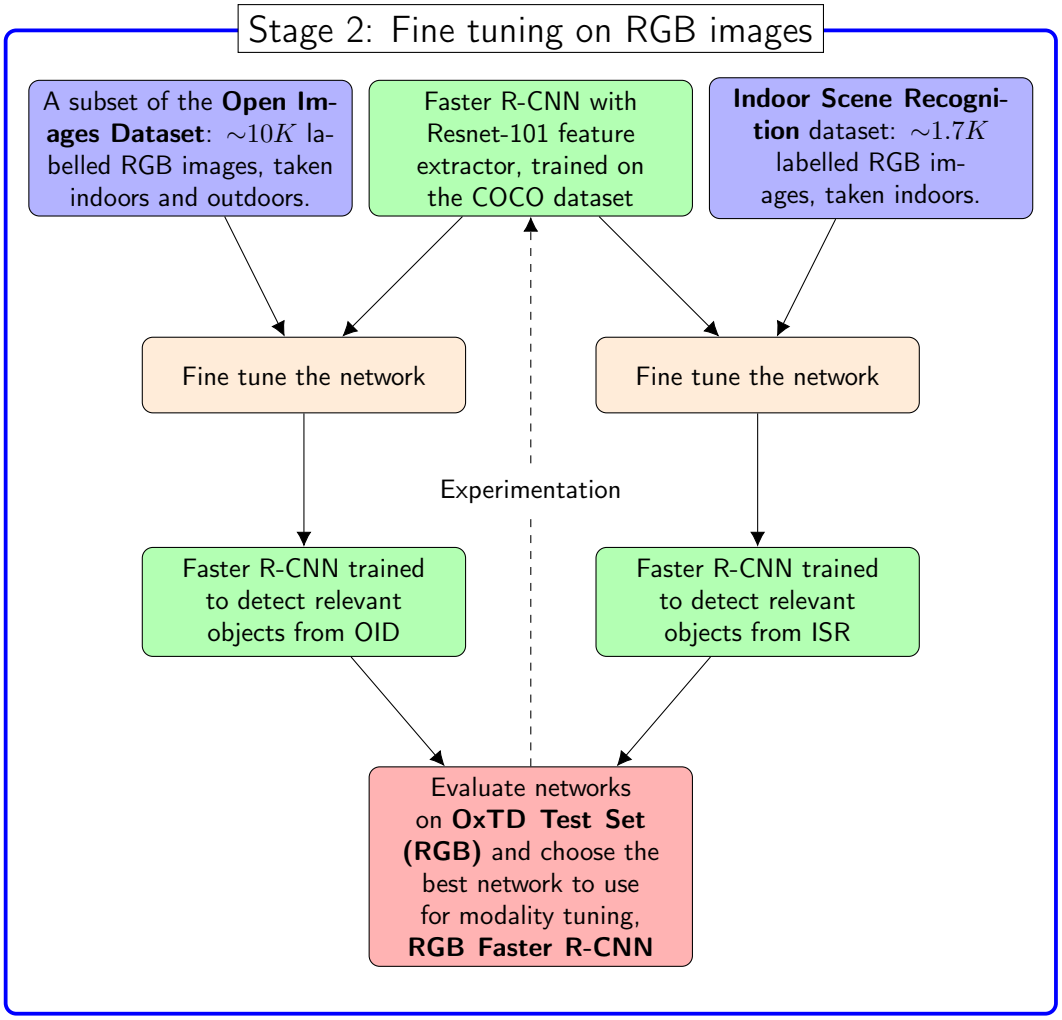


Figure 16: A flowchart illustrating Stage 2 of the training pipeline.

annotated examples of windows, chairs, doors and tables.[24] After some downsampling (details of which can be found in Appendix E.2), the former yielded examples of 1639 windows, 3170 chairs, 849 doors and 1574 tables, and the latter examples of 5458 windows, 7583 chairs, 4353 doors and 7043 tables.

In addition, we created a combined dataset of OID and ISR images (2612 windows, 3662 chairs, 1634 doors, 2354 tables), with a roughly equal split between OID and ISR. This was also used to fine tune the network.

Each of these datasets were split into class-balanced training and validation sets; the test set used was the OxTD test set.

4.5.3 Hyperparameters and Data Augmentation

When training the network, we used stochastic gradient descent (see Appendix A.2.2) with a momentum optimiser, batch size of 1 and learning rate of 0.0003. As part of input pre-processing, all images were resized to have a minimum dimension of 600 pixels, and

[24] Although standard benchmarking datasets, such as COCO, ILSVRC (ImageNet Large Scale Visual Recognition Competition dataset [Russakovsky et al., 2015]) and VOC (PASCAL Visual Object Classes dataset [Everingham et al., 2012]) all contain labelled items of furniture, neither windows nor doors feature as a category in any of these datasets.

a maximum dimension of 1024 pixels.

Data augmentation[25] was also performed to increase generalisability of the network and prevent overfitting: in all experiments, images (and their respective labels) were flipped horizontally with probability 0.5. In addition, to account for the small size of the 464×348 images taken by the thermal camera, on two experiments we randomly cropped images with probability 0.9 to an aspect ratio of 1.33 (the same as that of the thermal images) and an area ratio (cropped image to original image) of between 0.5 to 1.[26]

4.5.4 Fine Tuning and Evaluation

As described in the flow chart, we ran a series of experiments fine tuning the COCO-trained Faster R-CNN on both datasets, tuning the hyperparameters in order to achieve the best possible performance (highest mAP on the OxTD Test Set). To prevent overfitting, we used a validation-based early stopping method. At intervals of ∼5,000 iterations, we evaluated the network's performance on the validation set and saved a checkpoint of the network's weights. After training, we took as our final network the checkpoint with the highest mAP on the validation set – effectively halting training at the point at which the mAP on the validation set decreases.

For a baseline comparison, we also evaluated the performance of the TensorFlow model zoo's best performing OID-trained model (a Faster R-CNN with Inception-ResNet feature extractor [Szegedy et al., 2017]) on the OxTD Test Set in run RGB1.[27]

4.6 Stage 3: Inference and Labelling on Training and Validation Sets

A flowchart for this stage is provided in Figure 17.

From Stage 2, we were left with checkpoints containing the trained networks for each of the three non-baseline runs (RGB2–4). Our aim is now to provide labels for the RGB images in the unlabelled OxTD training and validation sets, so that we could in turn map these bounding boxes to the thermal images and create labelled thermal training and validation sets for modality tuning in Stage 4.

To do this, we used each of the three networks to perform inference on the unlabelled RGB images, creating three sets of bounding box annotations[28].

[25] Data augmentation refers to techniques that add value to base data by adding information derived from internal and external sources.

[26] Although most hyperparameters defining the network architecture itself have been retained from the original Faster R-CNN, some are significantly dependent on the dataset in question – specifically, the anchor scales and aspect ratios (see Section 3.3.4). A discussion on the choice of these hyperparameters can be found in Appendix E.3.

[27] When using this model, we removed all class weights from the last layer apart from those for the categories of windows, doors, chairs and tables. Due to time constraints and the speed of this network, we were unable to use it as part of the training pipeline proper.

[28] Given the fact that the initial layers of convolutional networks tend to detect generic features (e.g. edges, blobs, colour gradients) within images, the hope is that the top layers of the RGB-trained network will learn to detect the equivalent features in thermal images. Thus for a given thermal image the modality-tuned network should return bounding boxes that are as similar as possible to those produced when the RGB-trained network is run on the corresponding RGB image. (Note that this is unrelated to the actual accuracy of these bounding boxes – i.e. if the RGB-trained network misclassifies a certain bookshelf as a table, we would expect the modality-tuned network to do the same.) So for *each network* on which we perform modality tuning, we need to have available that network's annotations on RGB images, in order to evaluate the modality-tuned network's performance in this regard.

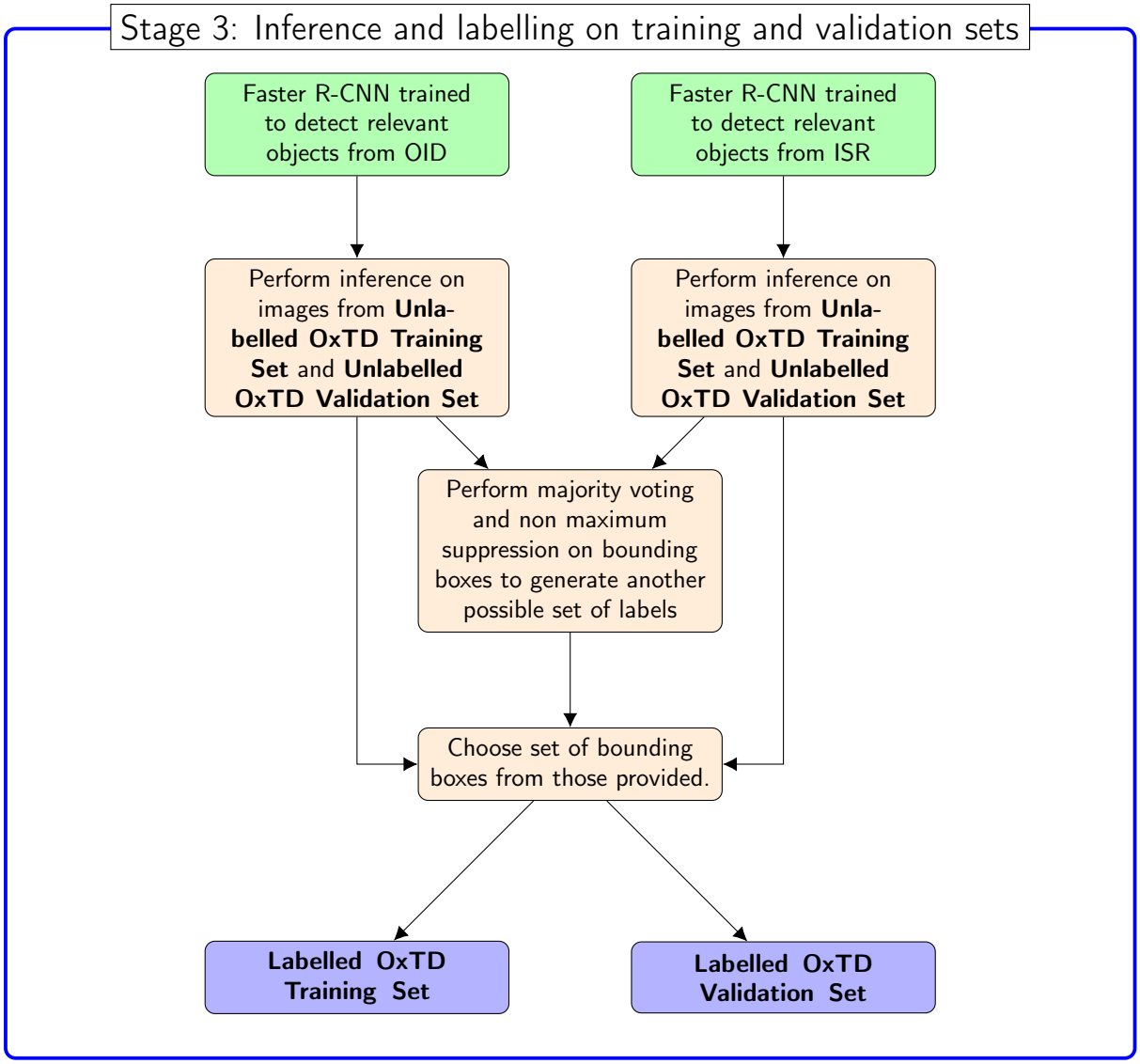


Figure 17: A flowchart illustrating Stage 3 of the training pipeline.

4.6.1 Naïve Labelling

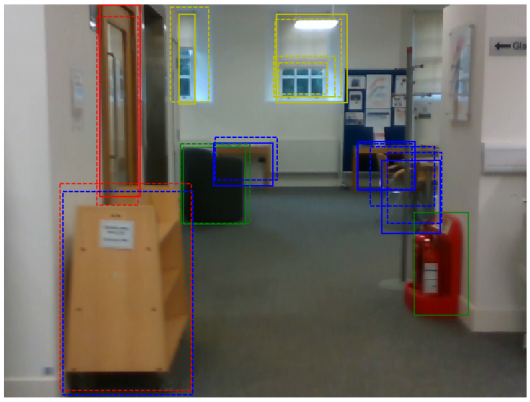
The RGB network architectures we have used output 300 object proposals per image, each with a corresponding confidence (probability that there is actually an object of the specified class within the bounding box). To create a set of annotations from runs 2, 3 and 4, we chose a confidence threshold of 0.75, and discarded all bounding boxes with a confidence less than this value[29].

[29]We somewhat arbitrarily chose this confidence threshold of 0.75. Analysis of the distribution of confidences for all bounding box predictions showed it was bimodal, with peaks at 0 to 0.1 and 0.9 to 1.0 – in other words, a significant number of proposals with low confidence / no object, and a significant number with very high confidence / likely to be an object. In general, empirical testing showed that the accuracy of predictions is (weakly) correlated with confidence, so setting a high confidence threshold would lead to more accurate training and test set labels.

