INVESTIGATION INTO INTELLIGENT IMAGE PREPROCESSOR TECHNIQUES FOR ARTIFICIAL NEURAL NETWORKS

A THESIS SUBMITTED TO

THE UNIVERSITY OF KENT

IN THE SUBJECT OF COMPUTER SCIENCE

FOR THE DEGREE

OF PHD.

By

Keith A. Greenhow

July 2017

Abstract

In this thesis we will discuss the process for data preparation of visual or image data ready for use in Artificial Neural Network systems. The thesis will present these concepts, their location in the broader field and the arguments as why certain practices are considered required for these systems; before presenting a number of novel algorithms that are intended as alternatives with desirable properties. These novel algorithms will then be testing in a practical domain (simulating the challenge of face-detection within a scene), followed up by discussions of their successes and failures. The findings presented show that some of the novel algorithms can show statistically significant improvement in accuracy compared to some of the traditional methods used in the field. This thesis concludes with recommendations in which situations the novel algorithms may (if at all) be suitable for use in future designs and potential avenues for further research.

Acknowledgements

I would like to thank Colin G. Johnson and all the staff in the Computer Science Department that have helped me through this process over the years. Without the support provided by them, I would have not completed this thesis.

I would like to also thank my family whom have had to put up with me for the duration of the PhD.

Contents

Abstract **ii**

Acknowledgements **iii**

Contents **iv**

List of Tables **ix**

List of Figures **xi**

Glossary **1**

1 Introduction **3**

2 Literature Review **7**

2.1 Artificial Neural Networks 8

2.1.1 Feed-Forward Perceptrons 8

2.1.2 Convolutional Neural Networks and Receptive Fields . . . 10

2.1.3 Curse of Dimensionality 14

2.2 Traditional Image Interpolation 16

2.2.1 Bilinear Interpolation 16

2.2.2 Bicubic Interpolation 19

2.2.3 Other Interpolators 21

2.2.4 Comparisons 23

2.3 Frequency Transformations 27
2.4 Related concepts 28
2.4.1 Visual Attention 28
2.4.2 Simulated Foveation 30
2.4.3 Superpixels 33
2.5 Related Implementations 34
2.5.1 MIPs based Preprocessor 34
2.5.2 A Superpixelwise CNN 34
2.5.3 ANNs and Improved Binary Gravitational Search Algorithm 35
2.6 Issues in the Literature 35
2.7 Summary 39

3 Research Methodology 41
3.1 Domain Commonalities 41
3.1.1 Hardware and Background Processes 42
3.1.2 Control Groups: Traditional Preprocessors 46
3.1.3 Stochasticity 46
3.1.4 Cohen's Kappa 46
3.1.5 Receiver-Operator Characteristic Curve 47
3.2 Test Domain 1: Object Presence Detection 48
3.2.1 Implementation Overview 48
3.2.2 Data Sets 51
3.2.3 Analysis 51
3.3 Test Domain 2: Baseline Face Detection 52
3.3.1 Implementation Overview 52
3.3.2 Prediction Merging and Quantifying 54
3.3.3 Datasets 56
3.3.4 Subset construction 57
3.3.5 Analysis 58

4 Image Downsampling and Preprocessors **59**

5 Voronoi Saliency Partition **65**

5.1 Voronoi Diagrams 66

5.2 Saliency: Measuring an Neurons 'Worth' 67

5.2.1 Foundation: Relevance Assessment 68

5.2.2 Saliency: Definition 70

5.2.3 Alternate Uses 72

5.3 Research Question 72

5.4 Preprocessor: Cellular Downsampling 73

5.4.1 Initial Site Positions 75

5.5 Testing the Downsample 75

5.6 Update: Adaptive Site positioning 78

5.6.1 Particle Inspired Update 79

5.6.2 Neighbourhood Watch 81

5.6.3 Powered Neighbourhoods 84

5.7 Testing Site Update 84

5.7.1 Results 85

5.8 Conclusion 87

6 Regional Downsampling Algorithms **91**

6.1 Terminology: Region 92

6.2 RDA as a Preprocessor 93

6.2.1 Comparison to CNNs 94

6.3 Research Question 95

6.4 Region Selection 95

6.4.1 Setup: Saliency Computation 96

6.4.2 Saliency Heat Map Selection 97

6.4.3 Restricted Saliency Heat Map Selection 97

6.4.4 Pruned-Bilinear Selection 100
6.5 Optimising for Runtime 102
6.6 Inter-comparisons of RDAs 103
6.6.1 Similarity 103
6.6.2 Likeness 104
6.7 Situating 105
6.8 Results 105
6.8.1 Test Domain 1 105
6.8.2 Test Domain 2 107
6.9 Conclusion 112

7 Subdividing Preprocessor 117
7.1 Subdividing as an Adaptive Preprocessor 118
7.2 Research Question 118
7.3 Region Selection and Subdivision 119
7.4 Situating 121
7.5 Parameter Selection 122
7.6 Results 124
7.6.1 Caltech Dataset 125
7.6.2 CMU Dataset 127
7.7 Conclusion 129

8 Evaluation and Discussion 134

9 Conclusion and Further Work 138
9.1 Future/Further work 139

Bibliography 141

A VSP Image Sources **149**

A.1 Stuffed Bear 149

A.2 Backgrounds (17 images) 150

List of Tables

1 Computation times to down-sample the 'Lena' image by a factor of 3 using each method. Percentage times are present as quick comparisons to the baseline (*) for each family. 26

2 Average execution time of various interpolation techniques against the CMU-130 data set. 27

3 Average computational Runtime ($\overline{time}$) for the VSP Down-sampling Algorithms and Control Group with standard deviation (σ) 77

4 Computational Runtime for the precomputed VSP Down-sampling Algorithms 77

5 Parameter domains for the Site Update tests. 85

6 Aggregate results for Test Domain 1. 107

7 Aggregate results for the tests against the Caltech Data Set. . . . 107

8 Post hoc Wilcoxon Signed-Rank tests for RDA Accuracy on the Caltech Data set 108

9 Aggregate results for the tests against the CMU Data Set. 109

10 Post hoc Wilcoxon Signed-Rank tests for RDA Overall Computation Time on the Caltech Data set 110

11 Post hoc Wilcoxon Signed-Rank tests for RDA Accuracy on the CMU Data set 111

12 Post hoc Wilcoxon Signed-Rank tests for RDA Overall Computation Time on the CMU Data set 113

13 The Likeness measure between each Region Set 114
14 The Similarity measure between each Region Set 114
15 Aggregate results for the tests against the Caltech Data Set. . . . 125
16 Post hoc Wilcoxon Signed-Rank tests for Subdividing Accuracy on the Caltech Data set 126
17 Post hoc Wilcoxon Signed-Rank tests for Subdividing Overall Computation Time on the Caltech Data set 126
18 Aggregate results for the tests against the CMU Data Set. 127
19 Post hoc Wilcoxon Signed-Rank tests for Subdividing Accuracy on the CMU Data set 128
20 Post hoc Wilcoxon Signed-Rank tests for Subdividing Overall Computation Time on the CMU Data set 128
21 Distance metrics (Sec. 6.6) of the three random selected region sets generated as part of the Baseline Face Detection test against the CMU-130 data set. See figure 31 for their visual appearance. . . . 130
22 An aggregate of the results for both the RDA and Subdividing tests against the Caltech Data Set. 136
23 An aggregate of the results for both the RDA and Subdividing tests against the CMU-130 Data Set. 137

List of Figures

1 A diagrammatic example of an image processing system, that uses a search algorithm to simplify the task. In this example, P, P_1 and P_2 are potential locations for image preparation to the subsequent search or classification tasks. 6

2 Receptive Fields. A diagrammatic representation of Receptive Fields, showing the down-sampling effect that they have on the previous layer. This example shows a $5 \times 5 \times 3$ input volume and the aggregating relationship to the output volume using a $3 \times 3 \times 2$ receptive field. 11

3 Bilinear Interpolation. In the above figure the notation (x, y) describes a 2 dimensional position, whilst the notation $[R, G, B]$ is used to describe the colour with red, green and blue components, each in the range 0-255, inclusive. Any sub-pixel point within an image can be assigned a colour with only 3 linear interpolations. 1) The original 2×2 pixels and the target point. 2) The first 2 linear interpolations moving right along the x-axis by 40% of a pixel. 3) The third interpolation, using the colour determined from the first two and moves down along the y-axis by 70%. 4) The result of performing a linear interpolation of all sub-pixel points of the original 4 pixels. 17

4 Bicubic Interpolation. In the above figure the notation $[G]$ is used to describe the grey-scale value in the range 0-255 (inclusive). f is the single dimension cubic function (Eq. 9). 1) The target point to be interpolated, P, and the surrounding 4×4 samples. 2) Applying the cubic function to each row, using $t = 0.2$ from P's x-offset. 3) Appying the cubic function to the vertical results, using $t = 0.7$ from P's y-offset. P is computed a value of 178 ($\approx$ 70% black); if bilinear was used, a value of 157 ($\approx$ 61% black) would have been computed. 20

5 The original Lena image and a selection cropped interpolated images. Each image is the result of down-sampling the Lena image by the specified method and then up-sampling by Nearest Neighbour for inspection. The blocky transition at the edge of the hat is due to the use of Nearest Neighbour and thus makes for un-fair comparisons cross such hard edges in this situation. Comparisons can be made about the textured surfaces and smoother edge transitions. Image (a) shows the roughest interpolation, with moiré patterns resulting from shadows on the woven surface. This roughness is not seen in the other images, but almost all visual indication of the hats texture have also been lost. Looking at the creases in the fabric and at the feather, images (d) and (e) show clearer edges than images (b) and (f) due to the overshoot present in their respective interpolation methods. 25

6 The General architecture of Itti, Koch and Niebur's model; as depicted in their original paper. 31

7 Diagram of the human eye (March i Nogué 2016). It shows the lower part of the right eye after a central and horizontal section. The fovea centralis can be located towards the bottom of the image (labelled Fovea). 32
8 The accuracy (measured using Cohen's kappa) as a function of down-sampling resolution. 49
9 A graphic representation of the data flow through a complete system, from source image to final classification. 50
10 Three sample images from three data sets. These provide examples of a single image with multiple faces, an image with a single face proportionally small to the image, and an image with a face that fills most of the image. 57
11 A representation of the additional phase for a Bespoke Preprocessor. P′ represents some sample preprocessor(s) used during the construction phase. Eval represents an evaluation function used to quantify the ANNs behaviour, with respect to P′. Activities, such as a Trial-And-Error refinement of a preprocessor, are represented by the Construction Phase. The preprocessor, P, used in the main learning phase and into production may be constructed given the results of Eval over the set of {P′}, or the instance from that set that allowed the ANN to score the most preferential value from the Eval function. Note that in this diagram an ANN using a supervised learning algorithm has been depicted. This is not a requirement of, but rather an easily representable and recognisable way to show where the ANNs traditional learning fits within this life-cycle. . . 60

12 A representation of the life-cycle of an ANN based system using an Adaptive Preprocessor. Note the addition evaluation function, Eval, and an Update process added to the Learning Phase process, used to modify the preprocessor in parallel with traditional ANN learning. 63
13 An example Voronoi Diagram of 50 randomly located sites in a planar euclidean space. The sites are highlighted in red, with cell boundaries marked in black. 67
14 This figure demonstrates the downsampling of an image from the Training Set. The source image is reduced to a vector of 300 values (100 cells, each with three colour channels) and the cellular image reproduced from the Voronoi Diagram used to perform the downsampling. 74
15 Figure showing the accuracy per epoch of the various VSP downsampling algorithms (Flat [Dotted Lines], Inverse Distance [Dot-Dashed Lines] and Inverse Distance-Squared [Dashed Lines]), their initialisation method (Rectilinear, CVT or Random) against the two control preprocessors (Bilinear and Bicubic [Solid Lines]). Standard deviation above and below the mean for each epoch are presented. 76
16 Figure showing the accuracy per epoch of the precomputed variations of the three presented VSP down-sampling algorithms, all using the Rectilinear initialisation method. 78
17 The figure shows the same Voronoi Diagram as Fig. 13, but shows the Voronoi Neighbourhood (green) of a selected cell (blue). . . . 83
18 VSP using Update Algorithm 1 86
19 VSP using Update Algorithm 2 89
20 VSP using Update Algorithm 3 89

21 Control Group: Traditional Methods 90
22 An ANN system (using Update Algorithm 3) early in its training (left) and at completion (right). In this sample, The system failed to stabilise until all the Voronoi sites clustered around point (0,0). 90
23 An example layout of 7 regions at a variety of resolutions (bottom left corner of each region) and different positions (top left corner). In this example none of the regions overlap and all the space is covered by a region for clarity and simplicity; these are not restriction imposed by regions themselves. 93
24 Representation of the implementation of a preprocessor using an internal RDA. This diagram also displays the internal component topology and data flow through the RDA. Each Bilinear block computes the equivalent output of the Bilinear Interpolation algorithm for it the single pixel represented by the associated Region. 94
25 The SHM generated for the baseline face finding task (Section 3.3). Both images have been normalised, with the Caltech image (25a) originally ranging 0.55 to 0.67 and the CMU image (25b) ranging 0.59 to 0.69. For the Caltech data set, the algorithm has focused on the perimeter of the face, cheeks, forehead and an anomalous region in the top right corner. Against the CMU data set the algorithm has highlighted the forehead, eyes, noes and jawline. 100
26 A diagram of an RDA constructed using the Region Set optimiser for reduced computational overheads. 103
27 The Region sets produced by (a) SHM, (b) Restricted-SHM and (c) Pruned-Bilinear. All regions at the same resolution share the same colour. 115
28 A visual representation of the subdividing event example. 122

29 The results from a brute-force parameter optimisation for the Subdividing Algorithm. Initially a linear best-fit was performed showing a slight positive correlation for T_{upper} and a strong negative correlation for T_{lower}. This was later replaced with a cubic best-fit (shown) to determine some finer grain details in the results. The graphs for both data sets appear highly similar, allowing for the attribution of the data-set independent parameters $\mathrm{T}_{upper} = 0.95$ and $\mathrm{T}_{lower} = 0.15$. 123

30 A sample of preprocessors from each of the Subdividing trials. Figures (a) and (d) where selected as representative of the typical density (though layout is highly variable). Figure (b) is an example of a Preprocessor-ANN system that failed to learn the task to a high degree of accuracy, and thus little has occurred in the way of subdividing events. Figure (c) is an example of the ANN overfitting, affecting the preprocessor produced. Each colour represents a unique resolution 132

31 A random sample of three resultant Region Sets from the produced against the CMU-130 data set. 133

Glossary

Preprocessor

The term preprocessor is used in the specific context of an image preprocessor as used by an ANN. It can be described as a data preparation stage, process or algorithm in which image data is normalised, mapped/ filtered, colour corrected, colour space converted and/or resized before being presented as a sequence of 'features' to a subsequent ANN for task specific processing. The usage of the term also encapsulates the algorithm(s) used in specific the preprocessor's construction and the parameters required to initialise it.