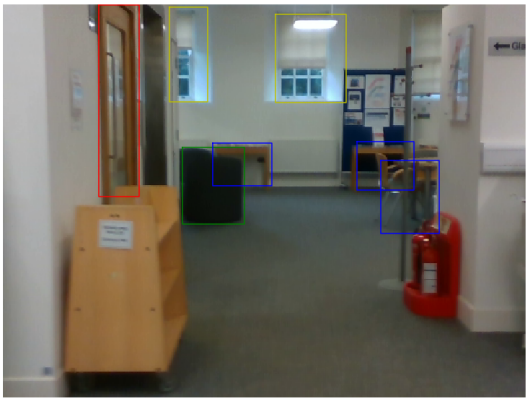
4.6.2 Majority Voting and Non-Maximum Suppression

In addition, we devised another method of labelling, designed to further increase the precision of labels (at the possible expense of recall).

Throughout this experiment, it has been challenging to deal with the noisy and unclean RGB datasets (OID and ISR), which have led to lower than expected performance on labelling test data.[30] An alternative labelling method[31] is to take the two sets of labels from the OID-trained and ISR-trained networks, and to preserve only the labels on which the two networks agree (i.e. pairs of labels, one from the OID-trained and one from the ISR-trained network, with an IoU over a given threshold). This can be regarded as a very rudimentary form of *ensemble learning* [Dzeroski and Zenko, 2004].



(a) Raw bounding box predictions (with confidence ≥ 0.1).



(b) Bounding box predictions after majority voting. Notice that duplicate predictions for the same object (e.g. the right hand window), and erroneous predictions made by only one of the networks (e.g. the trolley and fire extinguisher) have been pruned.

Figure 18: An illustration of majority voting / NMS bounding box pruning. Doors are annotated in red, chairs in green, tables in blue and windows in yellow. Bounding boxes with solid lines correspond to annotations from the ISR-trained network; those with dashed lines correspond to annotations from the OID-trained network.

[30] An observation was that, when run on the same image, both networks would tend to correctly label the actual objects in the image (with confidence > 0.1), but due to peculiarities in each dataset, each network might propose a number of other false positive labels unique to that network. (One example was the tendency of the OID-trained network to label certain bookshelves as tables; this behaviour was not seen in the ISR-trained network.) Although combining the OID and ISR datasets (as described in Section 4.5.2) managed to somewhat mitigate this issue, it did not resolve it completely, as there was still a proliferation of false positive labels with high confidences when run on the test set.

[31] The method used to do this is outlined below:

1. Prune all bounding boxes from both datasets with confidence under a given threshold (0.2).
2. Compare every OID-trained bounding box with every ISR-trained bounding box. If they are labelling objects of the same class, compute their IoU. If this is above some threshold (0.2), mark both bounding boxes for preservation.
3. For each class, run non-maximum suppression on the bounding boxes from each class with an IoU threshold of 0.2.

An example illustrating this procedure can be found in Figure 18.

4.7 Stage 4: Modality Tuning on Thermal Images

A flowchart for this stage is provided in Figure 19.

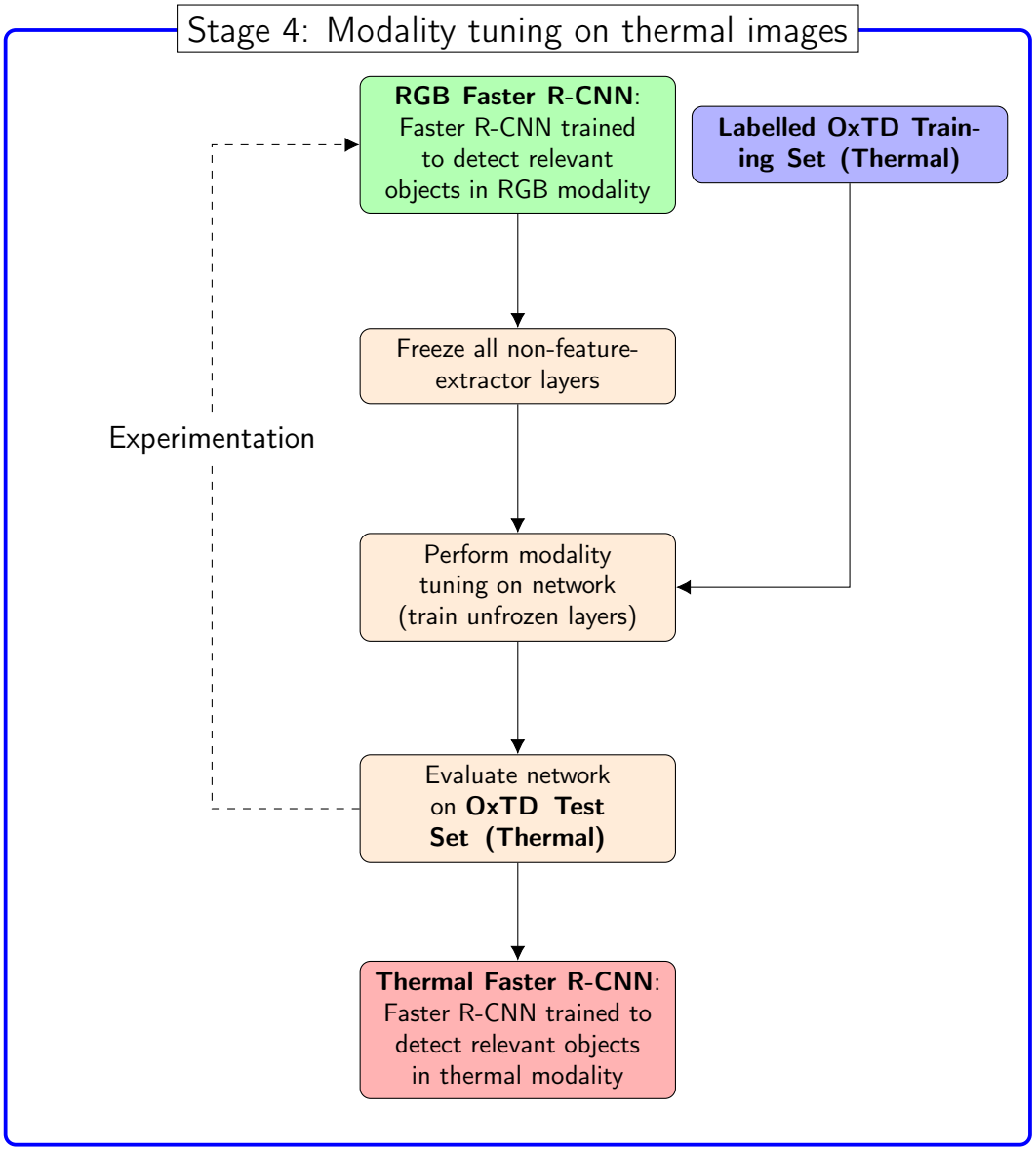


Figure 19: A flowchart illustrating Stage 4 of the training pipeline.

At this point, we are left with three trained networks, along with corresponding labels for the OxTD training and validation sets.

4.7.1 Freezing Layers

Recall that the Faster R-CNN architecture consists of all but the final layers of a feature extractor CNN, whose feature map output is then passed to region proposal and detection networks (see Figure 11). Now, we intend this feature map to be a modality-agnostic[32] high-level representation of features within the image. To achieve this, we froze all weights

[32] i.e. an RGB image and its corresponding image, when fed through the feature extractor, should return similar feature maps

of the region proposal and Fast R-CNN networks, and only fine-tuned the weights of the feature extractor CNN.

In addition, to judge the effectiveness of this method of *modality tuning* compared to simply fine tuning all layers of the network, we also fine tuned an RGB-trained network on the OxTD thermal training set without freezing any layers.

4.7.2 Modality Tuning and Evaluation

To modality tune the RGB-trained Faster R-CNNs on thermal images, we first pre-proccessed these thermal images so they can be used as inputs for the network (see Appendix E.2.1 for procedural details).

We then ran a series of experiments, fine tuning each of the three networks from Stage 2 on their corresponding OxTD training set, with all layers apart from the feature extractor frozen. In addition, we fine tuned two networks with no layers frozen: one network as a comparison between fine-tuning and modality-tuning, and another network on the OxTD training set with majority voting labels[33].

We used the same early stopping method as described in Section 4.5.4. For each of the three RGB-trained networks, we kept hyperparameters that influence the structure of the network (such as anchor size) constant, as changing these would prevent us from using the weights learned from the RGB dataset.[34] The only data augmentation method used here was a random horizontal flip (with probability 0.5).

[33] In this case, it does not make sense to freeze any layers, as the bounding boxes being used for training are not the output of any one network.

[34] We experimented with decreasing the learning rate from that used on the RGB networks, as some studies [Singh and Garzon, 2016] have reported that this yields an increase in performance when fine tuning, but this did not appear to yield any increase in test or validation mAP.

5 Experimental Results, Observations and Analysis

5.1 Results for Stage 2

Results for the experiments in Stage 2 are summarised in Table 1. On average, the networks took around 20,000 steps to converge, with validation and test mAPs plateauing or decreasing beyond this.

Run RGB1 was the baseline test described in Section 4.5.4. This achieved a surprisingly low mAP on the OxTD test set (mAP@.50 0.209) compared to the reported value of 0.54 on the OID test set [Huang et al., 2019]. This is most likely because the network was trained to detect 600 classes of objects rather than just 4, and the mAP is an average of the AP for each class, so the per-class AP for e.g. windows, tables, chairs and doors could be significantly lower than this if the AP for other objects is much higher.

Run	Dataset	Feature Extractor	Performance		
			Validation mAP@.50:.05	OxTD Test mAP@.50:.05	OxTD Test mAP@.50
RGB1	OID	Inception ResNet	n/a	0.109	0.209
RGB2	ISR	ResNet-101 (COCO)	0.298	0.139	0.250
RGB3	OID	ResNet-101 (COCO)	0.192	0.172	0.335
RGB4	Combined	ResNet-101 (COCO)	0.175	0.195	0.346

Table 1: RGB network performance (mAP) on validation and OxTD test sets. Run RGB1 was a baseline test using an Inception ResNet Faster R-CNN trained on the OID; run RGB2 used random horizontal flipping; runs RGB3 and RGB4 used an expanded range of anchor scales, random horizontal flipping and random cropping.

Compared to this OID baseline, all fine-tuned networks achieved better results, with run RGB4, trained on the combined dataset, yielding the highest test mAP of 0.195. What is interesting to note here is that, for successive runs, as the OxTD Test mAP increases, the validation mAP decreases. In particular, there is a large gap between the test and validation mAPs of runs RGB1 and RGB2, and those of runs RGB3 and RGB4 (the latter using a wider range of data augmentation methods). This indicates that, in runs RGB1 and RGB2 the network was overfitting to the environment of the images in the dataset it was trained on – in runs RGB3 and RGB4 the use of data augmentation means that the network is better able to generalise and performs better on test data of a different environment. In particular, for run RGB4, we note that it is unusual for the test mAP to be higher than the validation mAP – this is likely as combining the ISR (indoor) and OID (outdoor) datasets further reduced the network's tendency to overfit to the environment of the images it was trained on.

5.2 Results for Stage 4

Results for the experiments in Stage 4 are summarised in Table 2.

5.2.1 Effectiveness of Modality Tuning

All modality tuned networks (IR2, IR3 and IR4) performed substantially better than the baseline RGB-trained network IR1, which obtained an mAP@.50:.05:.95 of 0.034 on

Run	RGB Network (Dataset)	Method	Performance		
			Validation mAP@.50:.95	OxTD Test mAP@.50:.95	OxTD Test mAP@.50
IR1	RGB3 (OID)	No training (baseline)	0.038	0.034	0.068
IR2	RGB2 (ISR)	Modality tuning	0.169	0.132	0.268
IR3	RGB3 (OID)	Modality tuning	0.301	0.138	0.282
IR4	RGB4 (Combined)	Modality tuning	0.258	0.169	0.327
IR5	RGB3 (OID)	Fine tuning	0.301	0.136	0.266
IR6	RGB4 (Majority voting)	Fine tuning	0.218	0.152	0.293

Table 2: Thermal network performance (mAP) on validation and OxTD test sets. Run IR1 was a baseline test, evaluating an RGB-trained network (RGB3) on thermal data without any specific training on the thermal modality. The dataset indicated in brackets refers to the dataset on which the RGB network was trained, except in the case of IR6. In this case, we fine tuned the RGB4 (Combined) network on labels generated by majority voting and NMS.

OxTD thermal test data. The highest performing network, IR4, achieved a 5-fold increase in mAP (0.169) from this baseline.

	Experiment			
Metric	Baseline (IR1)	IR2	IR3	IR4
Validation mAP	0.038	0.169	0.301	0.258
RGB Test mAP	0.172	0.139	0.172	0.195
IR Test mAP	0.034	0.132	0.138	0.169
Δ (RGB$-$IR)	0.138	0.007	0.034	0.026

Table 3: A comparison of modality-tuned thermal (IR) network performance with RGB-trained network performance on the OxTD test set. All mAPs given here are mAP@.50:.05:.95. Validation mAP refers to the mAP obtained on the OxTD validation set, with experiment-specific annotations. For each IR run, RGB Test mAP is computed from the corresponding RGB-trained network.

Table 3 compares the performance of the modality tuned networks on OxTD thermal test data to that of their corresponding RGB-trained networks on OxTD RGB test data. In general, the validation mAP is significantly higher than either the RGB or the IR test mAPs; this indicates that the predictions of each IR run are similar to the predictions of the corresponding RGB-trained network.[35]

[35]In this regard, IR2 seems anomalous, with a low validation mAP of 0.169 – this is likely due to the

In addition, we note that the absolute difference between the IR and RGB test mAPs is relatively low – averaged across the three non-baseline IR runs, the IR and RGB test mAPs differ by 13.2%. This indicates that the noise / performance loss introduced by e.g. automating the dataset labelling process and mapping RGB to thermal coordinates only causes a slight decrease in mAP.

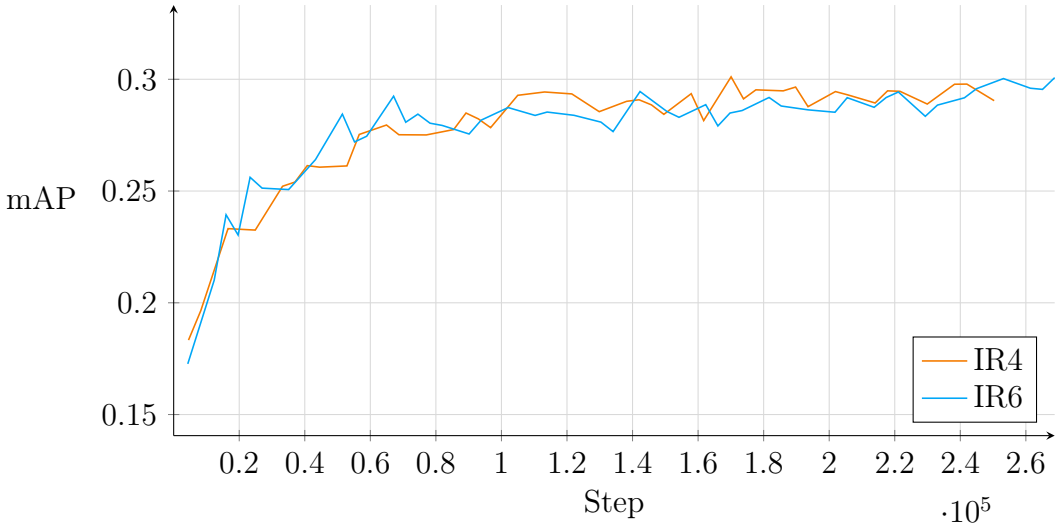


Figure 20: A graph illustrating the validation mAP@.50:.05:.95 per step during the training of the IR4 (modality tuned) and IR6 (fine tuned) networks. A step refers to one gradient update of the network – i.e. one training example being processed (as we are using a minibatch size of 1).

5.2.2 Modality Tuning or Fine Tuning?

In order to assess the effectiveness of fine tuning only the feature extractor ('modality tuning') as opposed to fine tuning all layers, we ran two experiments training the same network (RGB3) on the same training dataset, one using modality tuning and the other using fine tuning. Surprisingly, there is very little difference in results: the modality tuned and fine tuned networks have identical validation mAPs (to 3 d.p.) of 0.301, and their test mAPs (0.138 for modality tuning; 0.136 for fine tuning) agree to within 1.4%. In addition, the graph presented in Figure 20 suggests that both the modality tuned and fine tuned networks converge at roughly the same rate. This provides evidence against our original hypothesis that the method of modality tuning would provide significant performance improvements over just fine tuning the entire network – rather, as Chu et al. [2016] observed with convolutional neural network classifiers, fine tuning all layers of the network proves just as effective as selectively fine tuning certain layers.

5.2.3 Combined Datasets and Majority Voting

The two highest performing networks on the OxTD test dataset were IR4 (mAP 0.169) and IR6 (mAP 0.152); in contrast to IR1, IR2, IR3 and IR5, these both used multiple datasets in some way for training.

It is clear that amalgamation of the ISR and OID datasets pays dividends in terms of network performance, as demonstrated by IR4. In terms of raw mAP, the network

fact that the corresponding RGB-trained network (itself with a low mAP of 0.139) labelled the OxTD training dataset badly.

trained on the majority voting dataset underperforms compared to that trained on the combined dataset; this is in line with the findings of Dzeroski and Zenko [2004], who showed that "ensemble classifiers ... perform (at best) comparably to selecting the best classifier from the ensemble".

Analysis of a sample of annotated OxTD images suggests that IR4 has a higher precision than IR6 at low values of recall (although IR4 annotates fewer ground truth objects than IR6, it produces fewer false positives);[36] to confirm this, we would need to produce precision-recall curves for these networks[37].

5.2.4 Category-wise Performance

A brief discussion of the category-wise performance of each IR run can be found in Appendix F.1.

[36] Even if this holds, the comparative utility of both networks (in other words, whether or not one should prioritise precision over recall) is a question for discussion: on the one hand, if a false positive door or window appears in the dataset, a firefighter would waste significant time, and perhaps endanger their life, attempting to find a non-existent exit; on the other hand, if an obstacle such as a table is not identified correctly (a false negative), a firefighter might injure themselves if they walk into it.

[37] Unfortunately, this has not been possible, due to time constraints.

6 Conclusions and Further Work

Summary There are three main insights we can draw from this project.

First, we demonstrate that it is possible to train an object detection network on RGB images, run this network on a larger dataset of RGB images to obtain a pseudo-ground truth, and then use this pseudo-ground truth to perform transfer learning on the network such that it can detect objects in images of another modality (thermal images). To do this, we implement a proof-of-concept pipeline for training a Faster R-CNN object detection network on thermal images, and with some hyperparameter tuning achieved an mAP@.50:.05:.95 of 0.169 on the OxTD test data.

We also find that performing *modality tuning* [Castrejón et al., 2016] on an RGB-trained network (i.e. only training feature extractor layers) yields no significant advantages, either in performance or training time, compared to fine tuning all layers of the network.

Finally, we perform various methods of *data augmentation* when training the RGB and thermal networks; we also investigate training object detection networks on various combinations of datasets, alongside an ensemble method of annotating the OxTD training set. We conclude that data augmentation, along with combining different datasets, can yield a substantial performance increase (i.e. increase in OxTD test set mAP), and that ensemble methods of creating a pseudo-ground truth dataset yield a performance comparable to that when using the combined dataset.

Overall, these results are promising, and indicate that there is scope for further research in this area. While not yet robust, reliable or fast enough to operate in the field on an embedded system, it is a testament to the power of modern machine learning techniques that such a pipeline could be implemented in a short span of time.

Further Work Possible improvements[38] and further directions for research are outlined in Appendix G.

Acknowledgements This work would not have been possible without the help and support of Professor Trigoni and Dr Gusmão of the Cyber Physical Systems Research Group in the University of Oxford. I would especially like to thank Professor Trigoni for proposing this fascinating project, and for her tireless efforts in procuring the dataset and computing resources. At every stage of this project, I have received invaluable advice from Professor Trigoni, Dr Gusmão and Mr Andrew, for which I am deeply grateful.

[38] Owing to ethical issues regarding the dataset, we were only able to gain access to the OxTD at the start of April. Because of the time constraint, our main goal was to implement an "existence proof" of the training pipeline; time-consuming optimisation procedures such as hyperparameter tuning via grid search were unfortunately not feasible in the time available for the project.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org. pages 20

Tadas Baltrusaitis, Chaitanya Ahuja, and Louis-Philippe Morency. Multimodal machine learning: A survey and taxonomy. *CoRR*, abs/1705.09406, 2017. URL http://arxiv.org/abs/1705.09406. pages 19

Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. ISBN 9780387310732. URL http://www.worldcat.org/oclc/71008143. pages 4, 8, 15, 40

Lluís Castrejón, Yusuf Aytar, Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Learning aligned cross-modal representations from weakly aligned data. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2940–2949, 2016. URL https://doi.org/10.1109/CVPR.2016.321. pages 5, 19, 32

Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, 2015. URL http://jmlr.org/proceedings/papers/v38/choromanska15.html. pages 8

Brian Chu, Vashisht Madhavan, Oscar Beijbom, Judy Hoffman, and Trevor Darrell. Best practices for fine-tuning visual classifiers to new domains. In *Computer Vision - ECCV 2016 Workshops - Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III*, pages 435–442, 2016. URL https://doi.org/10.1007/978-3-319-49409-8_34. pages 30, 52

Zhipeng Deng, Hao Sun, Shilin Zhou, Juanping Zhao, Lin Lei, and Huanxin Zou. Multi-scale object detection in remote sensing imagery with convolutional neural networks. *ISPRS Journal of Photogrammetry and Remote Sensing*, 05 2018. URL https://doi.org/10.1016/j.isprsjprs.2018.04.003. pages 17

Saso Dzeroski and Bernard Zenko. Is combining classifiers with stacking better than selecting the best one? *Machine Learning*, 54(3):255–273, 2004. doi: 10.1023/B:MACH.0000015881.36452.6e. URL https://doi.org/10.1023/B:MACH.0000015881.36452.6e. pages 25, 31

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results, 2012. pages 22, 50

Mark Everingham and John Winn. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit, 2012. URL http://host.robots.ox.ac.uk/pascal/VOC/voc2012/htmldoc/devkit_doc.html. [Online; accessed April 28, 2019]. pages 50

Ross B. Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1440–1448, 2015. doi: 10.1109/ICCV.2015.169. URL https://doi.org/10.1109/ICCV.2015.169. pages 16, 17

Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 580–587, 2014. URL https://doi.org/10.1109/CVPR.2014.81. pages 15

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL http://proceedings.mlr.press/v15/glorot11a.html. pages 45

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. pages 4, 6, 9, 10

HM Government Defence Fire Risk Management Organisation. Breathing apparatus (ba) procedures, 2014. URL https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/444541/20140425-DFRMO_BA_Search_Procedures-O.pdf. [Online; accessed May 17, 2019]. pages 4, 62

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. doi: 10.1016/0893-6080(91)90009-T. URL https://doi.org/10.1016/0893-6080(91)90009-T. pages 6

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861. pages 62

Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016. URL http://arxiv.org/abs/1611.10012. pages 13, 14, 21, 50, 51

Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Tensorflow detection model zoo, 2019. URL https://github.com/tensorflow/models/blob/8d97814ef3f44622cf8ec3ae5f35d8bd0d2f4e6a/research/object_detection/g3doc/detection_model_zoo.md. [Online; accessed May 14, 2019]. pages 13, 28, 50

Markus Jangblad. Object Detection in Infrared Images using Deep Convolutional Neural Networks. 2018. URL https://uu.diva-portal.org/smash/get/diva2:1228617/FULLTEXT01.pdf. Master's thesis, Uppsala Universitet. pages 55

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pages 2146–2153, 2009. doi: 10.1109/ICCV.2009.5459469. URL https://doi.org/10.1109/ICCV.2009.5459469. pages 45

Andrej Karpathy. CS231n Convolutional Neural Networks for Visual Recognition, 2019. URL http://cs231n.github.io/convolutional-networks/. pages 11, 46

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, pages 528–536, 2017. URL http://proceedings.mlr.press/v54/klein17a.html. pages 61

Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Shahab Kamali, Matteo Malloci, Jordi Pont-Tuset, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. 2017. Dataset available from https://storage.googleapis.com/openimages/web/index.html. pages 21

Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541. URL https://doi.org/10.1162/neco.1989.1.4.541. pages 8

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, page 319, 1999. URL https://doi.org/10.1007/3-540-46805-6_19. pages 40, 45

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, pages 740–755, 2014. doi: 10.1007/978-3-319-10602-1_48. URL https://doi.org/10.1007/978-3-319-10602-1_48. pages 13, 21, 50

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context Detection Leaderboard, 2018. URL http://cocodataset.org/#detection-leaderboard. [Online; accessed April 27, 2019]. pages 13, 50

Yuan-Pin Lin and Tzyy-Ping Jung. Improving eeg-based emotion classification using conditional transfer learning. *Front Hum Neurosci*, 11:334–334, Jun 2017. ISSN

1662-5161. doi: 10.3389/fnhum.2017.00334. URL https://www.ncbi.nlm.nih.gov/pubmed/28701938. pages 52

Charles X. Ling and Victor S. Sheng. *Class Imbalance Problem*, pages 171–171. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_110. URL https://doi.org/10.1007/978-0-387-30164-8_110. pages 55

Zack Chase Lipton. Gradient descent and stochastic gradient descent from scratch, 2017. URL https://gluon.mxnet.io/chapter06_optimization/gd-sgd-scratch.html. [Online; accessed May 19, 2019]. pages 44

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I*, pages 21–37, 2016. doi: 10.1007/978-3-319-46448-0_2. URL https://doi.org/10.1007/978-3-319-46448-0_2. pages 62

Manolis Loukadakis, José Cano, and Michael F. P. O'Boyle. Accelerating deep neural networks on low power heterogeneous architectures. 2018. pages 12

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6232–6240, 2017. URL http://papers.nips.cc/paper/7203-the-expressive-power-of-neural-networks-a-view-from-the-width. pages 6

Pamela McCorduck. *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. AK Peters Ltd, 2004. ISBN 1568812051. pages 4

Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943. pages 6

Waldemar Minkina and Sebastian Dudzik. *Measurements in Infrared Thermography*, chapter 2, pages 15–40. John Wiley & Sons, Ltd, 2009. ISBN 9780470682234. doi: 10.1002/9780470682234.ch2. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470682234.ch2. pages 18

Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997. ISBN 978-0-07-042807-2. URL http://www.worldcat.org/oclc/61321007. pages 40

Hans Moravec. *Mind Children: The Future of Robot and Human Intelligence*. Harvard University Press, Cambridge, MA, USA, 1988. ISBN 0-674-57616-0. pages 4

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010. URL https://icml.cc/Conferences/2010/papers/432.pdf. pages 45

Andrew Ng. CS229 Lecture notes. *mimic.stanford.edu*, 2003. URL http://mimic.stanford.edu/public/tristan/CS229/cs229-notes1.pdf. pages 43

Lorien Y. Pratt. Discriminability-based transfer between neural networks. In *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 204–211, 1992. URL http://papers.nips.cc/paper/641-discriminability-based-transfer-between-neural-networks. pages 5

Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 413–420, 2009. doi: 10.1109/CVPRW.2009.5206537. URL https://doi.org/10.1109/CVPRW.2009.5206537. pages 21

Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 779–788, 2016. doi: 10.1109/CVPR.2016.91. URL https://doi.org/10.1109/CVPR.2016.91. pages 62

Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149, 2017. URL https://doi.org/10.1109/TPAMI.2016.2577031. pages 5, 17, 55

Seyed Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian D. Reid, and Silvio Savarese. Generalized intersection over union: A metric and A loss for bounding box regression. *CoRR*, abs/1902.09630, 2019. URL http://arxiv.org/abs/1902.09630. pages 48

Giorgio Roffo. Ranking to learn and learning to rank: On the role of ranking in pattern recognition applications. *CoRR*, abs/1706.05933, 2017. URL http://arxiv.org/abs/1706.05933. pages 50

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958. pages 6

Soumya Roy, Asim Unmesh, and Vinay P. Namboodiri. Deep active learning for object detection. In *British Machine Vision Conference 2018, BMVC 2018, Northumbria University, Newcastle, UK, September 3-6, 2018*, page 91, 2018. URL http://bmvc2018.org/contents/papers/0287.pdf. pages 61

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6. URL http://dl.acm.org/citation.cfm?id=65669.104451. pages 8

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. URL https://doi.org/10.1007/s11263-015-0816-y. pages 22

Dominik Scherer, Andreas C. Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Networks - ICANN 2010 - 20th International Conference, Thessaloniki, Greece, September 15-18, 2010, Proceedings, Part III*, pages 92–101, 2010. URL https://doi.org/10.1007/978-3-642-15825-4_10. pages 10

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556. pages 12

Diveesh Singh and Pedro Garzon. Using convolutional neural networks and transfer learning to perform yelp restaurant photo classification. 2016. URL http://cs231n.stanford.edu/reports/2016/pdfs/001_Report.pdf. CS231n course project, Stanford University. pages 27

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6806. pages 10

Wanhua Su, Yan Yuan, and Mu Zhu. A relationship between the average precision and the area under the ROC curve. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval, ICTIR 2015, Northampton, Massachusetts, USA, September 27-30, 2015*, pages 349–352, 2015. URL https://doi.org/10.1145/2808194.2809481. pages 50

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 4278–4284, 2017. URL http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14806. pages 23

Jasper R. R. Uijlings, Koen E. A. van de Sande, Theo Gevers, and Arnold W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013. URL https://doi.org/10.1007/s11263-013-0620-5. pages 15, 17

Stacey Venzel. Two dogs looking surprised, 2016. URL https://www.wideopenpets.com/how-to-succeed-with-doggy-meet-and-greets/two-dogs-looking-surprised/. [Online; accessed April 27, 2019]. pages 13

Eric W. Weisstein. Directional derivative. From MathWorld—A Wolfram Web Resource, 2019. URL http://mathworld.wolfram.com/DirectionalDerivative.html. [Online; accessed April 27, 2019]. pages 43

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014. URL http://arxiv.org/abs/1411.1792. pages 52

Kyongsik Yun, Alexander Huyen, and Thomas Lu. Deep neural networks for pattern recognition. *CoRR*, abs/1809.09645, 2018. URL http://arxiv.org/abs/1809.09645. pages 63

Xiangxin Zhu, Carl Vondrick, Deva Ramanan, and Charless C. Fowlkes. Do we need more training data or better models for object detection? In *British Machine Vision Conference, BMVC 2012, Surrey, UK, September 3-7, 2012*, pages 1–11, 2012. doi: 10.5244/C.26.80. URL https://doi.org/10.5244/C.26.80. pages 53

Appendices

A Overview of Machine Learning

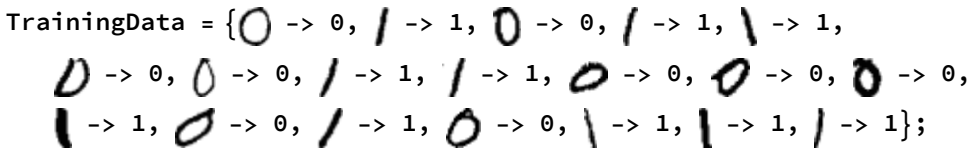
The following appendix introduces key concepts in machine learning related to computer vision. Through the lens of problems in the natural sciences, we provide a broad overview of terms and ideas in machine learning, and frame machine learning as an optimisation problem to be solved by numerical methods such as stochastic gradient descent (SGD).

A.1 Machine Learning

Machine learning is a term used to describe a variety of techniques, all of which use algorithms to make inferences from data: specifically those whose "performance at tasks in [some class of tasks] T, as measured by [some performance measure] P, improves with experience E" [Mitchell, 1997]. At its heart, it is simply the automation of the process of finding patterns in data – given a series of examples, and by learning the patterns and regularities inherent in these, an algorithm is able to improve its ability to take actions such as classifying this data into different categories [Bishop, 2007]. With machine learning, our approach shifts from that of a mathematical science (typical of most computational problems) to that of a natural science: we are making observations of and proposing models for an uncertain world, and using statistical methods to evaluate our results.

A.1.1 Example

Consider the problem of deciding whether a handwritten digit is a 0 or a 1, given the following set of 28×28 pixel images [LeCun et al., 1999]:



The goal, therefore, is to create an algorithm that, given a 28×28 pixel image not in this set, will output either 0 or 1 depending on whether the digit drawn in the image is a 0 or a 1.

Now one way to solve this problem could be to develop a heuristic based approach to classify these digits based on their shape. In practice, however, this method would lead to a plethora of rules and exceptions, and poor results.

An alternative solution, rather than coding the heuristics by hand, is to let the computer do it for you. A machine learning approach to this problem would propose an adaptive function, or *model*, and adjust the parameters of this function using the images provided such that it learns the relationship between the values of the pixels in the image and whether or not it represents a 0 or a 1.

A.1.2 Types of Machine Learning

Machine learning is usually divided into two main types: supervised and unsupervised learning. In supervised learning (as in our digit recognition example above) our goal is to estimate (or *learn*) the relationship between a dependent variable y and one or more independent variables x_i from a set of examples by tuning the parameters of an adaptive function, or *model* – a process called *training*. Each independent variable is called a *feature*: these features are individual properties of the example being observed, such as a person's height, a word or the value of a given pixel in an image. The dependent variable is called a *label* – this is the value we are predicting and can in principle be anything, although in practice this is almost always either a *categorical* variable from a finite set (represented as a vector) or a real-valued scalar. If the dependent variable is categorical, we call this a *classification* problem; if it is real-valued, we call this a *regression* problem.

In essence, we aim to learn a mapping from vectors of features $\mathbf{x}$ to labels y from a given set of N examples (associations between features and labels) $\mathbf{X} = \{(\mathbf{x}_i,y_i)\}_{i=1}^N$. This set is called the *training set* (this is sometimes represented as a matrix, where each row is a training example). This mapping should then be able to make accurate estimations of y for values of $\mathbf{x}$ not in the training set – in other words, to generalise to unseen data – in a process called *inference*.

By contrast, in unsupervised learning, we are only given a set of feature vectors $\{\mathbf{x}_i\}_{i=1}^N$, without their corresponding labels. The goal here can be varied, ranging from *clustering*, grouping similar feature vectors together, to *dimensionality reduction*, projecting the feature data from high-dimensional space to low-dimensional space either for visualisation or as input into another machine learning algorithm.

The problem of object detection, which we aim to solve, is a supervised machine learning problem (we are provided with training data consisting of a series of images along with the positions of objects in them); from hereon subsequent uses of the phrase *machine learning* will refer to *supervised* machine learning.

A.2 Linear Regression

To motivate our discussion of deep learning, we introduce a simple linear regression model. Consider the data shown in figure 21.

Suppose this data related, say, the force applied to a spring (on the x axis) to its length (on the y axis). We want to fit a model to this data – in other words, propose a function that will map values of the feature x (force applied), to labels y (length of the spring). It is known that the length of a spring increases linearly with the force applied to it, therefore it would be sensible to propose the following model for this data:

$$y(x,\mathbf{w}) = w_0 + w_1 x \tag{6}$$

Here, $\mathbf{w} = (w_0,w_1)$ are the parameters (or 'weights') of our function, so called as they define how much weight is given to each feature in the model. w_0 is sometimes referred to as the *bias* (b).

We can easily extend this model to feature vectors $\mathbf{x}$ containing D (> 1) features as follows:

$$y(\mathbf{x},\mathbf{w}) = w_0 + \sum_{i=1}^{D} w_i x_i \tag{7}$$

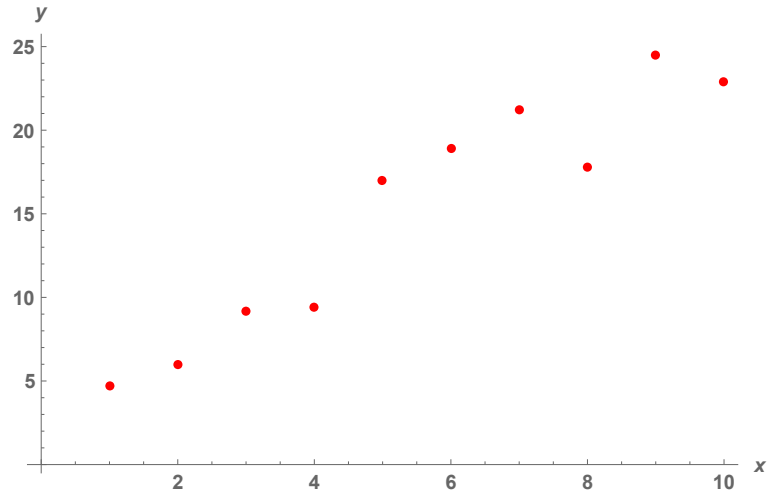


Figure 21: Linear data (generated from the function $y = 2x + 5$ with Gaussian noise added to the target values)

A.2.1 Reducing Loss

In order to fit the model to the data, we need to find the best values $\mathbf{w}^*$ for the weights defined in $\mathbf{w}$. This can be done by minimising a *loss*, or error, function $\ell(\mathbf{X}, \mathbf{y}; \mathbf{w})$, that quantifies how well the function y estimates the target values y_i for each $\mathbf{x_i} \in \mathbf{X}$ for given values of $\mathbf{w}$. By convention, the lower the loss, the better the fit of the model – if the model's prediction is perfect, the loss is zero.

One such loss function is *mean squared error* (MSE) – as its name suggests, it is computed by finding the mean of the squares of the errors (differences between the target value y_i and the predicted value $\hat{y}_i$ for each data point $\mathbf{x_i} \in \mathbf{X}$), so that we minimise

$$\ell(\mathbf{X}, \mathbf{y}; \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{N} (y(\mathbf{x_i}, \mathbf{w}) - y_i)^2 \tag{8}$$

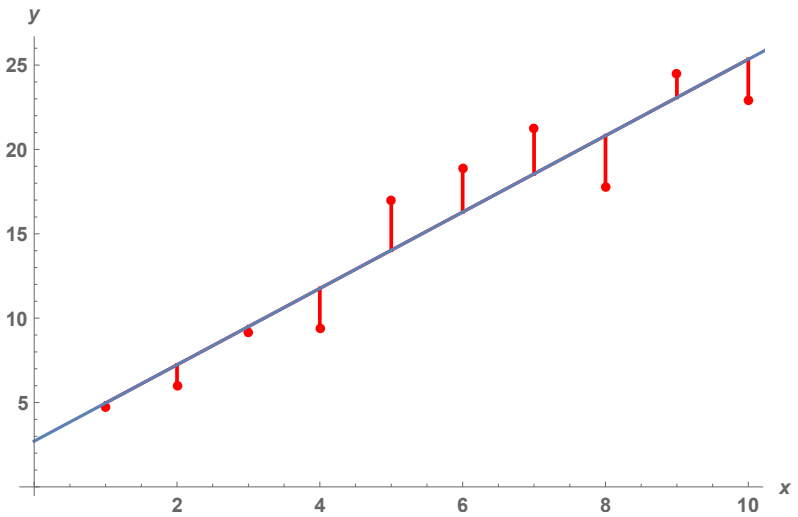


Figure 22: The error function in (8) corresponds to the mean of the sum of the squares of the displacements of each point from the function $y(\mathbf{x_i}, \mathbf{w})$ (shown by the red bars).