Downsampling

Downsampling is the process by which an image's dimensionality (i.e. its size in pixels) is reduced. Traditionally, this is done by a graphical interpolation method designed to approximate what the image would have been, had the image been originally produced at the target resolution, given the original pixels at the current resolution. This will be covered in Section 2.2

System

A generic term for a Preprocessor-Classifier pairing when the specific preprocessor is not known/identified or that multiple types of preprocessor are being discussed as a whole.

Input Context

Used to refer to the additional information or properties that are part of a feature, but are not explicitly expressing in the numerical representation of that feature. This is the additional information that is typically discarded from general discussion, when dealing with ANNs, as it is redundant on the condition that it is constant. Examples of Input Context may include: the location of a pixel from an image, the field details from a database record or the unit/dimension from which the input draws upon. If the Input Context of an input feature changes, then any predictions/classifications made by the ANN that are dependent on that input will not be as accurate. Keeping with the example of pixel location as a type of Input Context, if an ANN is trained to perform a particular task and its nth input feature has a non-insignificant effect on at least one output, then changing the location used by the preprocessor to generate the nth feature would likely lead to errors in any dependent outputs. Conversely, any feature that is not used in the determining of any output (all routes leading from the particular input towards the output are terminated with a zero or near-zero weight) can be freely manipulated without affecting the quality of output.

Chapter 1

Introduction

This thesis will look into a part of Image Processing using Machine Learning. Specifically, the image preprocessor used as part of the data preparation stage for Machine Learning tasks. Though this will be focused on Artificial Neural Networks as a representative of Machine Learning, some of the concepts and methods presented here will be applicable and transferable to other sub-fields of Machine Learning. The fundamental question that lead to this investigation is:

> "Why are simple rectilinear down-sampling techniques, such Bilinear and Bicubic Interpolations, used to prepare data for Machine Learning tasks, given that their design was for subjectively pleasing visual re-sampling of images; and can other techniques do better?"

This question has yet to be answered in the literature though recent progress in the field show that there may be attempts to unknowingly overcome problems relating to this, which will be discussed later.

The motivation behind this question additionally stems from the prevalent use of Convolution Neural Networks (CNNs), as is evidenced by their ubiquity in the literature. CNNs have the advantage of not requiring any image preprocessors to be as aggressive, allowing more information to be processed, but at the cost of runtime complexity. This additional cost may be perceived as negligible by

most when considering modern CPU/GPU/Server processing power available to perform these computations; but on embedded hardware, where computational resources are at a premium, the attempt uses a CNN based algorithm may run up against difficulties.

A technology video published by Canon Inc. 2009 confirms that the company Canon has been using neural network based technologies in some of their digital cameras since its release 2009 with the release of the Digic 4 processor. Other manufacturers, in order to protect patents and competitiveness, provide less information about the internal workings of there processors. The patents Mashito (2001); Nikon (2007) show that the Nikon Corp. use a face detection process that utilises feature extraction of " end points of eye brows, eyes, nose, and lips, contour points of the face, a top point of the head, a lower end point of the chin, and the like" in preparation for an unspecified face recognising process.

The overall aims of this work is to develop a tools for the generation of image preprocessors which, when applied to feed-forward neural networks, provides an alternative solution than CNNs with a reduced computation and memory footprint. If successful, this could provide benefits in the form of more computational resources remaining available of other activities a given system, or for a reduced need in hardware resources (decreasing IC component costs and/or circuit-board footprint).

The novel contributions put forward by this thesis are, a framework with which to classify and aid discussion of image preprocessors, a metric which can approximate the contribution of a neural unit or input feature to an Artificial Neural Networks function, three novel image preprocessors and the process with which to generate them, and finally a methodology to systematically compare these presented preprocessors.

From the work presented in later chapters, three papers were published:

Greenhow and Johnson (2014)

This paper presented the early work that lead to the foundation of Chapter 6, presenting the Regional Downsampling Algorithm and using the Saliency Heat Map algorithm, as presented in the early portion of that chapter, up to Section 6.4.2 (inclusive).

Greenhow and Johnson (2015)

The second paper presented two further selection methods, covered in Sections 6.4.3 and 6.4.4.

Greenhow and Johnson (2017)

The third paper presented the Subdividing Image preprocessor. This algorithm, introduced in Chapter 7, took what was learned from the Regional Downsampling Algorithm and its selection methods and applied them to earlier ideas.

The potential practical applications for the work presented in this thesis can be numerous in task, but limited to the data preparation for an image classification task or as a tool in the exploration of an image set. Figure 1 shows flow diagrams for two simple image classification tasks in which image preprocessors can be useful. The three main industrial fields that are consumers of image classification technologies are Manufacturing Automation (object tracking, quality control, human safety), Security (human tracking, weapon identification, autonomous drones) and Human Interface Design (face detection, eye tracking, emotion identification).

As a prerequisite to the discussions of the models presented, a coverage of the background literature will be provided. Chapter 2 will contain a review of the literature and provide descriptions to the reader of the concepts that are foundational to the models and a look into the issues in the literature that provided the original line of questioning. Chapter 3 covers the methodological process which will be used to evaluate the effectiveness of the presented preprocessors in terms of

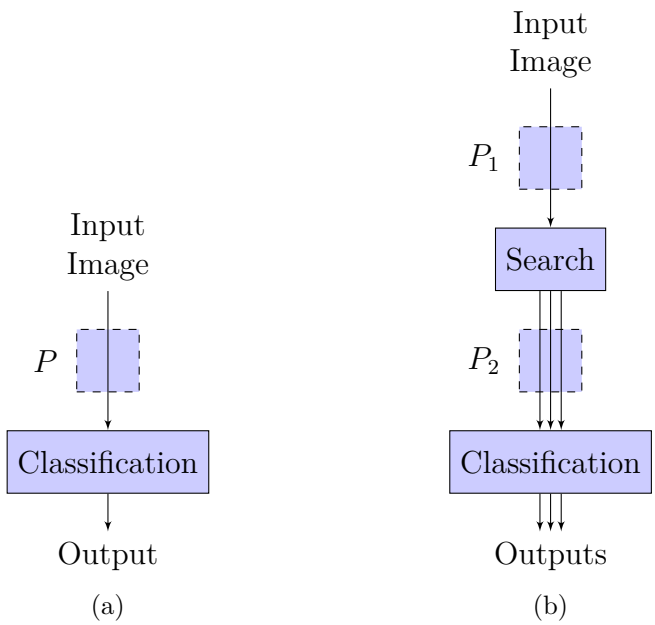


Figure 1: A diagrammatic example of an image processing system, that uses a search algorithm to simplify the task. In this example, P, P_1 and P_2 are potential locations for image preparation to the subsequent search or classification tasks.

accuracy and run-time complexity. Chapter 4 starts the novel contributions of this thesis with a novel framework in which different methods of constructing image preprocessors can be grouped and discussed. Chapters 5, 6 and 7 are dedicated to presenting three novel methods to design and/or construct image preprocessors. Chapter 8 will present an evaluation of the presented preprocessors in the aggregate (specifics having been covered in the conclusion section of their respective chapters) and finally Chapter 9 will end with a review of the contributions of this Thesis.

Chapter 2

Literature Review

This chapter is dedicated to presenting the core concepts required as part of the work presented in this thesis. This includes (in order of presentation),

- Artificial Neural Networks, discussing in detail Feed-Forward Perceptrons and Convolutional Neural Networks, due to their regular use in solutions to image classification problems,
- The traditional forms of image interpolation present in image preprocessors, their definitions and intended purpose,
- A collections of concepts used in the field of Image Processing to interpret or guide the development of models that are also applicable to the later works in this thesis, and
- A section demonstrating three previously developed image processors that function similar to those proposed in this thesis.

Finally, this chapter will conclude with a discussion on issues present in the literature that initiated this investigation and a summary of what to take forward from the literature into the research methodology.

2.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a branch of study interested in the simulation/digital replication of natural neural networks, (i.e. the cellular anatomy of brains) typically for the purposes of either understanding and modelling the functioning of the brain or to leverage to pattern recognition capabilities of neural networks systems for the construction of predictive systems. There are many variations presented in the literature, and here we will discuss a number of the models that show relevance to the work presented in this thesis.

Depending on how accurately the implementation intends to mimic the functionality of organic neural networks, they can be split into either Rate-Coding or Spiking models. In Rate-Coding models, the activity of a single computational unit is considered to be representative of the aggregate activity of a collection of equivalent organic neurons. Spiking models on the other hand, intend to replicate the functionality on the level of individual neurons, mimicking actual firing times and patterns of their biological counterparts. Due to their success in solving practical problems, Rate-Coding models have become the dominant method in industry, whilst Spiking models have remaind popular in academia due to their biological accuracy.

2.1.1 Feed-Forward Perceptrons

The most well-known for of ANNs is the Rate-Coding Feed-Forward Perceptron. A Perceptron is a neural unit the aggregates activity from its inputs by weighted summation and then normalises the result to compute its own activity, defined as:

$$o_j = \text{norm}(\sum_i w_{ij} o_i), \tag{1}$$

where o_x is the output of unit x, w_{xy} is the weight of the connection between units x and y, and $\text{norm}(x)$ is a normalisation function. In its earlier presentations,

a binary step function was used for normalisation, but due to the popularity of Back-Propagation, the Logistic function (sometimes simply known as the Sigmoid function due to its ubiquity) has become the de facto standard.

The term Feed-Forward refers to the way that the data flows through these network topologies. The neural units are structured into layers, the first layer is the Input Layer, containing simpler neural units that take the input data and convert it to a suitable format (or just an implementation of the identify function if it is already suitable) and can be used as buffer if the incoming data sources are not synchronised. The last layer, made up of the perceptron computational units described earlier, is referred to as the Output Layer, as the activity from these units is a representation of the output of the ANN as a whole. Between these are any number (≥ 0) of Hidden Layers using the additional perceptron units to compute intermediate interpretations of the input data. The size of the Input Layer is equal to the number of features the ANN is designed to work on, and the Output Layer is sized to match the intended encoding of the output; typically one output unit per class/label. The number of units per Hidden Layer and the quantity of the Hidden Layers is dependent on the task and varies between implementation and developer.

Back-propagation

Based on the earlier Delta-Rule, back-propagation is a learning algorithm commonly used on feed-forward neural networks. Its advantage over the earlier delta-rule is its ability to train ANNs that have one or more Hidden Layers.

After a prediction has been generated for a given input by the ANN, an error value, referred to as the units delta, δ_i, is computed. The delta of an output unit is the difference between the unit's activation, o_i and the intended result, t_i. For hidden layers, δ_i is the weighted sum of all connected subsequent deltas. In both cases, δ_j is scaled by the derivative of the normalising function (used during the

feed-forward phase) when applied to the units output, $\text{norm}'(o_j)$, used to prevent long term potentiation or depression of the weights. For each connection between units, w_{ij}, the weight is adjusted by the product of, the pre-synaptic activation, o_j, the post-synaptic delta, δ_i, and the systems learning rate, α; formally:

$$\Delta w_{ij} = \alpha \cdot \delta_i \cdot o_j \tag{2}$$

$$\delta_i = \text{norm}'(o_i) \cdot \begin{cases} o_i - t_i & \text{if unit i is an output unit} \\ \sum_k^{N_i} \delta_k w_{ik} & \text{if unit i is a hidden unit} \end{cases} \tag{3}$$

2.1.2 Convolutional Neural Networks and Receptive Fields

Convolutional Neural Networks (CNNs) are a variation of the feed-forward neural network model. In the CNN paradigm, neural units are arranged in volumetric layers, where each layer has *height*, *width* and *depth*. CNNs are constructed from a number of these volumetric layers, stacked one after another until the final layer is used to generate classifications or predictions. The most important of these layer types is the Convolution layer (for which the concept is named), based on the concepts of Receptive Fields and Convolution.

Receptive Fields are an instance of a partially connected network topology that can be implemented when the source layer has something analogous to a spatial distribution. In receptive field topologies, each neuronal unit in the secondary field is fully connected to only units in the input layer that are within some spatial bounding sub-volume of maximum *depth*. This implies that the latter unit also has a effective spatial location, based on to aggregate of the units it is connected to. Their typical construction is to have a grid of receptive fields of equal volume evenly distributed across the width and height of the input space so that each source is covered by at least one receptive field. See Fig. 2 for a digram of receptive fields.

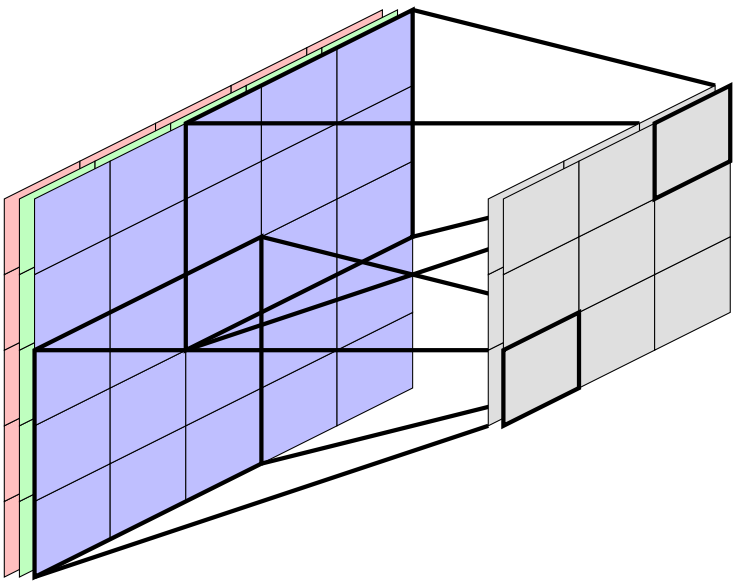


Figure 2: Receptive Fields. A diagrammatic representation of Receptive Fields, showing the down-sampling effect that they have on the previous layer. This example shows a $5 \times 5 \times 3$ input volume and the aggregating relationship to the output volume using a $3 \times 3 \times 2$ receptive field.

Each unit in the same $width \times height$ layer will be associated with a receptive field of the same dimensions, but occupies a different spatial location in the source layer. Each unit in the same column (*depth*) will have the same coverage (i.e. be associated with the same receptive field), but will apply a different function to that region.

Convolution is a process which aggregates a small sub-window of the underlying data, using a matrix of values, called the kernel. The size of the kernel is arbitrary along the *width* and *height* axis, and having the same *depth* as the source volume. The window is computed for each valid location for the given data resulting in stack of planar matrices, with each slice based on a different kernel.