Now, for the example of linear regression we have here, one way to train our model

would be to find an analytical / closed-form solution for $\arg\min_{\mathbf{w}} \ell(\mathbf{X}, \mathbf{y}; \mathbf{w})$. It so happens that, for the case of linear regression, this is possible [Ng, 2003]. However, for almost all other models (especially those with many parameters and in many variables), this is either mathematically impossible or computationally infeasible. Thus we need a way of numerically approximating this minimum: one such method is stochastic gradient descent.

A.2.2 Stochastic Gradient Descent

Suppose we are trying to minimise some arbitrary real-valued function $f(\mathbf{w})$, where $\mathbf{w}$ is a D-dimensional vector. Now we can visualise this function f for all possible values of $\mathbf{w}$ as a surface, or a kind of 'valley', for which we want to find the lowest point. One algorithm to numerically approximate this lowest point is motivated by the following analogy: if a blind hillwalker were standing at some point $\mathbf{w}$ on this surface and wished to walk down the valley to a minimum point, a logical way in which they could do this would be to ensure that, for each step they take, they go down the valley as much as possible (their vertical distance from the bottom of the valley decreases as much as possible). In other words, they would test all points that are some unit vector $\mathbf{u}$ away from their current position $\mathbf{w}$, find the lowest such point, and take a step of some size η in that direction.

Now, the *directional derivative* $\nabla_{\mathbf{u}} f$ (the gradient of f at a given point $\mathbf{w}$ in the direction $\mathbf{u}$) is defined [Weisstein, 2019] as

$$\nabla_{\mathbf{u}} f(\mathbf{w}) = \nabla f(\mathbf{w}) \cdot \frac{\mathbf{u}}{|\mathbf{u}|} \tag{9}$$

where $\nabla f(\mathbf{w})$ is the gradient of f at $\mathbf{w}$ (i.e. the vector of partial derivatives $(\frac{\partial f(\mathbf{w})}{\partial w_i})_{i=1}^{D}$). Hence, the lowest point a distance of $|\mathbf{u}|$ away from $\mathbf{w}$ will be when $\nabla_{\mathbf{u}} f$ is at its minimum.

For vectors $\mathbf{x}$ and $\mathbf{y}$, $\mathbf{x} \cdot \mathbf{y} = |\mathbf{x}||\mathbf{y}| \cos\theta$ (where θ is the angle between the two vectors). So we want to find the minimum of

$$\nabla_{\mathbf{u}} f(\mathbf{w}) = \frac{1}{|\mathbf{u}|} |\nabla f(\mathbf{w})||\mathbf{u}| \cos\theta \tag{10}$$

$$= |\nabla f(\mathbf{w})| \cos\theta \tag{11}$$

with respect to $\mathbf{u}$. Now this will be minimised when $\cos\theta = -1$ – i.e. when the angle between the vectors is 180°, and when $\mathbf{u}$ is in the opposite direction to $\nabla f(\mathbf{w})$.

Thus, in order to iteratively reduce the value of $f(\mathbf{w})$, we can perform the following gradient descent update:

$$\mathbf{w} := \mathbf{w} - \eta \nabla f(\mathbf{w}) \tag{12}$$

where η is a tunable hyperparameter known as the *learning rate*. This controls the size of each 'step': too high a learning rate and one risks overshooting the minimum; too low a learning rate and convergence may be very slow.

Substituting the loss function $\ell(\mathbf{X}, \mathbf{y}; \mathbf{w})$ in place of $f(\mathbf{w})$ we obtain the following iteration for minimising loss:

$$\mathbf{w} := \mathbf{w} - \eta \nabla \ell(\mathbf{X}, \mathbf{y}; \mathbf{w}) \tag{13}$$

However, as calculating the gradient of the loss function $\ell(\mathbf{X}, \mathbf{y}; \mathbf{w})$ over all training examples $\mathbf{x} \in \mathbf{X}$ is computationally expensive, we use a lighter weight solution: to approximate the gradient of ℓ, at each iteration we take a sample[39] of training examples (a 'mini-batch') $\mathbf{B}$, and compute the mean of the gradients of $\ell(\mathbf{B}, \mathbf{y_B}; \mathbf{w})$ over these examples as an approximation to the true gradient $\nabla\ell(\mathbf{B}, \mathbf{y_B}; \mathbf{w})$. This is known as *minibatch gradient descent*. In cases when the number of elements in each batch is equal to 1, and where this element is randomly selected, this is known as *stochastic gradient descent* [Lipton, 2017].

[39]This can either be a random sample of training examples, or it may move through the full training set systematically.

B Deep Learning

B.1 Activation Functions

It is important that the activation function be *non-linear*, as it allows for non-linear decision boundaries to be learned by the network – compositions of linear regressors will only ever produce a linear decision boundary, and will be no better than simply learning a linear regressor on the data.

A variety of different functions f can be used as activation functions, including the sigmoid function,

$$f(x) = \frac{1}{1 + e^{-x}} \tag{14}$$

the hyperbolic tangent (*tanh*) function,

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{15}$$

and the ReLU (*rectified linear unit*) function.

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{16}$$

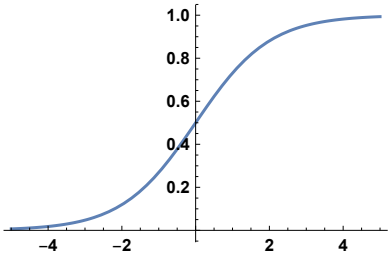


Figure 23: Sigmoid function

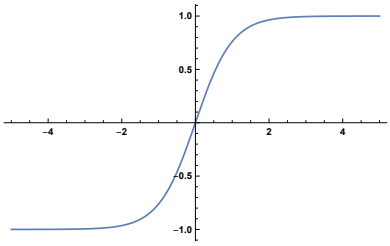


Figure 24: *tanh* function

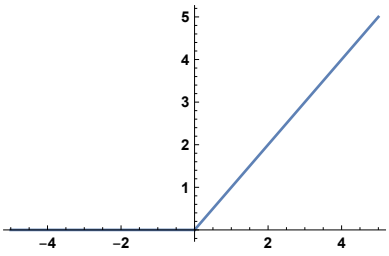


Figure 25: ReLU function

Studies by Jarrett et al. [2009], Nair and Hinton [2010], Glorot et al. [2011] have shown that, for many neural network architectures, using the ReLU activation function yields better performance than alternatives such as the sigmoid or *tanh* functions[40]; as a result it is one of the most commonly used activation functions in modern neural networks.

B.2 MNIST and Feedforward Neural Networks

The MNIST[41] dataset [LeCun et al., 1999] contains 60,000 training and 10,000 test examples of handwritten digits, represented as 28×28 images. A sample of these data is shown in Figure 26.

Dubbed "the *Drosophila* of machine learning" by Geoffrey Hinton, MNIST is one of the simplest and most widely used datasets in deep learning, allowing researchers to test and investigate machine learning algorithms under controlled conditions.

[40] For *saturating* functions such as sigmoid and *tanh*, and for large values of x, the gradient tends towards 0. This can lead to what is known as the *vanishing weight problem*: during backpropagation the partial derivatives $\frac{\partial E}{\partial w}$ of weights w are computed using the chain rule, which can result in the gradient update being extremely small and the weights not updating at all. ReLU avoids this as its gradient is either 1 if $x > 0$ or 0 otherwise.

[41] Modified National Institute of Standards and Technology

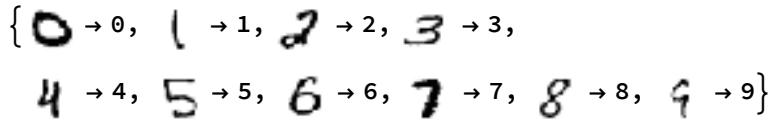


Figure 26: A sample of 10 digits from the MNIST dataset

We trained a feedforward neural network on 60,000 images from this dataset, achieving an accuracy of 97.3% on a test dataset of 10,000 images. This network consists of four layers, each with 225, 144, 64, and 10 neurons respectively, using ReLU activation and a softmax / cross entropy loss function.

We can gain some insight into how this network detects features within the digits of the dataset by looking 'under the hood'. Taking the first layer of the network, we can extract the weights from each of the 225 neurons. Now notice that each of these neurons takes as input all of the 784 pixels of the image, each of which is either 0 or 1. Thus for each neuron, each pixel has a respective weight. We can now represent the weights of each neuron as 28×28 images (where the top left pixel in the visualisation represents the weight assigned to the top left pixel in the image), as shown in Figure 27.

What is striking about this collage of neurons are two things. First, while the magnitudes of the weights of some neurons appear very low or close to 0, and their images appear close to random noise, for others one can very clearly see patterns – swirls, strokes and lines clustered in a particular region of the image – like the sorts of patterns one might find in an image of an MNIST digit. One might infer, therefore, that these neurons will fire if that particular pattern is detected in the image. Second, notice that there are a number of neurons which appear to be looking for the same (or a similar) feature, but in slightly different places.

This all goes to suggest that, for images, the full connectivity of a feedforward neural network is wasteful – and this problem only grows with larger networks. Take, for instance, a 200×200 3-channel RGB image. If each pixel is an input, this would need neurons with $200 \times 200 \times 3 = 120{,}000$ weights; and we would want multiple such neurons, further increasing the number of parameters. Not only does this lead to increased time and space complexity, but such a large number of parameters would likely yield a model prone to overfitting [Karpathy, 2019].

Convolutional neural networks are specifically designed to exploit the spatial structure of images – for instance, that comparing pixels that are close together is more relevant than comparing pixels that are far apart – making them more effective and faster to train, particularly on larger images. They do this through two new types of layer: the *convolutional* ('Conv') layer, and the *pooling* layer.

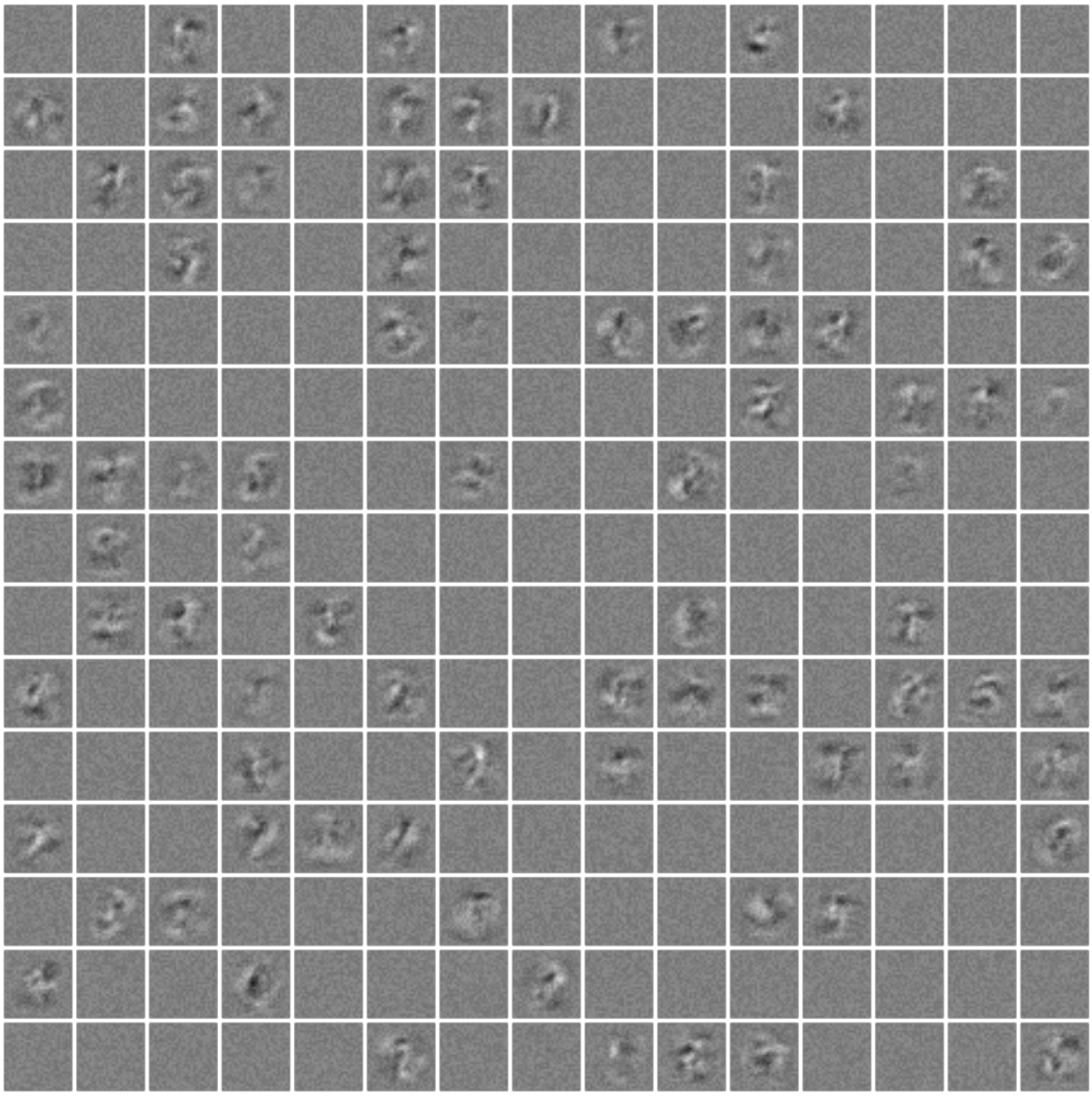


Figure 27: The weights of the 225 neurons in the first layer of the network. Each pixel represents a weight, scaled such that a completely black pixel represents a weight of 1, and a completely white pixel represents a weight of -1. Thus gray pixels (between black and white) will have weights close to 0 – the closer the pixel is to black or to white, the larger the magnitude of the weight.

C Evaluation Metrics

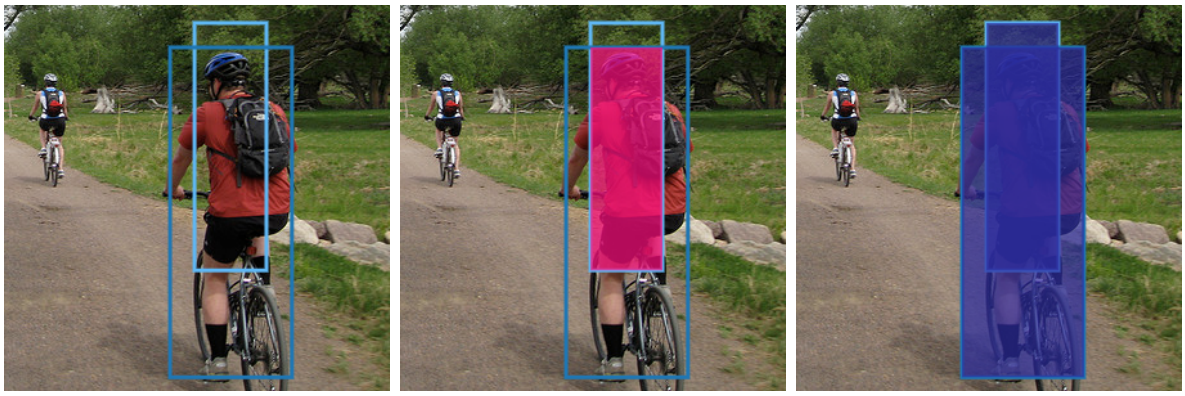
C.1 Intersection over Union

When training and testing an object detection network, we need to be able to evaluate the network's performance. In image classification tasks, this is straightforward: if the predicted label is the same as the ground truth label, the prediction is correct, otherwise it is incorrect. However, in object detection tasks, where the network must predict bounding box coordinates (a regression task) as well as classify the object within the bounding box, we have to be able to quantify how close the predicted bounding box is to the ground truth.

The most popular evaluation method for such problems is *Intersection over Union* (IoU, also known as the *Jaccard index*), and is computed as follows:

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} = \frac{|I|}{|U|} \tag{17}$$

where A and B are the ground truth and prediction bounding boxes respectively, I is the *intersection area* (Figure 28b) and U is the *union area* (Figure 28c). The value of IoU can range from 0 to 1, 0 being when the two bounding boxes do not intersect at all, and 1 being when they overlap completely (i.e. a perfect prediction).



(a) Bounding Boxes (b) Intersection area I (c) Union area U

Figure 28: An example demonstrating the *Intersection over Union* evaluation metric. Ground truth bounding box in dark blue; prediction bounding box in light blue. [Rezatofighi et al., 2019]

C.2 Precision, Recall and mAP

Recall that classification networks return probabilities (or confidences) – specifically, for each bounding box, an object detection network would return a probability of there being an object of the specified class within that bounding box. At this point, one naïve method to assess network performance might be to set a threshold value of IoU above which we would deem a prediction 'correct', and a threshold probability (classification threshold) above which we would deem the network to have made a prediction, and simply compute accuracy as a percentage of correct predictions. However, this method does not give us the full picture of a network's performance, particularly in cases with imbalanced datasets. Say, for illustration, that we had a dataset of images containing 18 cats and 2 dogs: in

this case the heuristic 'label every object a cat' would achieve an apparent accuracy of 90%.

Two better metrics to use when evaluating network performance are those of *precision* and *recall*, defined as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{18}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{19}$$

where TP is the number of *true positives*, TN is the number of *true negatives*, FP is the number of *false positives* and FN is the number of *false negatives*. From these equations, we can see:

- Precision is a measure of the proportion of 'positive' predictions made that are correct (the proportion of bounding box predictions that are actually surrounding the right object) – low precision indicates a high proportion of *false positives* compared to true positives.
- Recall, on the other hand, is a measure of the proportion of the positives in the ground truth that are identified correctly (the ability to detect all the objects in the dataset) – low recall indicates a high proportion of *false negatives*, or in other words, many objects in the dataset that were not identified.[42]

Notice that the values of precision and recall depend very much on the classification thresholds set: a high threshold (only predictions the network is very 'certain' about count) will lead to a high precision but a low recall, and vice versa. We can visualise this relationship between precision and recall by calculating the values of precision and recall at different classification thresholds, and plotting the graph of precision $p(r)$ as a function of recall r (Figure 29).

One metric we can compute from this data is *average precision* (AP) – the average value of $p(r)$ over the interval $0 \leq r \leq 1$, or in other words, the area under the precision-recall curve:

$$\text{AP} = \int_0^1 p(r)\, dr \tag{20}$$

As we are only able to compute $p(r)$ for a finite number of values of r, to approximate this integral we can instead rank the predictions made by the network (across the whole dataset) in descending order of confidence. Notice that, if we set the classification threshold such that the first n elements in this list are above it, the resulting recall will always be less than or equal to the recall if the first $n + 1$ elements were above the threshold. Thus we can approximate the area under the curve as follows:

$$\text{AP} = \sum_{k=1}^{n} P(k)\Delta r(k) \tag{21}$$

[42] At this point, it is worth being precise about what is classed as a true positive / false positive etc. in terms of object detection: given some threshold T, a prediction is a true positive if the IoU is $\geq T$, a false positive if *either* the IoU is $< T$ *or* the model has predicted two bounding boxes around the same object (the duplicate bounding box is a false positive), and a false negative if the IoU is $\geq T$ but the object is classified wrongly. The lack of a bounding box around an object is also a false negative, and the lack of a bounding box around a non-object is a true negative.

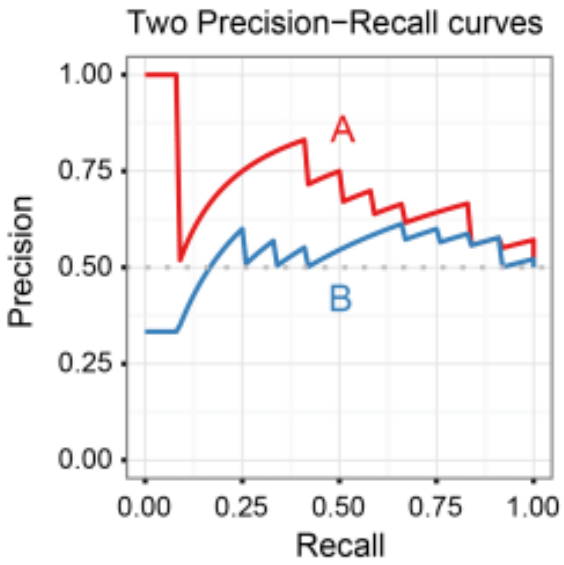


Figure 29: Two example precision-recall (PR) curves, representing the performance of two classifiers A and B. Here, classifier A clearly outperforms classifier B. [Roffo, 2017]

where $P(k)$ is the precision with a threshold set such that the first k elements in the list are above it, $\Delta r(k)$ is the change in recall from a threshold including the first $k-1$ elements to a threshold including the first k elements and n is the number of examples in the dataset [Su et al., 2015]. Some evaluation metrics, such as that used by the Pascal VOC[43] object detection challenge use an interpolation method to smoothen the curve, for instance ensuring the precision is monotonically decreasing by setting the precision for a given recall r to the maximum precision obtained for any recall $r' \geq r$ [Everingham and Winn, 2012].

One of the most popular set of object detection metrics are what are known as *COCO metrics* – the evaluation metrics used in the COCO[44] object detection challenge. The primary metric used here is what is known as *mAP@IoU=.50:.05:.95*: the mean of average precision values with IoU threshold varying from 0.50 to 0.95 with interval 0.05. Other common metrics include *mAP@IoU=.50* (Pascal VOC metric) and *mAP@IoU=.75*.

For the COCO dataset, the current state of the art is an mAP of 0.526, achieved by the Chinese AI startup Megvii [Lin et al., 2018]. A study by Google [Huang et al., 2016] analysing speed/accuracy tradeoffs in various object detection architectures for the COCO dataset yields mAPs as shown in Figure 30; the TensorFlow object detection 'model zoo' [Huang et al., 2019] contains COCO-trained models with mAPs ranging from 0.16 to 0.43.

[43]PASCAL Visual Object Classes [Everingham et al., 2012]
[44]Common Objects in Context [Lin et al., 2014]

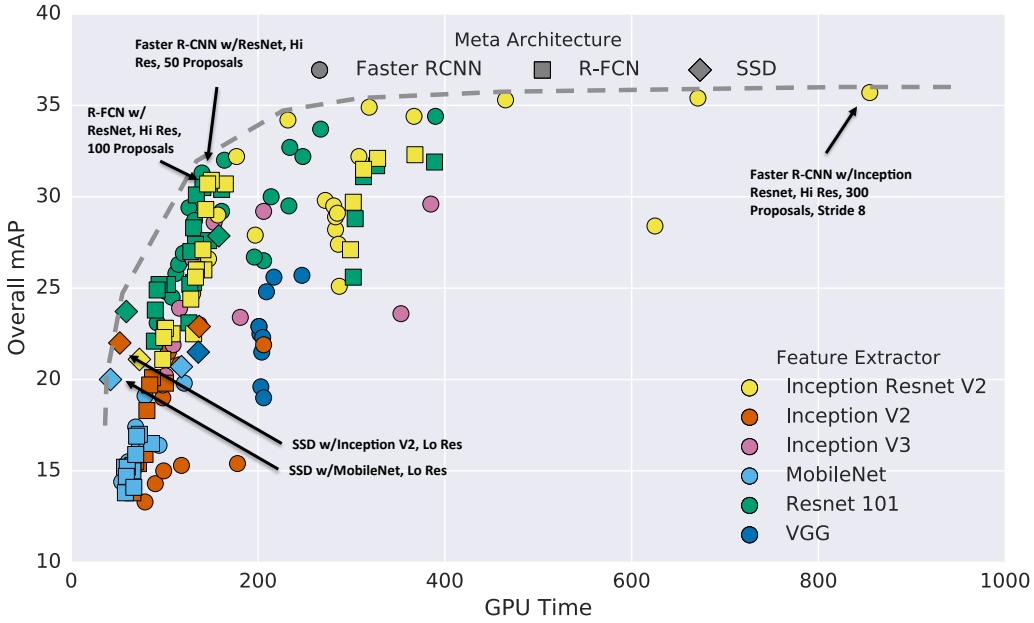


Figure 30: A graph of accuracy (percentage mAP) against per-image inference time. Colours indicate feature extractor; marker shapes indicate meta architecture. Each (meta architecture, feature extractor) pair can correspond to multiple points on the graph due to varying hyperparameters. [Huang et al., 2016]

D Transfer Learning

More specifically, transfer learning is the following problem: "given a source domain D_S and learning task T_S, and a target domain D_T and learning task T_T, transfer learning aims to help improve the learning of the target predictive function $f(\cdot)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$" [Lin and Jung, 2017].

The motivation for this process is the deep learning observation that layers in CNNs themselves — and by extension in networks such as Faster R-CNN — form a *hierarchy of concepts* (see Section 2), in that they progress from generic to specific. In the first few layers, CNNs often learn generic features similar to Gabor filters[45] and color blobs, whereas in later layers the CNN develops a high-level representation of the contents of the image, becoming progressively more specific to the classes of the dataset [Yosinski et al., 2014]. Thus a common practice in transfer learning is to 'freeze' the weights of the first few layers (i.e. to not modify these weights during training), and to only modify the weights of the last few fully connected layers, the idea being that, since the first few layers learn to detect general properties of an image (such as blobs, edges of objects and so on), the features extracted by the first few layers will be relevant to many different computer vision tasks, not just to the task the network was trained on.

This view is not universally held by the computer vision community, though: there is a growing school of thought that simply fine tuning all layers of the network – except in cases where there is a particularly small dataset – is more effective than freezing certain layers. [Chu et al., 2016]

[45] Gabor filters are linear filters used in image processing for low-level feature extraction and texture analysis.

E Methods

E.1 OxTD: Data Processing

E.1.1 RGB-Thermal Coordinate Mappings

In order to make sense of the RGB-thermal pairings provided in the OxTD, it was first necessary to compute a mapping between pixels in the RGB image and pixels in the thermal image. After analysing a number of RGB-thermal pairings, it became apparent that, with a geometric translation, one could superimpose the thermal image on the RGB image, while keeping both images at their original scales. Specifically, for some offsets m and n, we can map each pixel T_{yx} in the thermal image T with a corresponding pixel $R_{(y+n)(x+m)}$ in the RGB image R. Note that this alignment is very much approximate – although after alignment objects and features in the thermal image should be in similar positions to those in the RGB image, the perspective of the thermal camera is slightly different to that of the RGB-D camera. As these offsets would remain roughly the same for the duration of each run, but could change between runs (as they are dependent on factors such as the position of the RGB-D camera mount on the FLIR unit), we adjusted the offsets manually for each run until the thermal images were aligned correctly with the RGB images. This is demonstrated in Figure 31a.