In graphics application, another common usage of kernel convolution, every integral location in which the kernel fits is used as a valid location. If this were the only case, then both axis of the resultant image would be reduced by $k - 1$, where k is the size of the kernel along that axis. To mitigate this, the kernel size

in both axes are kept odd to provide a definite centre, and edge padding rules are applied to allow the kernel to approximate the locations at the edge of the source image.

For CNNs, the reduction in layer size is not considered a problem (it is actually a minor benefit as the complexity will always reduce in those two dimensions with each Convolution layer), so the necessity for the kernel to have odd sizes and edge data fabrication are discarded.

What makes these convolution layers 'neural' in functionality, rather than just kernel convolutions reminiscent of graphics processing, is that the kernels themselves are the neuronal units, subject to alteration by the learning algorithm. The values in the kernel are the weights of the neuron, totalling a single neuron per column depth, applied to multiple locations. In conjunction, the receptive fields reduce the complexity and maintain useful spatial relations, while the kernel reduces memory costs.

For completeness, the following is a list of the other commonly used layers in CNNs. The Input and Full Connected Layers are the volumetric equivalents to the their Feed-Forward counterparts, and the Pooling and ReLU layers are additional layers designs common place in CNN implementations.

Input Layer

The input layer (when dealing with images) treated as a volume of size $W \times H \times R$, where W and H are the width and height of the image or window, respectively; and R is the colour depth of the image.

Pooling

Pooling layers perform a simple down-sampling of the input volume, reducing *width* and *height* without affecting *depth*. The pooling layer divides the input volume into partitions of $X \times Y \times depth$, such that both $(width/X) \in \mathbb{N}^+$ and $(height/Y) \in \mathbb{N}^+$, applying the *max* function to aggregate; resulting in a volume of size $(width/X) \times (height/Y) \times depth$.

ReLU Layer

The Rectified Linear Unit (ReLU) is used to increase the non-linearity of the network, without effecting the dimensionality of the volume. It does this by applying the ramp function, $f(x) = max(0, x)$, to each location in the input volume.

Fully Connected Layer

The Fully Connected (FC) layer of a CNN is the analogue to a fully connected layer of traditional ANNs and are typically used as a terminal layer to the network. To remain consistent with the functionality of convolution layers, FC layers are represented as a volume, but due to the loss of any useful spatial information related to the source data passing through the neurons, the layer is represented as a single column, $1 \times 1 \times N$ where N is the number of neurons.

Topological Issues

An issue with CNN is the vastness of the potential design space. As mentioned previously there are numerous functionally different layers that can be used, each requiring different types and quantities of parameters for initialisation, from multiple domains of varying, potentially infinite, ranges. This makes the construction of functionally useful CNNs an arduous task for the developer who will be required to use trial and error approaches to resolve this construction.

A concern of the field (as opposed to a verified issue) is in its potential applicability to industry problems; translating a presented solution from the literature to a perceptually similar industry problem could require significant modification for even simple changes to the original problem[1].

[1]Significantly trivial alterations to the problem (such as source image lighting) can easily be solved with a re-train of the CNN.

2.1.3 Curse of Dimensionality

The *Curse Of Dimensionality* is a catch-all term used when discussing problems that arise when dealing with high-dimensional input data. The two most significant issues that fall within this moniker are Hughes Phenomenon (1968) and combinatorial complexity.

Hughes Phenomenon

The learning algorithms used by ANNs are designed on the assumption that identifying correlation between values in the source sample and the target class is sufficient to perform class prediction from unlabelled samples. This assumption is sufficient as long as the probability of coincidental correlation or partial-correlation are low in the training set; but runs into difficulty as the complexity (i.e. the dimensionality) of the sample increases (Hughes 1968). The chance of the training data including non-predictive data across the samples that, at least partially, correlate with the labelled classes increases with each additional non-predictive data point.

Combinatorics

The computational cost of a ANN is proportional to the number of connections in the network's topology. In a feed-forward topology, the number of connections between two layers is the product of the sizes of the two layers. The total connections in a feed-forward ANN can be described as

$$C = \sum_{l=2}^{L} u(l-1) \times u(l), \tag{4}$$

where $u(x)$ is the number of neural units in layer x, and L is the total number of layers. Two related situations can contribute to the increase the sizes of some of these layers, the dimensionality of the source data and the difficulty of the

task. The size of the of the input data is a direct contributor to the topological complexity of an ANN, for each additional dimension in the source data, the total number of connections is increased by the size of the subsequent layer. The size of the hidden layers can also be affected by the difficulty of the task; as the difficulty of the task increases, the need for additional intermediate hidden states to be modelled also increases, increases the connection count by the sum of the sizes of the prior and subsequent layer.

Mitigation

The two 'curses' discussed here can be mitigated by employing a number of different strategies. To combat Hughes Phenomenon, one can increase the size of training set; as the size of the training data increases, the probability of a data point producing a partial-correlation sufficient to appear as predictive to the learning algorithm decreases. This can present further issues as increasing the size of a training set can be difficult depending of the type of data.

To handle combinatorial complexity, numerous techniques to prune unnecessary dimensions from the source data have been developed. Known as Feature Selection (Chandrashekar and Sahin 2014), these class of processes are employed in an attempt to pre-emptively identify input data of low correlation (data irrelevant to the given task) and filter it out from the data set presented to the ANN. When dealing with image data, interpolation algorithms are frequently used as a method to reduce the input dimensionality of the image as they approximate a similar image for the task of lower dimension. In the cases of pruning the input data, the additional computational cost is typically less than the saved computational cost of processing the full data set; the task may have become learn-able by the particular implementation; and (assuming it was previously learn-able or other methods to achieve that were viable) the reduction in predictive accuracy is negligible.

If the data can be modelled spatially, and it is reasonable to assume that the distance between data-points is inversely-proportional to task-centric relationships, then receptive fields are a viable choice for pruning otherwise fully connected layers. This process is one of the major improvements provided by CNN models when dealing with image data.

2.2 Traditional Image Interpolation

To find an ANN system using a preprocessors based on Bilinear or Bicubic interpolation for image reducion is not an uncommon thing, though in recent years this has been replaced by CNNs or frequency transformations(which will be discussed briefly in section 2.3). The design of these two methods was originally to up-scale images for graphics applications, with the intent that the resultant image appear to be a sufficient representation of the original to human perception (in terms of aliasing and artefacts introduction). Other possible methods, Sinc/Lanczos3, Spline (Subbotin 2001), also exist which produce a subjectively better resultant image for human vision at the expense of increased memory and/or processing time.

When used in an ANN preprocessor, the typical reason is to reduce the complexity of the image; this can either be to combat the curse of dimensionality (Sec. 2.1.3), or to reduce the processing time required to propagate the image data through the ANN (Sec. 2.1.3).

2.2.1 Bilinear Interpolation

The Bilinear Interpolation algorithm (Press et al. 1992) is designed to produce smoothed transitions when rescaling an image. Its primary purpose is for up-sampling an image, in which the total number of pixels is increased along both dimensions of the image. The algorithm is based on linear interpolations, or

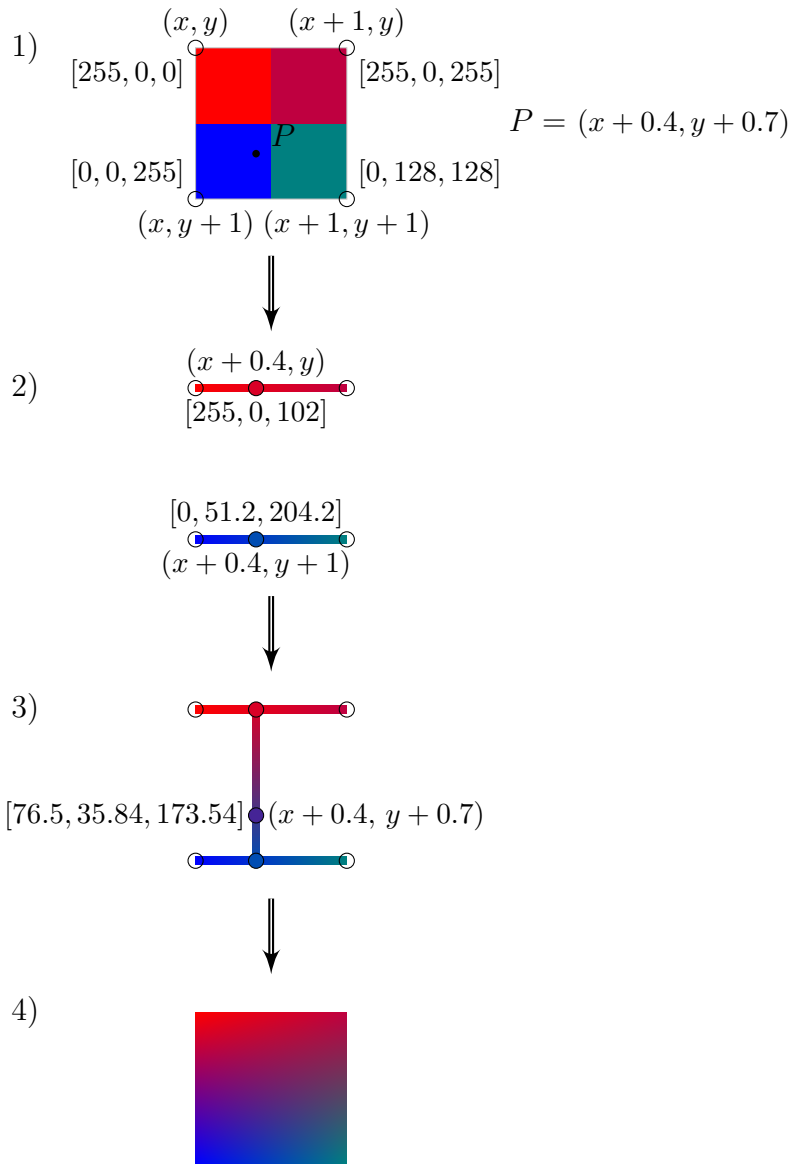


Figure 3: Bilinear Interpolation. In the above figure the notation (x, y) describes a 2 dimensional position, whilst the notation $[R, G, B]$ is used to describe the colour with red, green and blue components, each in the range 0-255, inclusive. Any sub-pixel point within an image can be assigned a colour with only 3 linear interpolations. 1) The original 2×2 pixels and the target point. 2) The first 2 linear interpolations moving right along the x-axis by 40% of a pixel. 3) The third interpolation, using the colour determined from the first two and moves down along the y-axis by 70%. 4) The result of performing a linear interpolation of all sub-pixel points of the original 4 pixels.

Lerps, along the two spatial dimensions. For the remained of this section, each pixel in an image be considered as a point with 2 spatial dimensions, x and $, y$ and three colour dimensions, r, g and b that correspond the red, green and blue colour components of a colour.

In linear interpolation, two N-dimensional points, A and B, and a fraction of traversal, R, are used to interpolate a new point, P, that is the fraction R along the line defined by A and B,

$$P = A + R(B - A) = B + (1 - R)(A - B) \tag{5}$$

To extend lerping to additional dimensions, for each additional dimension, the points A and B are computed via the same method along the additional dimension. In a 2D plane, this requires a total of 3 lerps. The two extra lerps to compute the values A and B, and the final lerp in the final axis. In a 3D volume, this adds an additional 4 lerps, following the pattern

$$L(n) = 2^{(n-1)} \tag{6}$$

$$\begin{aligned} L_{total} &= \sum_{n=1}^{d} L(n) \\ &= 2^d - 1 \end{aligned} \tag{7}$$

where d is the number of spatial dimensions, $L(x)$ is the number of additional lerps required to extend to x^{th} dimension, and L_{total} is the total number of linear interpolations required to compute the target value. This is demonstrated in Figure 3.

The Bilinear algorithm can also be used for down-sampling; a down-sampling to less than 50% along either axis can result in a poorer quality than would be expected. This is the result of a sampling issue, as reductions of greater than 50% result in periodic skipping of particular pixels. To mitigate this, a number

of implementations perform the following multi-step operation: if an axis is to be scaled by a factor less then 0.5, then that dimension is additionally scaled by a factor of 0.5 and the scale factor doubled. Whilst the scale factor remains less than 0.5, this process repeats. As each the length of each axis must be a positive whole number after each step, consideration of rounding errors must also be taken into account. This usually takes the form of slight adjustment to the scale factor after each step and before the recursion check. The blog post "The Perils of Image.getScaledInstance() Blog" (Campbell 2007), provides an apt description of this particular problems for Bilinear interpolations and discusses the concept of the multi-step interpolation in the context of the Java programming language.

2.2.2 Bicubic Interpolation

Catmull-Rom Interpolation (Catmull and Rom 1974), also know as Bicubic Interpolation, is commonly used in graphics processing when time is not a critical factor. A typical re-sampling with bilinear interpolation is based on 4 sample pixels, where bicubic is interpolated from a source of 16 samples, making for a noticeable increase in computational complexity. This additional cost is usually considered acceptable due to the smoother result from this interpolation, as the derivative of the resulting curves are continuous, unlike Bilinear, which has constant derivative between samples an discontinuous across sample boundaries.

To compute the Bicubic Interpolation of an image, the kernel

$$W(x) = \begin{cases} 1 & x = 0 \\ 0 & x \in \mathbb{Z} \\ 1.5|x|^3 - 2.5x^2 + 1 & |x| < 1 \\ |x|^3 + 2.5x^2 - 4x + 2 & 1 < |x| < 2 \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

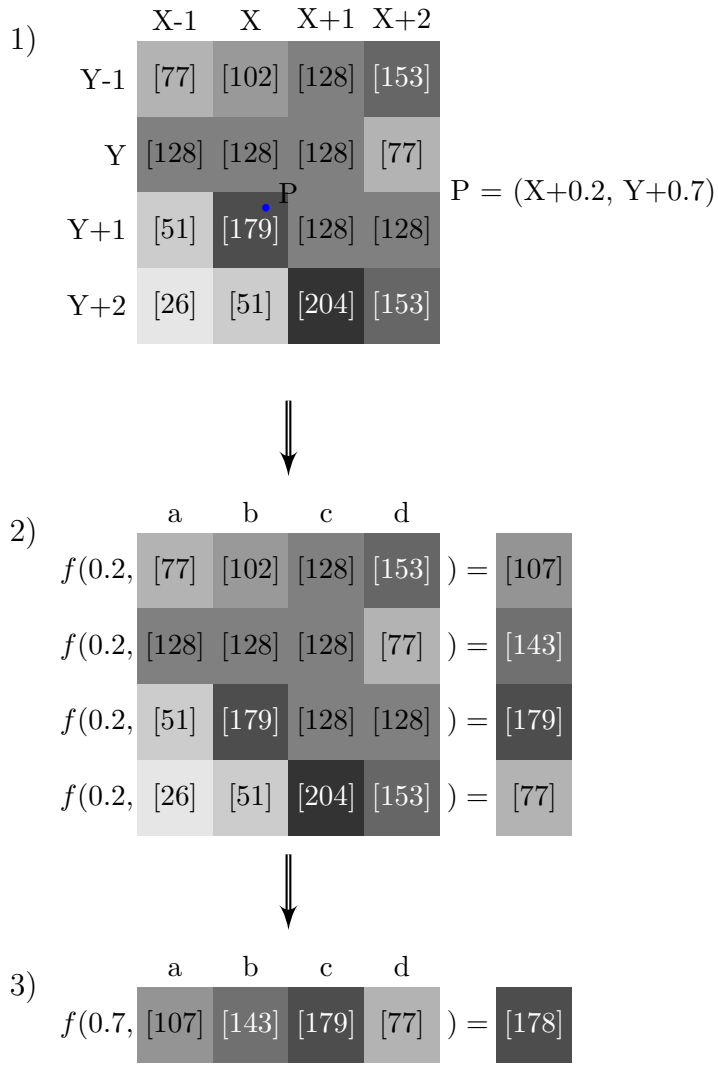


Figure 4: Bicubic Interpolation. In the above figure the notation $[G]$ is used to describe the grey-scale value in the range 0-255 (inclusive). f is the single dimension cubic function (Eq. 9). 1) The target point to be interpolated, P, and the surrounding 4×4 samples. 2) Applying the cubic function to each row, using $t = 0.2$ from P's x-offset. 3) Appying the cubic function to the vertical results, using $t = 0.7$ from P's y-offset. P is computed a value of 178 ($\approx$ 70% black); if bilinear was used, a value of 157 ($\approx$ 61% black) would have been computed.

is applied, by convolution, to each dimension of the image in an identical manner as discussed in the bilinear interpolation explanation. Applying the kernel to the single dimension convolution, this can be simplified to

$$f(t, a, b, c, d) = \frac{a(-t^3 + 2t^2 - t) + b(3t^3 - 5t^2 + 2) + c(-3t^3 + 4t^2 + t) + d(t^3 - t^2)}{2}, \tag{9}$$

where a, b, c and d are a sequence of sample points and t is the interpolation distance from point b ($t = 0$) to c ($t = 1$). Figure 4 demonstrates the application of this function.

The optimised version will typically apply the convolution to each dimension independently, storing the result for the first dimension in an image-like data structure so that these values do not need to be recomputed during the subsequent pass for the second dimension.

As the derivative of cubic interpolation produces a continuous curve, significant changes in value of the source material can result in overshooting in the solution, usually resulting in clipping but with the potential for under/overflow in a poorly implemented interpolation. Visually, this appears as a haloing artefact along edges of high contrast, and in some situations can be beneficial for human perception as it increases the apparent sharpness along that edge. If not accounted for, this can cause issues in computational systems that rely on or attempt to identify edges of objects in images.

2.2.3 Other Interpolators

This section contains a brief overview of some of the less common interpolators available. This will include a description of how the algorithms work and their pros and cons.

Nearest Neighbour

The simplest resizing process, the colour of a pixel returned by the Nearest Neighbour algorithm will be the nearest pixel to the pre-scaled equivalent location. This can be performed with trivial integer mathematics. This makes the process very fast, but produces very blocky results when up-sampling and will miss out portions of the image when down-sampling. Up-sampling with nearest neighbour can be beneficial when carrying out visual examination of the pixels within an image, as this method does not change or generate any new colourations from the source image.

Sinc/Lanczos

The Lanczos re-sampling algorithm is used as a popular, almost de-facto, standard in graphics re-sampling and other interpolation tasks due to its ubiquity and quality (Turkowski 1990). This algorithm uses a windowed version the normalised Sinc function, bounded to the region $[-a \ldots +a]$, where a is an integer, typically 2 or 3. The Lanczos re-sampling algorithm has a number of issues similar to those of Bicubic; As the kernel contains negative lobes, it is possible of large changes in neighbouring values to produce a haloing effect. Additionally, the calculation at each point requires $(2a+1)^2$ pixels, significantly increasing computation time over the Bicubic and Bilinear algorithms.

Area Averaging

Sometimes known as Weighted Average, each pixel in the returned image has the (weighted) average colour of each source pixel that has been amalgamated in the re-sampling. This algorithm was designed to produce smooth images but with fewer artefacts than Bilinear and Bicubic. In the case of up-sampling, this algorithm will degrade to Nearest-Neighbour at an increased computational cost. If this algorithm is used to down-sample by factors of 2, the result coincides with

those produced by Bilinear Interpolation. The algorithm is rarely used in practice despite the smoother results as similar results can be achieved by super-sampling (up-sampling to a resolution of $2^N \cdot R$ and then down-sampling to the target resolution of R) with the considerably cheaper Bilinear and Bicubic implementations.

Basis Spline

Most image interpolation methods can be re-interpreted as optimised implementations of B(asis)-Splines using specific kernels. These re-interpretations allow for the identifying of the implied kernel, which can be used to generalise the task of image interpolation to an implementation of a B-Spline algorithm and then defining the kernels to produce the desired image interpolation algorithm. This is beneficial in some cases as this allows for mathematical modelling of the algorithm and simplification of the algorithm due to the removal of any branching. The cost of this generalisation is in the computational cost of B-Spline implementations which generally makes these impractical for production systems where runtime is an important factor.

Box Spline is one such re-interpretation; The Box kernel will recreate the effect of Nearest Neighbour when re-sampling the image.

Similar to Box Spline, Triangle Spline is a re-interpretation of Bilinear Interpolation as a specific kernel for a general purpose B-Spline algorithm.

2.2.4 Comparisons

This section will provide a brief comparison for the most common graphical image interpolations as a way of presenting the 'feel' of each presented. Figure 5 shows a visual comparison of the quality of the algorithms and Tables 1 provide a runtime comparison. In both cases, the 512×512 Lena image is used as the baseline image and all down-sampling was done by a factor a three. The Lena image is the de-facto standard used to visually demonstrate compression, of which down-sampling

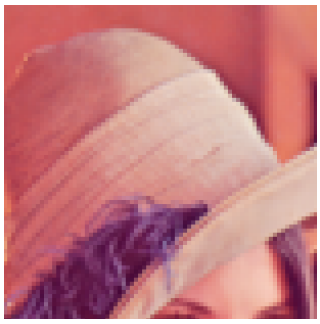
is the most basic lossy form. This factor was chosen as some implementations are known to take advantage of 2^n scaling factors and this may bias the timing results.

The execution times presented in Fig. 5 cover a variety of image interpolation techniques and a variety of implementations. The primary focus for these comparison was between the different techniques discussed above, with a secondary interest in the variance between implementations. The table does not include the C/C++ languages as their landscape is extremely diverse and there is little consensus on 'standard' libraries for image interpolation. Most of the times on this table present an ordering that is to be expected; Nearest Neighbour < Bilinear < Bicubic < Lanczos and Box < Triangle. Unexpectedly, MatLab's Triangle implementation was faster than the Bilinear implementation, which may be due to the design focus of MatLab towards mathematical accuracy instead of computational optimisations. The Java Image Scaling Library (JISL) was notably slower than the two native Java implementations; the interpolation algorithms in JISL are all executed directly by the Java Virtual Machine on the CPU, compared to the native implementations that has the ability to off-loaded the process to the GPU.

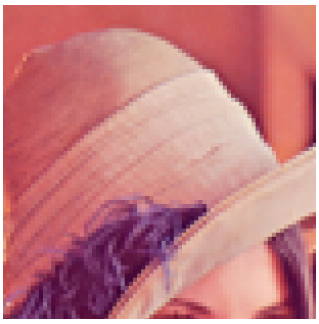
To provide a more detailed exploration of some of the interpolation techniques discussed a brief test was performed. The five interpolation methods 'Nearest Neighbour', 'Bilinear', 'Bicubic', 'Lanczos3' and 'Area Average' were tasked with downsampling the CMU-130 Dataset (Section 3.3.3). The images in the CMU-130 dataset were gathered from many sources and are highly variable in size and content. The first downsampling factor of $^1/_2$ was chosen as many of these implementations are able to take advantage of 2^n factors, giving a best-case scenario. As not all of these implementations can take advantage of this optimisation, the second factor of $^1/_3$ was selected, forcing these implementations to uses program paths of higher cost. Each pass was repeated 10 times and the results averaged. The outcome of this test is presented in Table 2. The results here show the same tendency in computational as that found above. It is interesting to note that Area

(a) Lena (Original)

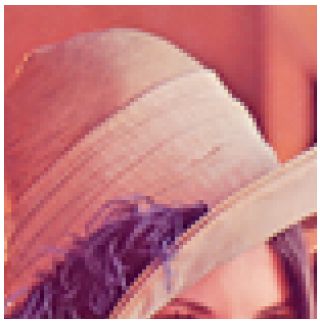
(b) Nearest Neighbour



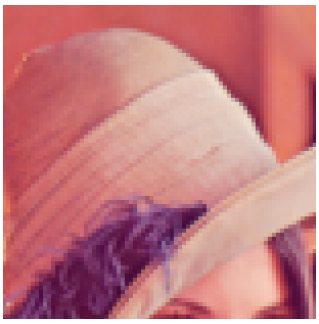
(c) Bilinear



(d) Bicubic



(e) Lanczos



(f) Triangle

Figure 5: The original Lena image and a selection cropped interpolated images. Each image is the result of down-sampling the Lena image by the specified method and then up-sampling by Nearest Neighbour for inspection. The blocky transition at the edge of the hat is due to the use of Nearest Neighbour and thus makes for un-fair comparisons cross such hard edges in this situation. Comparisons can be made about the textured surfaces and smoother edge transitions. Image (a) shows the roughest interpolation, with moiré patterns resulting from shadows on the woven surface. This roughness is not seen in the other images, but almost all visual indication of the hats texture have also been lost. Looking at the creases in the fabric and at the feather, images (d) and (e) show clearer edges than images (b) and (f) due to the overshoot present in their respective interpolation methods.

Interpolation Method	**Nearest Neighbour**	**Bilinear**	**Bicubic**	**Lanczos (a=3)**	**Area Averaging**	**Box**	**Triangle**
getScaledInstance() (Java)	159ms (*)				5956ms (+3646%)		
Graphics2D (Java)	31ms (-93%)	429ms (*)	1576ms (+267%)				
Java Image Scaling Library				12630ms (+81%)	9421ms (+35%)	5533ms (-21%)	6988ms (*)
Python Image Library (Pillow)	27ms (-98%)	1249ms (*)	2240ms (+79%)	3111ms (+149%)		789ms (-37%)	
MatLab R2016a	565ms (-60%)	1428ms (*)	1942ms (+36%)	2553ms (+79%)		1108ms (-22%)	1288ms (-10%)

Table 1: Computation times to down-sample the 'Lena' image by a factor of 3 using each method. Percentage times are present as quick comparisons to the baseline (*) for each family.

Table 2: Average execution time of various interpolation techniques against the CMU-130 data set.

Factor	Interpolator	Average Execution Time
$^1/_2$	Nearest Neighbour	8.04ms
	Bilinear	68.12ms
	Bicubic	231.99ms
	Area Average	219.37ms
	Lanczos3	596.79ms
$^1/_3$	Nearest Neighbour	5.72ms
	Bilinear	32.53ms
	Bicubic	105.11ms
	Area Average	207.90ms
	Lanczos3	516.92ms

Average was faster than Bicubic when tasked to downsample by a factor of $^1/_2$, but slower with $^1/_3$.

This section has presented a number of the common re-sampling algorithms used in practise. In the later chapters, some of these will be used as controls to compare the novel preprocessors presented later. For these controls, Bilinear and Bicubic interpolation methods were selected for a number of reasons. Both have reduced computational time whilst producing smooth results, ubiquity across platforms and languages, minimal production of artefacts, and known use in past literature. Bilinear was specifically selected as it was the fastest and simplest of the discussed smoothing interpolators. Bicubic was selected as a second control as it is popular with a notable portion of the image processing community due to its basis on the continuous differential kernel. This is not a requirement of the work in this thesis, but may be of benefit to future researchers wishing to comparing this work to others.

2.3 Frequency Transformations

Feature extraction from an image can also be performed using frequency transformations. Discrete Fourier-Related transformations, such as Discrete Cosine

Transformations (DCTs), are used to extract frequency information from the input sequence. Discrete Wavelet Transformations (DWTs) can also be used to extract frequency information, and also provide positional information of the frequency along the sequence. Both forms are common in lossy image compression techniques as, generally, the high frequency information can be discarded at minimal cost to reproducible quality of the represented image.

When applied to a 2D image for feature extraction (Qazi and Wong 2016; Ooi et al. 2016; Lukas et al. 2016), these techniques can be used to convert an input image to a frequency description, allowing for selection/identification of key frequencies, usage of positional information where available or as an alternate presentation to further processing systems. Along with CNNs, these techniques have seen an increase in popularity in the literature recently due to the improved of computational availability (cost per computation) despite there increased complexity and runtime when compared to the traditional methods discussed above.

2.4 Related concepts

This section discusses a number of concepts that relate to the models presented later in this thesis. The intermediate stages of the presented models could be perceived as a statically encoded variation of visual attention. The methods by which these are produced uses ideas that correspond to the comparable to those present in Saliency Map. The resultant image preprocessor can be reinterpreted in terms of Foveation, Superpixels and/or Receptive Fields. As such, a brief overview of these paradigms are presented here.

2.4.1 Visual Attention

The novel concepts presented in later chapters of this thesis bear a conceptual resemblance to *Visual Attention* (VA). Visual attention is a field of study that

bridges the fields of Cognitive Neurobiology (Crick and Koch 1990; Buia and Tiesinga 2006) and Computational Intelligence (Itti, Koch and Niebur 1998). For Cognitive Neurobiology, it is an attempt to model the behaviours of the brain in its behaviour of providing cognitive processing to a reduced portion of perception, specifically focusing on visual perception, used by the brain to focus awareness and reduce cognitive load. In the field of Computational Intelligence, it investigates the problem of identifying some part of a visual field as more salient, relevant or important, thus requiring attention (in the form of processing or selection) of the system. A subtle difference between the models of this thesis and typical VA methods is that VA will search the visual field and identify sub-regions as requiring attention for later processing stages, where as the models presented in this thesis generate a model that applies a defined attention to the visual field for later processing. A anthropomorphism of the two concepts is that VA is the attentive choices made by the brain (conscious or unconscious) and the models presented in this thesis correlate to the fixed attention provided by the fovea and peripheral vision of the eye.