Unfortunately, there are some isolated instances within a given run where the alignment deviates from that specified by the corresponding offsets – here, due to the not insignificant lag between visible and thermal image capture, it is likely that the camera perspective has shifted between capturing the visible and thermal image due to videographer motion (as in Figure 31b).

(a) An image demonstrating correct alignment

(b) An image demonstrating incorrect alignment, due to videographer motion

Figure 31: Examples of thermal images superimposed on RGB images. For ease of viewing, thermal images are converted to colour maps using a histogram equalisation method.

E.1.2 OxTD: Dataset Splitting

The OxTD contains on the order of $150K$ images; although more data generally improves model performance, the benefits of a larger dataset decrease with increasing size [Zhu et al., 2012], and a dataset of this size would be impractical to manipulate and store in memory for training. Thus, we construct[46] smaller training, test and validation datasets as follows:

[46]Note that we use *systematic sampling*, instead of random sampling, to select the datasets, as the dataset is comprised of frames from a video recording. Due to the fact that two adjacent frames in the

1. **Training Set**: 10,568 unlabelled pairs of RGB and thermal images (all images with index[47] 1 mod 15) – once the RGB images have been labelled by the RGB-trained network, the thermal images will form the training data for modality tuning.

2. **Validation Set**: 2,114 unlabelled pairs of RGB and thermal images (all images with index 28 mod 75) – once the RGB images have been labelled, the thermal images will form the validation data for modality tuning. The modality-tuned network's performance on this dataset will be an assessment of how close its labels are to the labels on the original network when given RGB images – i.e. *how well the first few layers in the thermal network have learned to detect the same features as the first few layers in the RGB network.*

3. **Test Set**: 127 *manually annotated* pairs of RGB and thermal images (all images with index 0 mod 500, with images containing no relevant or discernible objects removed). The purpose of these images is twofold:

   (a) The RGB network's performance on the RGB images of the test set (**OxTD Test Set (RGB)**) evaluates how well the network can detect objects of the relevant classes in the environments present in the OxTD, and will be used to select the network used for labelling the OxTD Training and Validation sets.

   (b) The thermal network's performance on the thermal images of the test set will be an assessment of how close its labels are to the ground truth – i.e. *how well it can detect objects of the relevant categories.*

E.2 RGB Dataset Analysis and Downsampling

The ISR dataset, developed primarily for the task of *indoor scene recognition*[48], contains $15K$ images, a subset of which are annotated with the objects they contain. After some data cleaning, this dataset yielded annotated examples of 1639 windows, 3170 chairs, 849 doors and 1574 tables; these examples were all from indoor environments. The OID is a dataset of ∼9 million images annotated with bounding boxes spanning 600 categories; these are very diverse, taken in indoor and outdoor environments. This dataset yielded annotated examples of 29,862 windows, 15,693 chairs, 4353 doors and 7043 tables. The fact that the OID contains indoor and outdoor environments is potentially a disadvantage, as the OxTD, which we are attempting to label, consists of only indoor images (and 'windows' in outdoor environments, such as car windows, look substantially different from windows inside a building) – nevertheless, its size compared to the ISR dataset should make up for this.

Note that, in both datasets, there is a significant class imbalance: in the ISR datasets, there are fewer examples of doors and more chairs than other categories, and in the OID this problem is even more pronounced, with almost seven times more windows and four times more chairs than there are doors. In cases where class distributions are highly imbalanced, models trained on this data often suffer from what is known as the *class imbalance problem* – that is, they have low predictive accuracy for the infrequent class

dataset will be very similar, to ensure as little repetition of data as possible we sample systematically (every nth image) in order to ensure an even distribution of environments within the dataset.

[47] Here, the index of an image refers to its position in the dataset when all images are sorted in chronological order.

[48] Given an image, name the environment in which it was taken – e.g. kitchen, living room, restaurant.

[Ling and Sheng, 2010]. Thus, we performed downsampling[49] on the OID in order to resolve this issue, leaving us with examples of 5458 windows, 7583 chairs, 4353 doors and 7043 tables. As the ISR dataset is already very small, however, and the imbalance is less severe than that of the OID, we decided to leave the dataset as-is: it is likely that the decrease in performance from losing a significant fraction of training examples would be more significant than that caused by the imbalance in class distributions.

E.2.1 Thermal Image Preprocessing

Since the thermal images in the OxTD have only one channel (luminance), and the RGB images, on which the ResNet feature extractor operates, have three (red, green and blue)[50], we must modify either the thermal images or the first few layers of the network in order to use the network with thermal images.

Possible approaches include:

1. Duplicating the raw luminance values of the thermal image and using these as input for each of the three channels (i.e. for each pixel, the red, green and blue channel values all equal the luminance for that pixel).

2. Adding another convolutional layer before the first layer of the feature extractor, taking as input a tensor of dimensions $(h, w, 1)$ (where h, w are the height and width of the input image[51]) and containing three kernels. This layer will have an output volume of dimensions $(h, w, 3)$, which then forms the input of the first layer of the feature extractor.

3. Converting the thermal images to RGB colour maps using methods such as histogram equalisation (as in Figure 31) and using these as three-channel inputs for the feature extractor.

Similar studies on object detection from thermal images [Jangblad, 2018] have shown that method (i) performs better than methods that modify the structure of the network, such as (ii). Thus, we used method (i) to transform the 1 channel thermal images into 3 channel images that can be used as inputs for the Faster R-CNN.

E.3 Faster R-CNN Anchor Scales

The dimensions of proposed anchor boxes are important as they are used by the RPN to determine possible RoIs in which objects could be – thus the range of anchor boxes must be such that almost all objects in the dataset have dimensions similar to those of at least one anchor box.

[49] Downsampling is the process of removing examples of the majority class(es) from the dataset in order to achieve a more balanced class distribution – i.e. training on a disproportionately low subset of the majority class examples.

[50] The thermal images form a matrix of shape (348,464), with each pixel containing a single luminance value between 0 and 65535; the RGB images form a third rank tensor of shape (348,464,3) with each pixel containing three values for the respective channels. Note that, as part of preprocessing, the values of each channel of each pixel are normalised as floats from 0 to 1, and the dataset channel-wise mean subtracted from the respective channels of each pixel.

[51] As the region proposal network and the ResNet feature extractor are fully convolutional [Ren et al., 2017], the input image can be of arbitrary dimensions (above some minimum size).

To determine these parameters, we can examine the dimensions of bounding boxes in the OxTD (Test). From the set of graphs in Figure 32, we can see that most bounding boxes are within the range of 0 to 350 pixels – thus the original anchor scales of 64, 128, 256 and 512 pixels are suitable for this dataset. In an attempt to improve detection for small items within the dataset, we test two different anchor configurations: the anchor scales listed above ({64,128,256,512}), and the anchor scales ({32,64,128,256,512}). In terms of aspect ratio, as shown by the graph the majority of the points fall between the lines $y = 2x$ (aspect ratio 0.5) and $y = \frac{1}{2}x$ (aspect ratio 2). Thus a sensible choice of aspect ratios for anchors would be $\{0.5, 1.0, 2.0\}$. In addition, considering the actual objects to be labelled, very few windows, chairs, tables or doors have aspect ratios less than 0.5 or greater than 2 – the spread either side of the lines in the graph could be accounted for by the presence of objects in the dataset that are partially occluded and hence have particularly low or high aspect ratios.

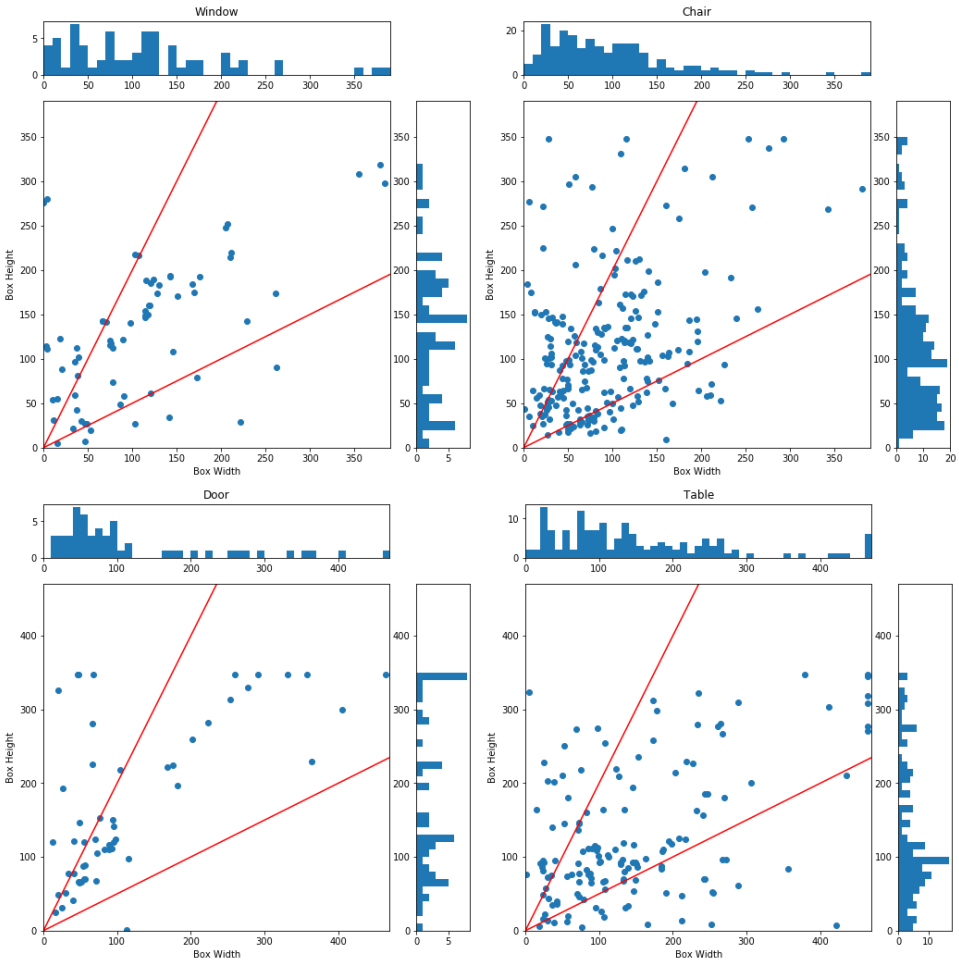


Figure 32: Scatter plots of bounding box width against bounding box height for each of the four categories in the OxTD Test Set. Histograms of distribution of box width and box height are to the top and right of each graph respectively. The red lines indicate points with aspect ratio 0.5 and aspect ratio 2 – any point above the top red line has aspect ratio < 0.5 and any point below the bottom red line has aspect ratio > 2.

F Results

F.1 Category-wise Performance

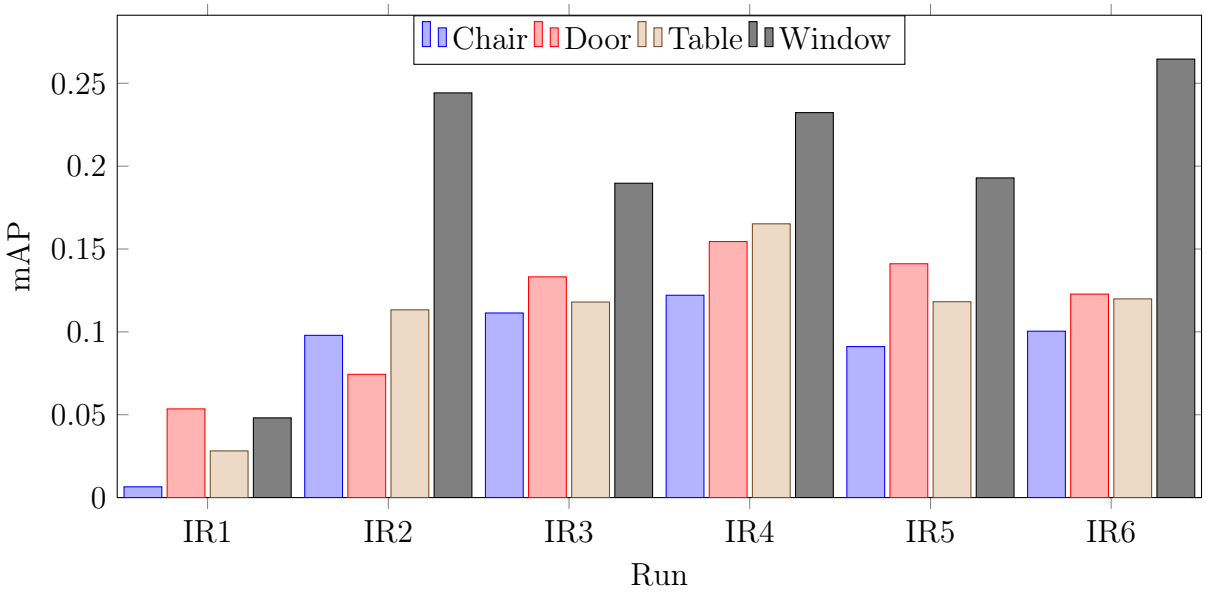


Figure 33: A bar chart illustrating per-category performance (AP@.50:.05:.95) for each of the six IR runs.