Saliency Maps

Saliency Maps are a process by which a visual field is explored in a bottom-up approach for objects or regions that, are distinct from the background, or may be of value for the task of the subsequent systems. The model by Itti, Koch and Niebur (1998) is a well known model that based on this idea.

Itti's model is inspired by the functioning of primate brains due to "a remarkable ability to interpret complex scenes in real time, despite the limited speed of the neuronal hardware available for such tasks." The model takes the presented image and generates a Gaussian pyramid with scales $2^{-n}, n \in [0..8]$. The pyramid is then along 13 different feature pathways corresponding to Intensity, Colors (Red, Green, Yellow and Blue) and Orientations (in 45° increments). For each

pathway, the pyramid has the Center-surround differences computed to produce the pathways Feature map, then an Across-scale combination and normalisation is applied to reduce the pyramid to a single Conspicuity map for each pathway. The Conspicuity maps are combined via Linear combinations to produce a single Saliency map and a winner-take-all selection is applied to identify where the attention is to be applied. If multiple attentive locations are desired per input image, an blanking circle is applied to the previously selected region of attention (reducing saliency maps values around that location to zero) and the modified saliency map is reused in the winner-take-all process. Figure 6 displays the data flow diagram of the model, as presented in their original paper.

2.4.2 Simulated Foveation

The Fovea centralis (Fig. 7) is a small indentation in the retina at the back of the eye (Fig. 7). This 'pit' is connected to over half of the optic nerve and is responsible for the focal in human visual perception. Foveation is the term used to describe the act of applying this effect to imagery. The technique, is used by artists and designers to encourage focus on particular targets within an image or video sequence and has been generalised to Depth-of-field techniques where shapes beyond points can be used as foveation targets. It has also been used as a method of image or video compression to lower the required data for parts of the scene inversely proportional to the distance to the focal target(s), (Itti 2004; Wang, Lu and Bovik 2003; Lee, Pattichis and Bovik 2001; Geisler and Kortum 2001) and is still an active area of research (Li, Liu and Chen 2016; Cheng et al. 2015) for H.265 and similar streaming video compression algorithms.

This concepts can also be applied to AI related image processing tasks, though has since fallen out of favour, possible due to the surge in popularity of CNNs. The works by Scassellati (1998, 1999); Shipley and Kellman (2001); Terzopoulos, Rabie and Grzeszczuk (1997) all show examples of various forms of foveation used

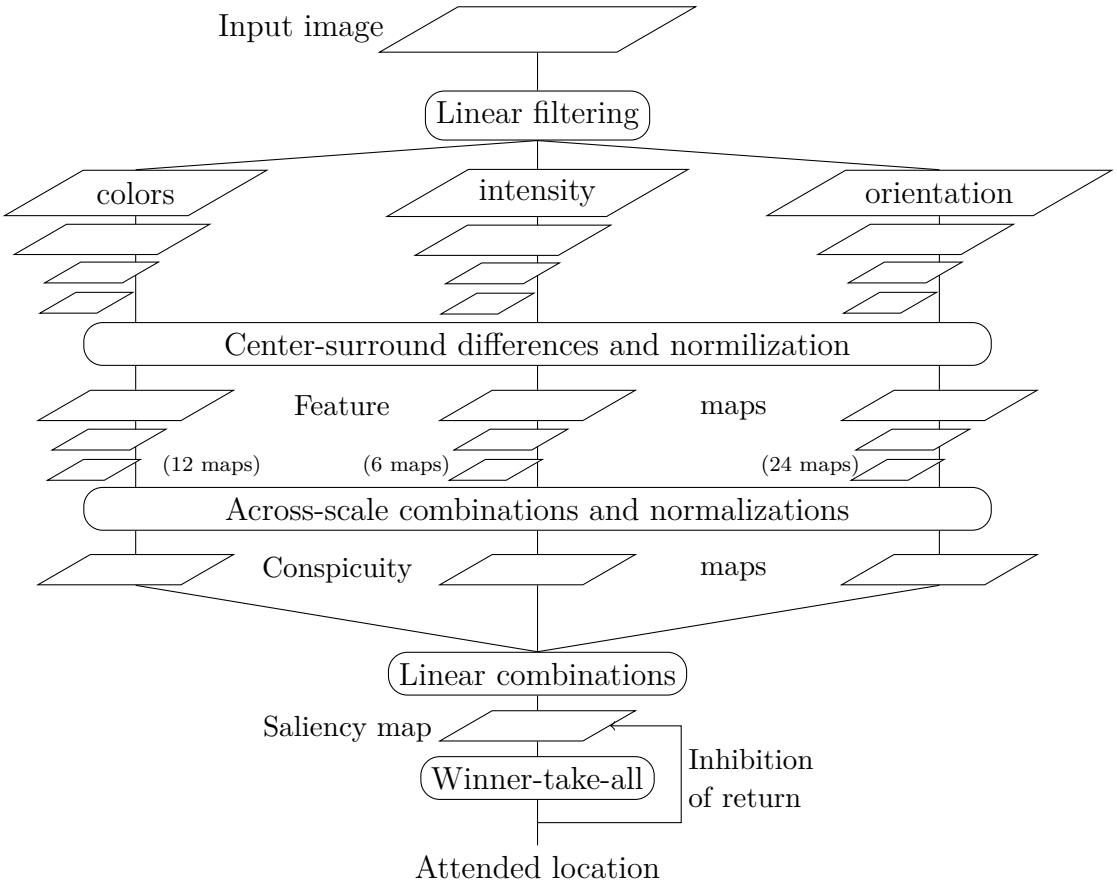


Figure 6: The General architecture of Itti, Koch and Niebur's model; as depicted in their original paper.

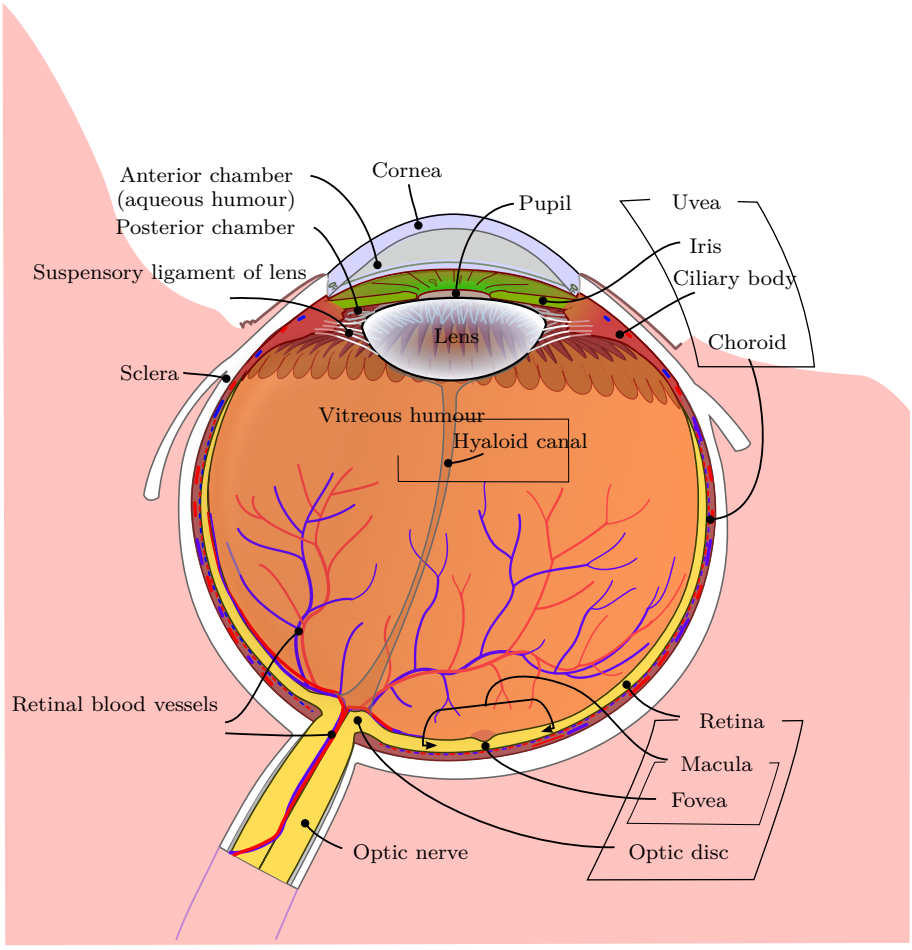


Figure 7: Diagram of the human eye (March i Nogué 2016). It shows the lower part of the right eye after a central and horizontal section. The fovea centralis can be located towards the bottom of the image (labelled Fovea).

to compress the source data as part of data preparation or as a way to target 'perception' from the later processing systems.

2.4.3 Superpixels

Superpixel is a term used to describe a segment of an image which only contains spatially continuous pixels that share similar colour properties. In the article Achanta et al. (2012), the authors present a comparison between a number of Superpixel algorithms and their novel design, Simple Linear Iterative Clustering, SLIC.

In the SLIC model, the generation of superpixel segmentation is performed in four steps, Setup, Assignment, Update and Post-Processing. In the Setup step, the superpixels centres are pre-seeded at regular intervals of $S = \sqrt{N/k}$ where N is the number of pixels in the image and k the number of desired superpixels. The label assigning each pixel to a superpixel is initialised invalid (i.e. -1) and the distance between the pixel and superpixel centre is set to $+\infty$. During the Assignment step, the distance between the superpixels centre and each pixel in a $2S$ square region, centred on the superpixel, and assigns them to the superpixel if the distance is less than the distance to their previous containing superpixel. The Update recomputes the cluster centres and the residual error (from the L_2 norm). The Assignment and Update steps are iteratively repeated until the reported residual error is below some threshold[2]. The final Post-Processing step reassigns disjoint pixels to spatially nearby superpixels to enforce connectivity.

AN issue with Superpixel technologies is in the knowledge required by the developer. Most Superpixel algorithms require the developer to specify a target number of Superpixels to segment to; along with the issue of ordering the resultant Superpixels. Subtle changes in an image can alter the colour distribution enough the signifcantly alter portions of the image segmentation. With this instability

[2]The authors report that 10 iterations a sufficient for most images.

and the variability in locality of the resultant Superpixels, it can be hard to assign where each segmentation feature will be assigned when applying these algorithms to methodologies like ANNs.

2.5 Related Implementations

These following three models are presented as examples of work closely related to that discussed in this thesis. All three provide unique solutions to the issues of data preparation for image based systems.

2.5.1 MIPs based Preprocessor

Lew and Huijsmans (1996) present a face detection model whose preprocessor does not use traditional down-sampling interpolation techniques. For their design, they take the face and non-face training sets and calculate then Kullback relative information of each pixel of the data set to determine the amount of information each pixel provides to the face detection task. Applying Gibbsian Random Fields allows them to select the 256 "the most informative pixels" (MIPs) from this set with regards to the information provided by the other pixels in the subset, as to make sure large groups of pixels provided a good interpretation of the same piece of information to the task. This allows for a large decrease in the computational cost of both the preprocessor and backing system: The preprocessor is a filter, taking in the image from the sliding window and filtering out only the pixels previously identified as one of the 256 MIPs.

2.5.2 A Superpixelwise CNN

He et al. propose a novel method to detect the salient object from a scene (2015). Their method, SuperCNN, uses superpixels to divide up the scene into arbitrarily shaped regions (Superpixels) and a process to standardise the information to

present to the CNN. The model includes extracting the colour uniqueness information from the superpixels and an additional metric of colour distribution to restore the spatial information required by the CNN.

2.5.3 ANNs and Improved Binary Gravitational Search Algorithm

In the paper by Pourghahestani and Rashedi (2015), the authors provide a clear description of a system that performs an object detection task using an ANN and feature selection. The preprocessor used in the model is divided into 3 steps. First, the image is processes to compensate for illumination variations due to environmental lighting conditions; then the watershed algorithm Meyer (1991) is applied to segment the image into objects within the frame; and lastly, feature extraction is applied to the segmented objects to determine which features are to be passed to the ANN.

The paper also describes the iterative process used to determine which of the features to be extracted for classification. This processes uses KNN classification and Improved Binary Gravitational Search Algorithm (Rashedi and Nezamabadi-pour 2014) to iteratively identify potential candidates for classification from a set of training objects. This process is a clear description of what would be considered as the Construction Phase of a Bespoke Preprocessor.

2.6 Issues in the Literature

When building a high-throughput or real-time image processing system that uses an ANN, it is common place to down-sample the input image as part of the preprocessing or data preparation stage. As discussed in Sec. 2.1.3, the typical reasons for these choices are either that it is computationally cheaper to resize

the image than it is to have the larger ANN input layer (and the resulting Input-to-Hidden synapses), or that there is an inability to learn/solve the task due to relevant signals getting lost in the noise of the full image; as seen in the following extracts:

> "The size of the captured image is 640×480, but due to the limited computational power, it is used after resizing to 26×20 using the OpenCV library." (Shibata and Utsunomiya 2011, p. 1447)

> "...the DVS128 retina has 16,384 elements, ... [so] a relevant method of reducing the data flow must therefore be employed, without losing information relevant to the line-following task."
> (Davies et al. 2010, p. 8)

> "...only a coarse elevation map is necessary for autonomous maneuvers. This requirement allows us to lower the resolution of the reconstruction, which drastically reduces the processing time for one depth map. In practice, we initialize one depth filter for every 8×8 pixel block in the reference frame. We therefore obtain dense depth maps of size 94×60, totalling 5820 depth filters for every reference image."
> (Forster et al. 2015, p. 114)

> "...every two successive images, we densely sample pixels (every 20 pixels in our case) on the image ..."
> (Garg, Yang and Scherer 2018, p. 3)

In these examples, a considerable amount of information is lost in the process. In the case of the first two quotes provided, the tasks were performed using sources of high-contrast images in a search for simpler geometric patterns that is mostly resilient to even these extreme image reductions. The latter two quotes are in

reference to algorithms specifically constructed to work with the coarser data provided by the downsample. In other situations where more complex tasks are to be performed, it is hard to argue that using image sources in which most of the data is considered to carry little predictive value or is highly redundant, and can therefore be discarded with little effect to the ANN's functionality. Such losses or inefficiencies may reduce the effectiveness of a system or outright prevent it from working at a given image resolution. The challenge occurs when the inverse is also true, as discussed in Section 2.1.3, where the excess of information, useful or not, can hamper an ANNs ability to learn. This can be exaggerated when the developer has little (or no) prior knowledge of what areas in the source images will prove useful to the task as the developer will have to spend significant time in attempts to gather this knowledge, or brute force without it. Typically practice is to search multiple resolutions to find a few that provide some early, sufficient solution to the problem, then progressing to finding sufficiently accurate variations.