The bar chart in Figure 33 illustrates the AP per category[52] for each IR run. Looking at these graphs, it is evident that the network recognises windows very well, but struggles somewhat to recognise chairs. One possible hypothesis for this could be uneven category distributions in the dataset. However, on analysis of the training dataset annotations (as shown in Figure 34), there is significant randomness (and class imbalance) apparent in distributions of training dataset classes, whereas the distributions of category-wise AP stay relatively consistent across runs.[53] This suggests that the comparatively high mAP of the 'window' class is likely either due to windows being easier to recognise (or of a more consistent shape/size) than doors, tables or chairs, or due to the fact that although the network is effective at labelling chairs, doors and tables, it is prone to recognising other objects incorrectly as chairs, doors and tables (as evidenced in Appendix F.2). Again, precision-recall curves would allow us to further investigate this phenomenon; this has not been possible, however, due to time constraints.

[52] For each run, mAP is computed as the average of APs across all categories (see Appendix C for more details).

[53] It is pleasing to note that, although the frequency of examples for chairs, doors and tables in the OxTD training set varies wildly between runs, there is much less variation in the APs of chairs, doors and tables on the OxTD test set, suggesting that the network is able to learn to detect these objects with a consistent level of precision regardless of class distribution.

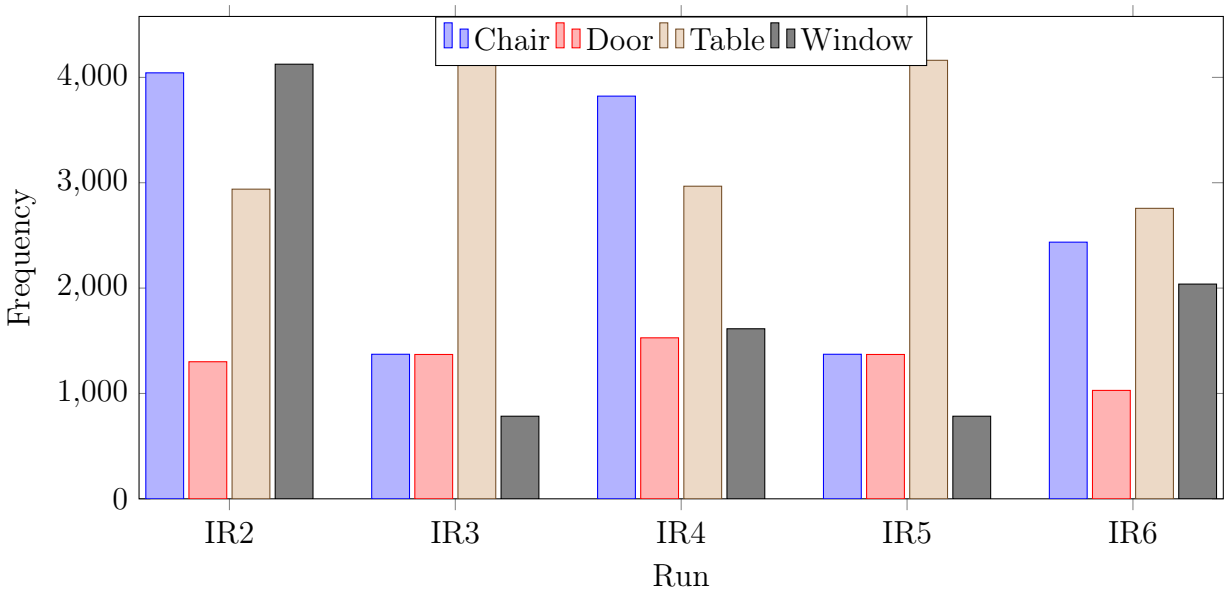


Figure 34: A bar chart illustrating class distributions in the training datasets used for each of the five non-baseline IR runs.

F.2 Example Network Annotations

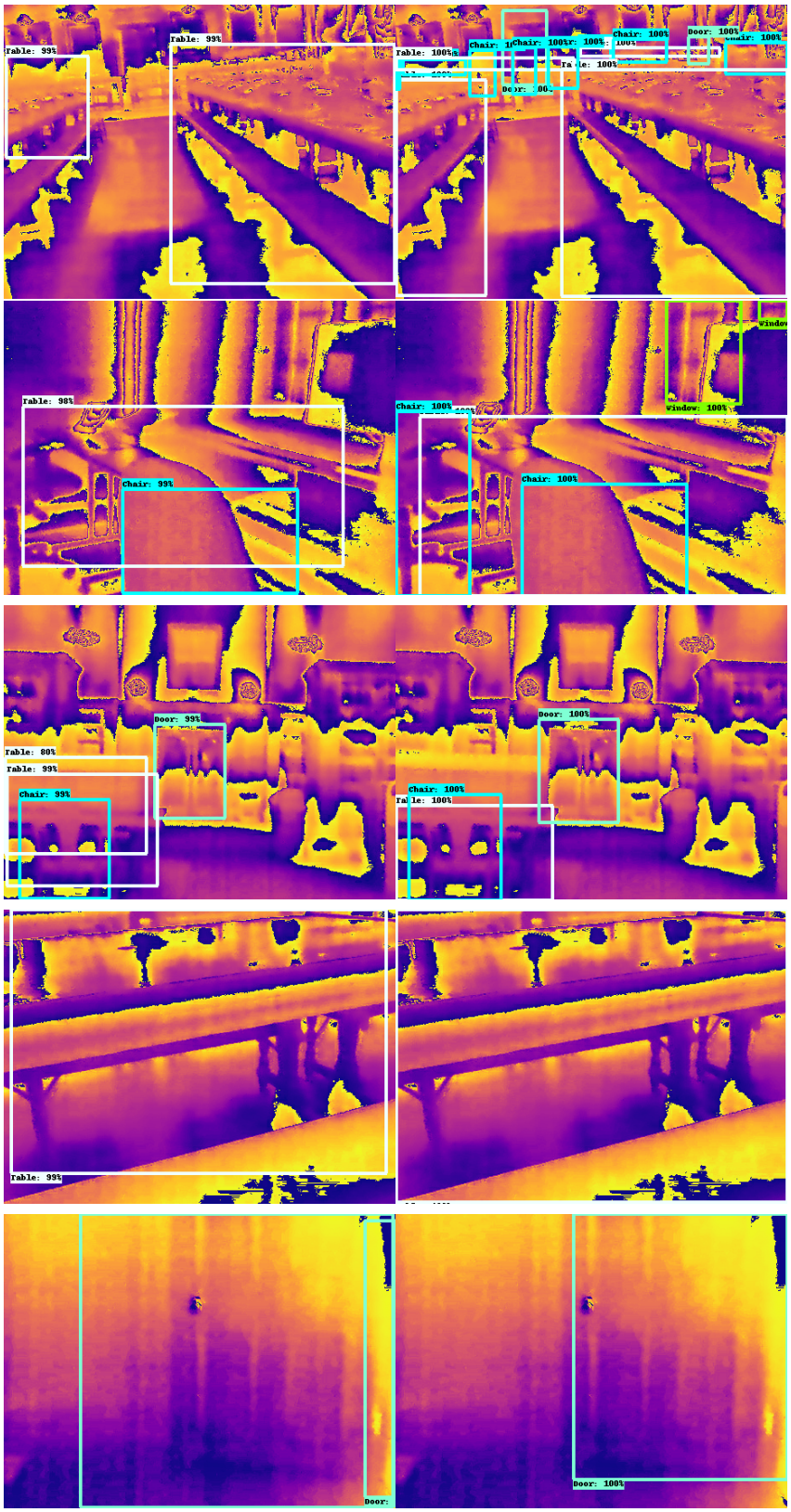


Figure 34: 10 images randomly sampled from the OxTD thermal test set. Ground truth annotations are on the right; network (IR4) annotations are on the left. Chairs are annotated in light blue, doors in turquoise, tables in white and windows in green. Each annotation is labelled with its class and the network confidence (as a percentage).

G Further Work

We can divide further work into two categories: improvements to the current experimental method, and new directions in which to carry out research.

G.1 Improvements

Datasets, Labelling and Data Collection

1. *More hand-annotated images.* One way to avoid inaccuracies caused by automatically labelling images is simply to label all images manually – however, given that one would need to hand label a few thousand images in order to obtain a suitably sized training dataset, this would be a very significant endeavour.

2. *Active learning.* An alternative method to labelling all images is to implement what is known as *active learning* – this is a semi-supervised machine learning method in which the algorithm is allowed to interactively request that a given image be labelled by the user. In general terms, this involves the user hand-labelling a small set of images as an initial training set, the network training itself on these images and then attempting to expand its training set by labelling more images. Out of these images, it will request that the user relabel those whose labels it is least confident about. This process continues until the whole dataset has been labelled [Roy et al., 2018]. This is a compromise between accurate but slow human labelling, and inaccurate but fast automatic labelling.

3. *Refining RGB-thermal coordinate mapping.* While the simple method of approximating a single translation for all images of each run works reasonably well, there are instances where there is a temporal lag between a given thermal and RGB image. To improve the accuracy of this coordinate mapping, one possible method could be to, for example, detect features, edges and points of interest in both the RGB and thermal images, and to find the thermal image which best matches the RGB image (within a certain range of time relative to the RGB image).

Training and Network Architecture

1. *Systematic treatment of hyperparameter tuning.* An important improvement to make to the training process would be to systematically perform hyperparameter optimisation, using techniques such as grid search (an exhaustive search through a subset of hyperparameter space for a given model) or even Bayesian hyperparameter optimisation [Klein et al., 2017]. These would be used to tune not only network hyperparameters but also, for instance, thresholds in the majority voting algorithm. In addition, it would also be prudent to investigate the effect of using different learning rate decay schedules when performing modality tuning and fine tuning – while we briefly investigated this, results were not forthcoming.

2. *Further investigation into modality tuning.* While the experiments detailed in this report provide evidence to suggest that modality tuning is just as effective as fine tuning when transfer learning across modalities, this claim should be investigated further – for instance, we could experiment with freezing different numbers of layers of the feature extractor, and explore how this affects performance.

3. *Exploration of alternative object detection architectures.* Alongside *two-stage* object detection architectures, such as Faster R-CNN, other state-of-the-art object detection networks include *one-stage*, or 'single shot' networks such as SSDs[54] [Liu et al., 2016] and the YOLO[55] network [Redmon et al., 2016] – we can try performing similar experiments using different network architectures and comparing the results. In addition, we could experiment with varying the feature extractor CNN used in the Faster R-CNN network.

Evaluation

1. *Estimating uncertainty in network performance.* If we had more time, it would be useful to perform repeats of each training run (without varying hyperparameters): as training is a stochastic process, there will always be some variance present in the results. This would also allow us to estimate the uncertainty in the values of mAP quoted for a given network configuration (architecture and hyperparameters).

2. *Precision-Recall curves.* To further investigate a number of the phenomena observed in Section 5.2, it would be useful to plot category-wise *precision-recall curves* for each run (as detailed in Appendix C.2).

G.2 Extensions

1. *More classes.* The *Government Breathing Apparatus Search Handbook* [DFRMO, 2014] references the following items (windows, tables, chairs, beds, wardrobes, cupboards, baths, chests, fridges, doors, furniture and people) as important for first responders to identify and search in, under or around. Once an initial proof-of-concept pipeline is implemented, we could investigate extending the approach to work with a larger set of classes relevant to the target environment.

2. *Real-time object identification.* As this network will likely be used for real time object detection (e.g. for semantic mapping / to aid firefighters in locating casualties or hazards), it is worthwhile investigating whether we can optimise the network for this purpose. In addition, since in the field the network would run on an embedded device, memory considerations are also important. We would most likely need to look into efficient, low memory feature extractors such as MobileNet [Howard et al., 2017], and object detection architectures with a lower inference time per image such as the YOLO network. Here, the balance of speed and performance is crucial, and will need to be carefully investigated. Also worth considering is whether the network should be run on an embedded device attached to the FLIR, or whether it will be possible to send or stream images to a more powerful base unit which can perform inference more quickly.

3. *Environment-specific data collection / augmentation.* As the target environment for this network is that of a burning building, it would be useful to train the network on thermal images that more closely simulate this environment. There exist many FLIR videos taken within burning buildings for the purpose of firefighter training – these, however, would need to be annotated. An alternative approach is to simulate the effects of fire in thermal images.

[54] Single Shot Multibox Detector
[55] You Only Look Once

4. *Burning objects.* In actual firefighting scenarios, the network will need to be able to adjust for high variance in temperature (e.g. a very hot object in the room). It would be worthwhile to investigate whether or not the network can cope with images with a very high dynamic range.

5. *Obstructions / occluded images.* If most of the object is blocked in the thermal image (e.g. if a window is covered / distorted as it burns), it is important that we are still able to detect the object (or better yet, reconstruct the occluded part of the image, in the manner of [Yun et al., 2018]). In order to do this, we could try randomly occluding or obscuring parts of thermal images before they are used for training.