A common problem in the literature is a vagueness in how the author(s) came to using particular resolutions (with regards to traditional down-sampling techniques) or topologies (when discussing CNNs). Additionally it is not unusual for a lack of detail pertaining to the implementation specifics of any down-sampling performed to make source imagery compatible with the presented model.

As examples of these issues in regards to traditional down-sample, the two papers quoted in the previous section (Shibata and Utsunomiya 2011; Davies et al. 2010) are used as examples. Shibata and Utsunomiya provides a preprocessor output resolution of 26×20 and Davies et al. settle on 16×16, but neither provide details of the algorithm used to perform these reductions or how they came to the choice of the stated resolutions. It can be assumed, in cases like these, that the likely algorithm was Bilinear Interpolation due to is speed, ubiquity and low memory footprint compared to the other methods available; and in the case where Bilinear was not the algorithm used, if Bicubic had been used, Bilinear can be

used as a sufficient substitute due to the nature of the task[3].

As described earlier, the differences in the various interpolation techniques mostly present themselves in the form of differing responses to edges in the source image. In the case of down-sampling, the reduction in data mostly cancels out these variations, but not completely, and sometimes even these subtle changes can affect the quality of the resulting ANN. The testing domains used in this thesis will be using the two of these interpolation methods as controls for this reason. This effect will be seen in these as slight variations in predictive accuracy and other similar measures. The analyses performed will additionally show that this difference is significant in occasion.

There is a similar issue present in the CNN literature for the Convolution and ReLU layers Zbontar and LeCun (2016); Lang, Zhao and Jia (2016); Covell et al. (2017). Both types of layers will reduce the width and height of the previous volume and the same question applies to each. Discussed models will include a detailed description of the layers used and the internal structures of each layer; but a lack of detail on how a choice was made on the reductions present in the Convolution and ReLU layers.

The lack of answers to these questions may not constitute throwing out the works discussed, by their inclusion in the literature would allow for the gauging the ease of implementation and how well the model could diversify to other problems. A model whose parameters were able to be generated by a automated system, or works sufficiently with a very broad range of parameters, would be significantly easier than a system that required fine tuning by hand to a high degree of precision; or a model where the parameters are highly dependent on one another or the results unstable with even slight alterations to initial conditions.

[3]Bilinear can be used as a rough approximation for Bicubic in most circumstances at an implementation level. Bicubic is useful at a conceptual level as it is a continuous function, i.e. differentiable at every location.

> "To detect faces larger than the window size, the input image is repeatedly reduced in size (by subsampling) ... by a factor of 1.2 for each step in the [image] pyramid."
>
> (Rowley, Baluja and Kanade 1998, p. 8)

Again looking at traditional down-sampling methods, the authors of this third quote have a similar task as the authors of the earlier quotations. Rowley, Baluja and Kanade implement an image pyramid in which the source image decreases in size with each step of the pyramid. The paper specifies that the image is subsampled up by a factor of 1.2 for each step, but does not specify if this is processed from the previous step (compounding edge variations and rounding-errors) or if it recomputed from the original image by a factor of $1.2^{(\text{steps})}$.

2.7 Summary

In the case of all prior works, Section 2.6, there is a clear pattern of taking in a source image that is too large to be directly processed by the primary classifier; passing through an additional preparatory process that reduces the complexity of the image in a variety of ways until it is able to be parsed by said classification system.

There is little work, if any, that looks at the preprocessor as a separate entity from the primary classifier, but instead there are many works that will take the classifier from one model and combine it with their own preprocessor specifically crafted for their novel system.

An aim of this thesis is to provided a baseline with which to describe and define the many preprocessors displayed throughout the literature. With the effect of delineating the preprocessor from the classifier, it becomes apparent that the preprocessors provide data in not just a particular format, but with also with specific properties. Looking back to SuperCNN (Sec. 2.5.2), the preprocessor described

takes a simpler algorithm that provides a significantly reduced output than the original image and extends it provide the location information required for a CNN to function. Repeating this process with other systems will provide a catalogue of algorithms, what data they provide and properties the data has. For the case of single algorithm preprocessors this is already the case; but with multi-algorithm preprocessors it may be non-obvious which properties have been maintained or what new properties have been gained due to the particular sequence of functions. This will be further discussed later in Chapter 4.

Chapter 3

Research Methodology

This chapter will present two testing domains that will be used in later chapters to evaluate the described preprocessors. The first of these Test Domains will be used to briefly identify if the preprocessor warrants a full analysis, where as the second will provide a detailed investigation and relating analysis. Before describing their details, this chapter will first cover the components that are common to both domains: Hardware and Platform specifics, Set-up of the testing environment and Control Groups. This will be followed up by the specifics of the two test domains used in the later chapters: Object Presence and Face Detection.

3.1 Domain Commonalities

This section covers a few aspects of the two test domains that are consistent between them. This section will first cover the execution environment in which the test domains are performed and then follow up with a description of the control groups used in the two testing domains.

3.1.1 Hardware and Background Processes

Some tests performed in response to the questions presented in this thesis require the comparison of algorithm run-times. When performing tests of this nature, extra precautions are required to make sure that the extra activities of the testing environment, and computer as a whole, have near-equal and near-minimal effect on the timings of the algorithm(s) being tested. The strategies implemented to achieve this can be divided into two groups, those that affect the operating system and those that affect the Java Virtual Machine (JVM).

Physical machine and Operating System

Various strategies where applied to counter the effects of additional processes on-going on the test machine and to minimise I/O delays affecting any of the tests.

To minimise the effect of external processes, and maximise the throughput of the testing application, multiple strategies were applied:

- All unnecessary processes (background or foreground) were terminated or suspended.
- The test application was given maximum priority, giving a preference to execute threads/sub-processes belonging to it over other processes of other applications that were required to be left running.
- Tests using different preprocessors would be run independently, but concurrently.

These first three strategies were used to minimise the effect of external process on the test results. Terminating, suspending and reducing the priorities of these process minimised the amount of available runtime the receive to the benefit of testing. The use of concurrent execution of tests allowed for any unavoidable effects to be spread among competing preprocessors, reducing any effect that these process may add to the statistical analysis of tests.

On current hardware, reading and writing to permanent storage, such as HDDs or SSDs, is a much slower task (though lesser for SSDs) then storing data to temporary RAM (disregarding the effects of Paging). To combat these I/O issues, all data sets were loaded and prepared before the timed execution of the target algorithm. Equally, all results generated were stored in memory and only written to permanent storage after the completion of all timed experiments. Further, the algorithms used to support the test were chosen such that they use minimal memory as opposed to fastest run time, reducing the amount of the paging the OS needed to perform.

The portion of the test application that performed the evaluations of the algorithm(s) was also performed concurrently. This increased testing throughput as multiple timed trials were executed at any one time; additionally, in the event of any external issues (as mentioned above) the time delay caused would tend to towards evenly applied to all of the algorithms being tested. Were possible, the total number of threads allowed was equal to one less than the number of cores available. This practise allowed one core to remain un-utilised by the timed tests, so it could be used by other threads belonging to the support process of the application or any external OS processes being executed.

Java and JVM Warm-up

The implementations used in this thesis were coded in the Java Programming Language. Programs written in Java, unlike typical programs, are not compiled to native machine code for the hardware/Operating System; instead they are compiled to the platform independent Java Byte Code. When it is desired to execute the compiled Java program, it is executed in a virtual environment (the Java Virtual Machine, JVM); providing various advantages and disadvantages when compared to native execution.

In the context of this work, the most notable difference between native programs and Java applications, is the ability to perform run-time optimisations. Like many modern compilers, the Java compiler performs various optimisations to code as it is being compiled, but it can only base these strategies on the static appearance of the program. As a Java program runs in the virtual environment, profilers analyse how the program is performing, how often various routines are invoked, the relationships between function arguments and the return values and many other such statistics. This type of run-time profiling within a virtual environment presents the ability to optimise the currently executing programs, such as (but not limited to) prioritising optimisations to the routines that are most frequently called, remove conditional checks that never change or replacing/prefixing functions with faster look-up actions.

The act of run-time optimisation, as is intended, can dramatically improve the run-time of a Java program; but this benefit becomes a detriment when it comes to statistical analysis and comparison of various algorithms. During the runtime of the application, the program will initially execute with the compile-time optimised version of the program, at some time into the execution of the program, some portion of the program will reach a sufficient threshold to be optimised by the JVM. The JVM will then generate a new version of the routine that is intended to perform the same functionality in a reduced amount of time and then re-route all new calls to it once completed. If this occurs during a timed test it can result in a noticeable and sudden drop in computation time on subsequent runs. If two or more algorithms are being compared sequentially and they share some routine as part of there implementation, then the run-time optimisations can result in bias towards the later running algorithm[1]. As the routine will have had X number of executions before the second algorithm executes, this increases the likelihood of

[1]This is assuming they have been run as part of the same execution of the JVM. The current JVM implementation does not maintain optimisations from one execution of the JVM to the next.

the JVM performing the optimisations early on in its timed trial. If the usage of the target routing remains similar between the two algorithms then the second algorithm will benefit from the speed increase produced from the optimisation. If the usage changes significantly, then the optimisation may still occur, but with the intent of improving the speed for the old usage (as this is the most common usage according to the profiler), potentially resulting in a decrease in performance.

The two most common strategies to overcome this problem is the either disable run-time optimisations or to perform JVM "warm-up". Disabling the run-time optimisation feature of the JVM is the simplest of the two options, but can mean the programs will run for extended amount of time than is necessary. For smaller programs this is a suitable solution, but for programs expected to run on the order of hours, or even days, this quickly becomes an ineffective solution. The warm-up strategy is to perform a pre-trial using the exact code that would be run as part of the timed trial. This allows for the profiler to analyse the various calls and routines used and optimise them as defined by its various optimisation policies. This pre-trial is not timed, and any other results from them are discarded. The duration of the pre-trial is sufficiently long enough such the the JVM will not perform any further optimisations during the main timed trials, typically due to optimisation policy exhaustion.

As all the problems tested in this thesis are stochastic in nature, the test typically last for long periods of time to generate statistically useful results. For this, the "warm-up" strategy was most feasible solution to the problem of JVM run-time optimisation. This strategy requires running the code additionally many times within the same execution before the beginning of any data collection for statistical analysis. For this to be effective, the warm-up of the algorithms in question must be representative of their used during timed trials and data collection.

3.1.2 Control Groups: Traditional Preprocessors

As discussed in section 2.2, traditional preprocessors use Bilinear or Bicubic interpolations to prepare images when down-sampling is required. For this reason, two additional control populations, one using images prepared by a Bilinear Interpolator and the other using Bicubic, will be form the control group for all experimentation in this thesis.

3.1.3 Stochasticity

Neural Network implementations are typically stochastic by nature, due to the random values used to initialise the weights in the network. To compensate for this, each test will be repeated a number of times with differing initial states and then averaging across the results. The exact number of repetitions will be dependant on the test being conducted and stated in their description.

3.1.4 Cohen's Kappa

Cohen's kappa coefficient (Cohen 1960, 1968) is a method to quantify the agreement between to classifiers, the 'Raters'. In the use-case of this thesis, Cohen's kappa is be used as alternative to accuracy by measuring the agreement between the ANNs prediction and the training data's ground-truth label. The advantage of using Cohen's kappa over more traditional metrics such as Accuracy is that its reported value uses chance as a baseline instead of perfect failure.

Computed as

$$\kappa = 1 - \frac{1 - A}{1 - p_e}, \tag{10}$$

where A is the traditionally computed accuracy and p_e is the probability of agreement. The probability of agreement can be calculated as

$$p_e = \frac{1}{N^2} \sum_c n_{c,a} n_{c,l}, \tag{11}$$

where N is the total number of tests evaluated, c is the set of valid classification outputs, and $n_{c,a}$ and $n_{c,l}$ are the total number of times those classifications were presented by the ANN and training labels respectively.

Typical values for Cohen's kappa range from 0 to 1, though negative values can theoretically tend towards $-\infty$. A value of 1 represents perfect agreement between the two Raters, and values close to 0 represents that the agreement between the two is no better than chance. If a value less than 0 is returned, this means there is a large amount of disagreement between the two Raters and is usually caused by the accidental inversion of one of the Raters.

3.1.5 Receiver-Operator Characteristic Curve

ANN classification systems are typically set up to return a normalised numeric value which can be discretised into a binary classification by a threshold value. The optimal value for this threshold is rarely 0.5. Receiver-Operator Characteristic curves (ROC curves) allow for optimisation of this threshold value and comparison between multiple models that implement independent optimal threshold values.

The ROC curve provides a mapping from False Positive Rate (FPR) to True Positive Rate (TPR) for the given classification method. The optimal value for the threshold can be identified as the point on the graph closest to the perfect classifier, 0 FPR and 1 TPR. Multiple models can be compared by this method with the comparison of the area under the ROC curve, AUC. A value of 1 (maximum) implies that at least one of the threshold values results in a perfect classifier and a value of 0.5 (minimum) indicates that the classifier is no better than chance.

The area enclosed by the ROC curve and the horizontal axis (the area under the curve, AUC) "is equal to the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one" (Wikipedia contributors (2018)). For this reason, it will be used to compare the various preprocessors.

3.2 Test Domain 1: Object Presence Detection

The first of these two test domains is a simple task to predict if a presented scene contains a predefined object. This is a fairly simple task and is primarily used to evaluate functionality and give some early insight into the algorithm. This domain was selected due to its simplicity in implementation and evaluation.

The task was implemented as the detection of an object against a complex background. For the sake of simplicity at this stage, the object (if present) was always in one of a set of predefined locations, with some variation in rotation with respect of the image plane. The object had to be of a non-trivial shape and the set of background images was constructed to contain both images of high and low colour contrast with the object.

Due to the design, it would be fairly easy for the neural network to learn the particular backgrounds and the target object, leading to over-fitting. As the primary interest in the test is for that of functionality (i.e. Can the ANN learn using inputs from novel preprocessors?), excessive learning that leads to over-fitting will not be a considered an issue at this stage.

3.2.1 Implementation Overview

This section will provide details as to the implementation specifics of this testing domain, covering the specifics of stochasticity, control group parameter selection and ANN structure.

To cover the issue of stochasticity (Section 3.1.3), each preprocessor test is repeated 200 times and results averaged. As this test constitutes a trial of the preprocessors functionality, as opposed to the preprocessor & ANN system quality, a fixed number of epochs (200) will be used.

Control Group: Parameter Selection

To find a suitable resolution for the control group, a quick exploration of the space was performed. In this exploration, a variety of potential downsampling resolutions were tested and ANNs trained with the resulting preprocessors. Figure 8 presents a visualisation of the results. The results show some interesting patterns that are not too surprising given the task. Resolutions that result in the location of object being split across multiple pixels significantly hamper the effectiveness of the ANN. An output image resolution of 6 pixels wide by 5 pixels vertically was selected for the preprocessors as it represents the resolution with the fewest resultant pixels that was able to learn the task.

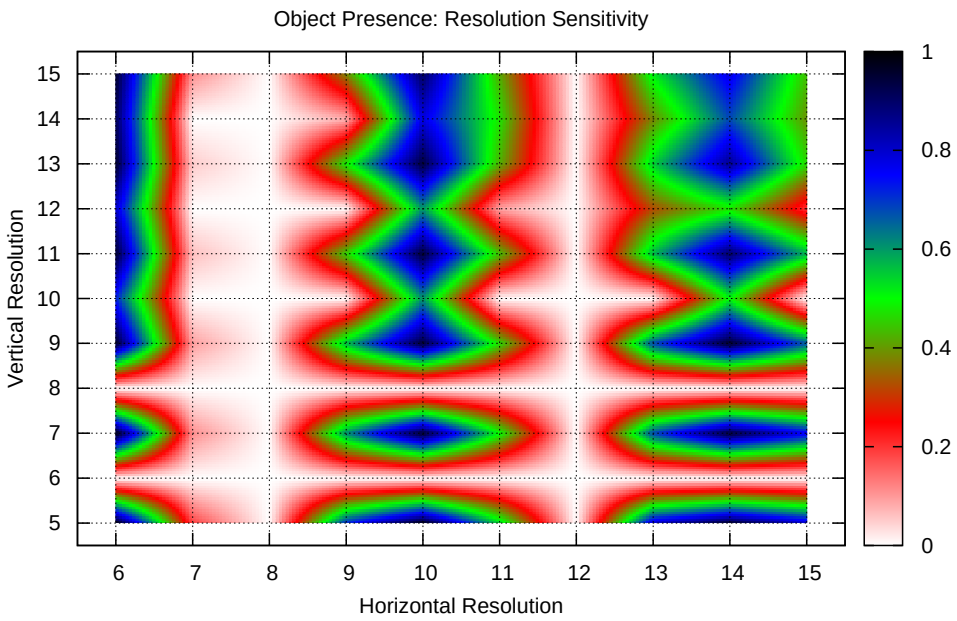


Figure 8: The accuracy (measured using Cohen's kappa) as a function of downsampling resolution.

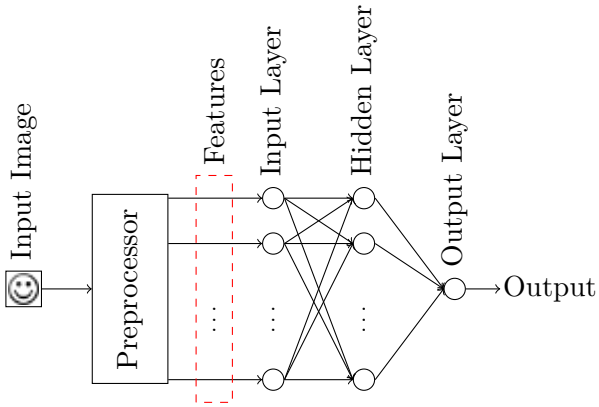


Figure 9: A graphic representation of the data flow through a complete system, from source image to final classification.

Topology and Classification

In the implementations of ANNs the question of topology poses a complex challenge. The solution to this is either to trial a collection of competing plausible topologies or to use techniques to generate the topology automatically (Stanley and Miikkulainen 2002). As part of the goal of this thesis, is the demonstration of preprocessor algorithms, the veracity of the ANN is not under scrutiny. Given the simplicity of this task, it was decided a small number of topologies would be trialled and the first one capable of learning under with the Traditional Preprocessors would be used.

The result object detection algorithm was implemented as a standard feed-forward perceptron based neural network with a topology of $I \rightarrow 2 \rightarrow 1$ (where I represents the number of neurons required to fully represent the data provided by the accompanied preprocessor). Early in the design of this test, it was decided to use an ANN that is still capable of solving the given task, but reduced to a minimal topology to intentionally slow the convergence to a solution; to increase the number of epochs available to evaluate the implementations. In an earlier implementation of the test domain, the ANNs were implemented with a topology of $I \rightarrow 10 \rightarrow 1$; with the accuracy beginning to plateau after epoch 5). Figure

9 shows a simple representation of this topological design. The output from the network was a value between 0 and 1 (inclusive). This was applied to a discrimination threshold, τ, to identify whether the value indicates an 'Object Present' prediction or a 'Background' prediction; allowing for comparison via ROC curves. Learning was implemented using back-propagation with a learning rate of 0.8.

3.2.2 Data Sets

The data set used in this preliminary experiment consisted of 300 generated images. Each image was generated from one of 17 possible background images[2] and, where relevant, the object to be found overlaid on the image in either the left or right half of the image, with a random orientation. The object used was an image of a stuffed bear[3]. To generate the 300 test images, the background set of was expanded to 100 images by selecting one of the 17 background images at random and producing a duplicate with altered the contrast and/or brightness. For each of the 100 images, two further duplicates were produced with the target object either centred in the left or right half of the image. The resultant 300 images were finally shuffled. The re-use of the background images was done deliberately to minimise the effect of learning the correct response based solely on the background.

3.2.3 Analysis

For this test domain, no statistical analysis will take place. The purpose of this test domain is to act as a filter for potentially viable algorithms. If an algorithm (or its variant) can reach accuracy measure of 0.5 for Cohen's kappa and have an average runtime of less then a factor of ten above that of the worst of the control algorithms, then it will progress on to Test domain 2.

[2]Images from around the Cornwallis building, University of Kent, Canterbury; See Appendix A.2

[3]see Appendix A.1, the bear had a simple colouration, complex shape, and a varying colour contrast with the used background images

3.3 Test Domain 2: Baseline Face Detection

To provide a qualitative comparison of the presented image preprocessing algorithms against the traditional methods, a baseline face detection algorithm will be used. Its design is based on a sliding window over an image pyramid methodology. This method was chosen as it allows for a location and scale independence search whilst keeping resource consumption and implementation difficulty low. This method only allows for minor orientation independence as an emergent property of the ANN rather that the face detection methodology itself.

3.3.1 Implementation Overview

Theo topology of the ANNs follow the same structure as those presented for Test Domain 1 above. Training of the ANNs will be performed continually until overfitting occurs. The training phase will complete once all ANNs have reached this stage. To cover the issue of stochasticity (Section 3.1.3), each preprocessor test is repeated 200 times and results averaged.

The basic design of the image pyramid is such that the largest image is sized to $5^1/_{16}{}^{\text{th}}$ the size of the original image to allow for searching of small faces in the scene. This value was chosen as each resolution step in the image pyramid is scaled by $^2/_3{}^{\text{rd}}$ the previous resolution, so after 5 passes, the scaled image is exactly the same size as the original image ($^{81}/_{16} = 5.0625$, $^{27}/_8$, $^9/_4$, $^3/_2$, 1, $^2/_3$, ...) and reducing artefacts produced from rounding errors (The fraction scales larger than 1 are all based on powers of 2). The pyramid was continued along this geometric series until either of the resultant width or height was less than windows size. For each pass, the window starts in the top left corner and steps right in 20 pixel increments; At the end of each row, the window returns to the left hand edge and drops down by a step of 20 pixels. The pass is complete when the window would clip the bottom edge of the image.

The specific size of the sliding window was set 100 pixels wide by 125 pixels tall, roughly approximating the proportions of a human face. To further reduce the generation of erroneous artefacts due to intermediate stages, the window cropping and scaling are performed on the source image (See Sec. 2.6). This results in the windows implementation being the inverse of the sequence presented above, with the window starting at $^{16}/_{81}$ its base size and increasing until the resultant window would exceed either axis of the source image.

Performing the Windowing

A step-wise description of the procedure is as follows:

1. Each face detection system has a sliding window that is initially $^{16}/_{81}$ the base window size of 100×125.

2. The sliding window is initialised to the top left corner.

   (a) For each location of the sliding window, the test image is extracted by interpolating the contents of the sliding window onto an image 100×125 in size, using the Bilinear interpolation method.

   (b) The windows section is then passed to the systems preprocessor and processed into the feature set.

   (c) The feature set is used as the input to the ANN and returns the prediction of whether the window appears to be over a face or the background. All windows identified as potential faces are recorded to a set of detections for further analysis. (See the next section for the specifics on the ANN.)

3. After processing, the sliding window is shifted 20 pixels right (before scaling, i.e. ≈ 3.95 pixels in the first pass) and the process repeated.

4. At the end of each line, the sliding window is moved to the left edge and down by 20 pixels (before scaling).

5. Once the sliding window has traversed the whole image, the size of the sliding window is increased by $^3/_2$ and moved back to the top left corner.

6. This is repeated until the one of the sliding widows axis is greater or equal to that of the whole image.

After the face detection process has been applied to an image, it returns a set of potential face detections. The following section looks at how this is further processed to determine which are correct detections (True Positive), which are incorrect detections (False Positive) and which labels have been missed (False Negative). Correct background detections (True Negatives) are inferred from these values.

3.3.2 Prediction Merging and Quantifying

The complexity of the prediction set is first reduced by merging detections that are potentially of the same 'stimuli'[4]. Predictions are clustered using the dbscan (Ester et al. 1996) algorithm, where two predictions are considered part of the same cluster if the ratio of intersection to union of the two windows is greater than 0.9. To collapse a cluster of window predictions down to a single prediction, a new bounding rectangle is computed such that has the following properties equal to the averaged equivalents of the clustered windows:

- Aspect ratio

- Location of centre

[4]Here, stimuli is used to describe a generic input that triggered the ANN to give a positive output. In a correctly trained ANN, this will typically be a faces, with the occasional false positive stimulus.

- Area

Two methods were considered for quantitative evaluation in this study. The first metric, *Correct Detection/False Positive* (Roth, Yang and Ahuja 2000; Feraud et al. 2000; Gu and Li 2000; Lew and Huijsmans 1996; Lin, Kung and Lin 1997; Osuna, Freund and Girosi 1997; Colmenarez and Huang 1997; Yang, Abuja and Kriegman 2000) was the typical measure used to quantity face detection of this type in the literature when this was considered a solvable, unsolved problem. The second method, the Bipartite Graph Metric (Jain and Learned-Miller 2010), is used in contemporary problems of the field for face detection. The following sections cover two different approaches to determine the truthfulness of a positive or negative prediction.

Correct Detection/False Positive

This method uses two values to quantify the quality of the algorithm, percentage Correct Detection (CD) and number of False Positives (FP). A correct detection is defined as a positive prediction for a window that contains all the facial features (eyes, nose and mouth) of at least one face. A false-positive is recorded for a positive prediction that does not meet the prior prerequisite. Though mostly negated by the merging performed prior, for the sake of completeness, repeat detections of an already identified face are not counted towards CD or FP (Rowley, Baluja and Kanade 1998). If a window with positive prediction actually contains multiple faces, all are marked as identified. Simply, CD is the quantity of faces from the data-set that where identified at least once, and FP is the total number of incorrect identification of faces.

Bipartite Graph Metric

Jain and Learned-Miller (2010) propose a newer quantitative model intended for uses in face detection algorithms where orientation and pose are also required to

be accounted for. In this methodology, each prediction (after clustering/merging) in the set of all predictions is connected to each label by an edge (forming a Bipartite Graph). Each edge is scored by ratio of intersected to union of the computed/merged bounding rectangle. A perfect prediction of a particular face will result in the connection between the two being weighted 1, whereas an unrelated face for a given prediction will be get a weight of 0. The Hungarian Algorithm is applied to the graph, generating a sub-graph that connects each detection to at-most one label, each label to at-most one detection, and that the sum of connection weights is maximised.

Quantifying Selection

In review of these two measures, it was decided to apply the modern standard of the Bipartite Graph Metric for the following reasons:

- Less prone to anomalous results due to duplicate detections - each prediction can be assigned to, at most, one target face.
- Clarity of association - there is a one-to-one relationship between label and prediction.
- Acceptance in the field - the use of CD/FP has fallen out of favour by practitioners in the field.

3.3.3 Datasets

The following data sets where used for the baseline face detection system. (See Fig.10 for samples.)

CMU-130

This data set was constructed from the union of the CMU frontal face data

(a) CMU A (b) CMU B (c) Caltech

Figure 10: Three sample images from three data sets. These provide examples of a single image with multiple faces, an image with a single face proportionally small to the image, and an image with a face that fills most of the image.

sets[5] A, B and C. The data set consists of numerous faces that are looking towards the camera in an upright manner, formatted in indexed grey-scale in a loss-less compression format. The data sets are provided with a ground truth labelling of each face in the scene, identifying the eyes, nose and mouth (2 corners and centre).

Caltech

The Caltech Faces 1999 (Front) data set[6] includes 450 images of faces from 27 distinct people that look towards the camera and are oriented upright. They are formatted in full RGB with negligible compression artefacts (due to the JPEG file format). The provided ground truth of this data set provides a rectangle identifying the location of each face.

3.3.4 Subset construction

As data sets CMU-130 and Caltech were to be used to construct the neural networks, they were segregated into the Building[7], Validation[7] and Test sets. To do this the images, along with associated ground-truths, of both sets were initially loaded into memory. The first 10% of faces were separated into the Test sets,

[5]Data sets A and C were collected by Rowley, Baluja and Kanade (1998), whilst data set B (also known as the MIT face data set) was compiled by Sung and Poggio (1998)

[6]Caltech Faces 1999 (Front) was compiled by Weber (1999)

[7]The Building and Validation sets collectively make up the Training Set

T(CMU-130) and T(Caltech). The faces from the remaining images are extracted (cropped and scaled), along with an equal number of background samples[8]. The faces and background images are randomly assigned to either the respective Building set or their Validation set, so that 90% form the Building sets, B(CMU-130) and B(Caltech), whilst the remainder 10% form the Validation sets, V(CMU-130) and V(Caltech).

3.3.5 Analysis

During the training phase, average processing time for the preprocessor and corresponding neural-network are gathered. Accuracy results are gathered during the verification phase as predictions relative to some discriminating threshold, τ. τ is varied from 0 to 1, to produce a ROC curve (Sec. 3.1.5).

The comparison of the preprocessors (both time and accuracy) will be performed by Friedman tests with a significance threshold of $\alpha = 0.05$. In the event of significance, this will be followed up by post-hoc Wilcoxon signed-rank tests (with Bonferroni correction).

[8]A background sample is considered to be a section of the image that would fail to register as a correct detection, as described in section 3.3.2.

Chapter 4

Image Downsampling and Preprocessors

This chapter introduces the overarching concept of adaptive preprocessors and a collection of novel terminology used to classify and distinguish different forms of preprocessors.

In the following chapters, three novel preprocessors will be presented. Two of these are designed to be constructed in a way significantly different to the majority of those in the literature. To aid in the identification and discussion of these forms of preprocessors some new terminology is introduced. The majority of image preprocessors and one of the novel algorithms presented here are constructed independent of the learning process of the latter classification system. These preprocessors come under the class of Bespoke Preprocessors as each instance of the algorithm is constructed (in terms of algorithm selection and parameter optimisation) for a specific case. The second class are the Adaptive Preprocessors; these preprocessors are designed to be automatically adjusted and modified in conjunction with the learning system, reducing the equivalent construction to algorithm selection and possible initial parameter biasing. Figure 11 represents a stylised view of how an ANN based system for image processing along with

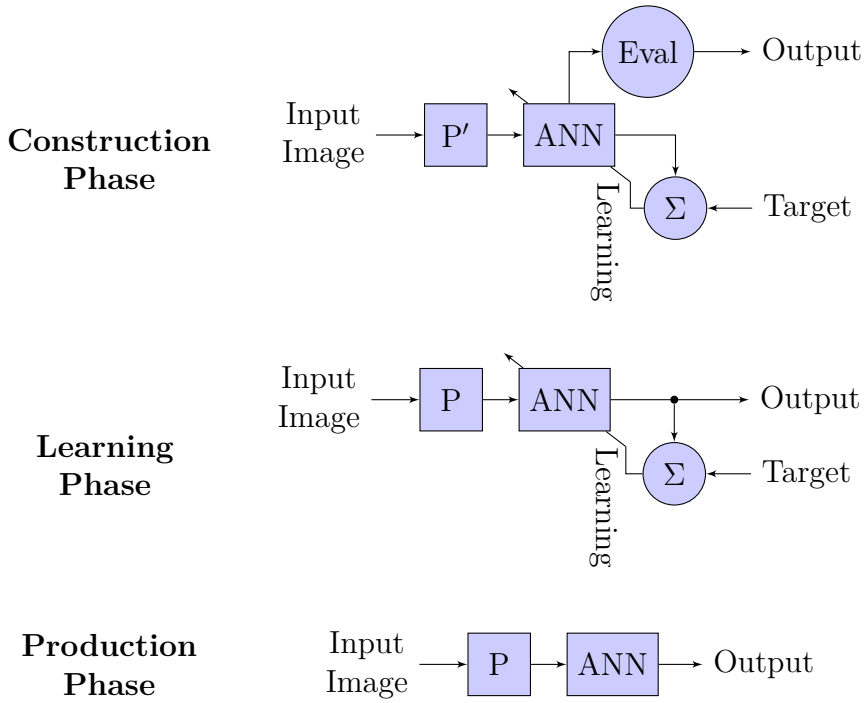


Figure 11: A representation of the additional phase for a Bespoke Preprocessor. P′ represents some sample preprocessor(s) used during the construction phase. Eval represents an evaluation function used to quantify the ANNs behaviour, with respect to P′. Activities, such as a Trial-And-Error refinement of a preprocessor, are represented by the Construction Phase. The preprocessor, P, used in the main learning phase and into production may be constructed given the results of Eval over the set of {P′}, or the instance from that set that allowed the ANN to score the most preferential value from the Eval function. Note that in this diagram an ANN using a supervised learning algorithm has been depicted. This is not a requirement of, but rather an easily representable and recognisable way to show where the ANNs traditional learning fits within this life-cycle.

Bespoke preprocessor may function and the stages it would migrate through during its life cycle.

Construction Phase

The first stage in a model's life-cycle where data preparation techniques are refined for the given model. During this phase, learning models may require a training process or the process may be iterated multiple times with varying components as part of the exploration/construction.

Learning Phase

A stage in a learning model's life-cycle where the primary training occurs. A common strategy is to generate a pool of candidates and train them against the Training Set. The best performer is then selected for the Production Phase.

Production Phase

The final stage in a model's life-cycle where no or restricted alterations are applied, and the algorithms can be replaced with hard-coded or static variants for improved run-times.

The Bespoke Preprocessor is generated as a result of a process that occurs prior to the traditional Learning Phase, the Construction Phase. An advantage of Adaptive Preprocessor is the partial development during a shorter Construction Phase resulting from its ability to iteratively refined itself alongside the ANN during the Learning Phase. In both cases, evaluation metrics that quantify the behaviour or quality of the ANN's predictive ability are a requirement to provide insight into whether features provided by the preprocessor are useful towards the predictive or classification accuracy of the ANN.

Bespoke Preprocessor

In the context of this thesis, Bespoke is used to describe a preprocessor that

is fully constructed prior to the main learning of the ANN. The life-cycle of bespoke preprocessors generally follow the same path. First the developer will use priori knowledge to determine a short list (domain) of potential candidate preprocessor algorithms and, potentially, some restrictions on parameter ranges. Through an evaluation process either, this short list is then refined such that one is selected as the final preprocessor to use; or a specific preprocessor can be constructed from the given evaluation data of the short list. Figure 11 provides how a Bespoke Preprocessor may be used and constructed through the life-cycle of a ANN using a Supervised Learning model. In this figure, the preprocessor P' represents a particular instance of one of the preprocessors from the short list and the parameters used in instantiating it.

Adaptive Preprocessor

The intention of an Adaptive Preprocessor is to simplify the task of constructing a preprocessor by providing systems that perform refinement during the training of the ANN. This style of development requires three components, a domain in which the adaptive preprocessor should be searching (similar to the short list required by Bespoke Preprocessor generation), a quality metric by which to compare parts of the current preprocessor (the equivalent of the Bespoke Preprocessor's evaluation function), and an additional method to present new preprocessors based on those metric results and the previous generation preprocessor, not dissimilar from the genetic operators of Genetic Algorithms. This allows for less work on the part of the developer as they only define the short list using the same a priori knowledge as is used for the bespoke preprocessor, with the option of culling the short list to speed up the iterative refinement and the attachment of the Adaptive Preprocessor to the ANN via the quality metric. Figure 12 provides a graphical representation of a potential Adaptive Preprocessor Model.

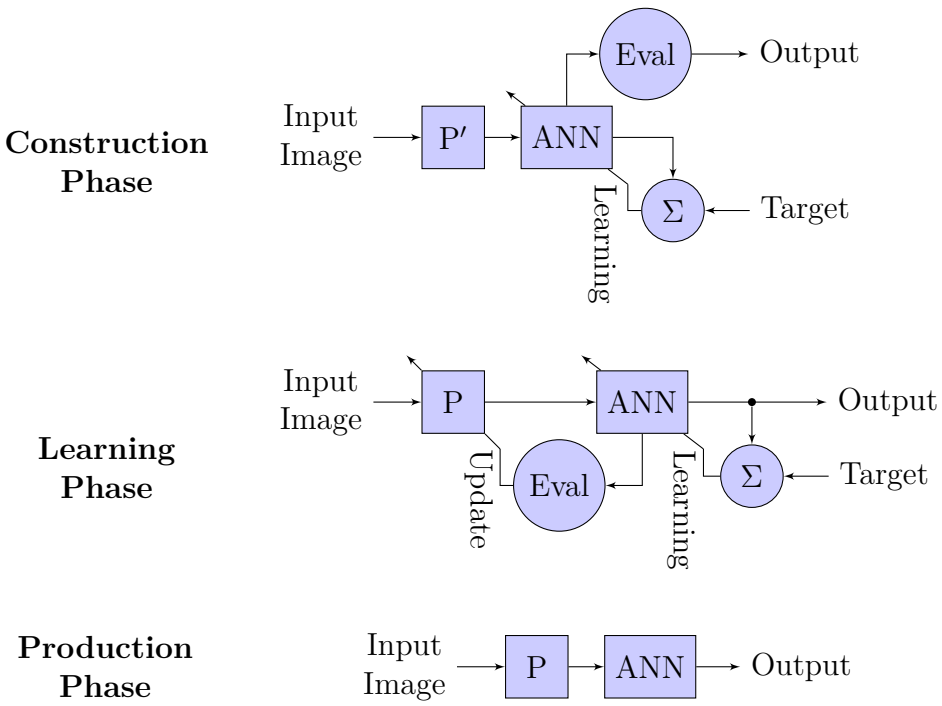


Figure 12: A representation of the life-cycle of an ANN based system using an Adaptive Preprocessor. Note the addition evaluation function, Eval, and an Update process added to the Learning Phase process, used to modify the preprocessor in parallel with traditional ANN learning.

A minor issues that arises from these two definitions is the question of where is the boundary drawn between the classifier and the preprocessor. In the case of Bespoke Preprocessors, the delineation is rather clear as the preprocessor is a static process, prefixed to the learning classifier. In regards to the Adaptive preprocessor, the delineation is not as clear; at what point in a classifying system (made from a sequence of, potentially learning, sub-processes) does the 'classifier' moniker describe the behaviour better than a learning 'preprocessor'? Too resolve this issues, two potential extensions to the definition of preprocessor are presented:

Domain Conversion

The additional sequence of component algorithms (the 'preprocessor') prefixed to a sequence of components (the 'classifier') that perform non-image

classification. An trivial example of this may be a feature extraction process used to preprocess an image for a decision tree. The decision tree is the component performing the classification and (though technically capable of taking in the colour values of a standardised image), it is trivial to demonstrate that a decision tree processing the raw image would be of poor quality. In this example the feature extraction algorithm is used to take the source image from a domain that is inappropriate to one that is sufficient to the algorithm.

Qualitative Refinement

The sequence of optional components (the 'preprocessor') prefixed to the minimum sequence of components (the 'classifier') that perform image classification to improve accuracy measures, simplify the user/developer's set-up requirements (in both time and/or knowledge) or to reduce the training time; but not required, at least conceptually, to provide the minimum requirements for performing even poor quality classification.

The extensions here are designed fit with the intuitive idea of what a preprocessor is and allow for a way to cleanly mark where the preprocessor end and the classifier begins. In the previous chapter, it was stated that normalising the generated features is part of the duty of the preprocessor. This was done to reduce any biasing that may affect evaluation of the various instantiations. Using an ANN as the learning classifier, it is intended to take input values in the range 0..1, but is able to function with any finite value and perform the normalising internally; therefore, by Qualitative Refinement, the boundary between the classifier and the preprocessor coincides with that used in this thesis.

Chapter 5

Voronoi Saliency Partition

This chapter will discuss the first novel image preprocessor introduced in this thesis, Voronoi Saliency Partition (VSP). The concept of this preprocessor is to use a set of convex shapes to define the partitions of the image space into features to be presented to the ANN as its inputs. These convex shapes are generated by Voronoi Diagrams from a set of points, iteratively refined by the Update process (Chapter 4). These points are adjusted in response to an evaluation function that determines how much of an effect the individual features have on the selection of a classification or prediction by the ANN. The feature is computed from the pixels whose centres fall within the bounds of the shapes when the diagram is scaled to the size of the source image.

The choice of using Voronoi Diagrams as a foundation for this first implementation comes from a number of its properties:

- If $S \Rightarrow P$ describes the process of converting set of points, S, to partitioning of space, P, then the likelihood of finding two sets, S_1 and S_2 (Assumption: $|S_1| = |S_2|$), such that $S_1 \Rightarrow P$ and $S_2 \Rightarrow P$ decreases with the size of the sets. This means that for sufficiently large sets, care does not need to be taken to protect against the exploration of distinct sets of points resulting in a looping exploration of partitions.

- Both the set of points and the partitions are continuous; a smooth transition from S_1 to S_2 will result in a smooth transition from P_1 to P_2 (where $S_1 \Rightarrow P_1$ and $S_2 \Rightarrow P_2$).

First, this chapter will cover some additional terminology (both explanation of concepts from other works and variations of other's work used in this chapter). The design of VSP will be a two stage process, both of which will require some comparative tests, so the next section will focus on how these tests will be performed and their quality evaluated. The following section will introduce the first design process: "identifying a usable down-sampling algorithm based on Voronoi diagrams", followed up by a comparison of the presented methods. The second stage will expand this design by adding one of a number of adaptive update processes that will iteratively refine the sites that define the Voronoi diagram preprocessor to improve the task-suitability of the preprocessor; the comparison is then carried out using this expanded preprocessor. This will finally be brought into overview with a conclusion for the work presented in this chapter.

5.1 Voronoi Diagrams

A Voronoi diagram (VD, also known as a Dirichlet tessellation, a Voronoi-Dirichlet partition and other variations) is defined as the partitioning of an N-dimensional space, X, for which a distance function, $\text{dist}(a, b)$, can be defined for any two points a and b within that space(Dirichlet 1850; Voronoi 1908). It is constructed from a set of points (sites), $S = \{s_1, ..., s_n\}$ within that space, $S \subset X$. Each site, $s_x \in S$, determines a partitioning of the space (known as a cell), $\text{c}(s_k)$, that contains all the points whose distance from s_k is no greater than that of any other site; formally

$$\text{c}(s_k) = \{x \in X \mid \forall s_j \in S : (\text{dist}(x, s_k) \leq \text{dist}(x, s_j))\}.$$

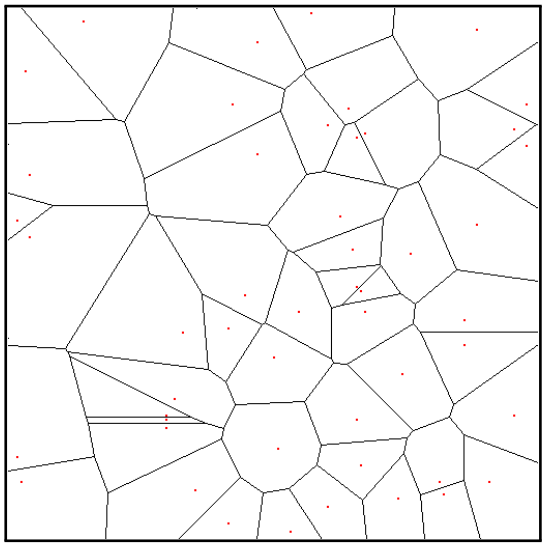


Figure 13: An example Voronoi Diagram of 50 randomly located sites in a planar euclidean space. The sites are highlighted in red, with cell boundaries marked in black.

This results in a partitioning of the space made up entirely of convex polygons (an example of which is shown in Fig. 13) with each point in the space belonging to one cell (the internal area of the enclosing cell), two cells (making up the delimiting edge between them), or more (marking the vertex of multiple cells).

5.2 Saliency: Measuring an Neurons 'Worth'

Throughout this thesis is the use of a metric called Saliency. This metric is used to estimate how much predictive value an input or computed feature (from individual neuronal units) provides to the overall predictive quality of an ANN. First will be a discussion on its inspiration, Relevance Assessment, originally used as part of a pruning algorithm for ANN topologies. This will be followed up by the adjustments and simplifications applied to the model and the reason behind them. This will be finished of with a look at how the Saliency metric may be used beyond the original proposed purpose presented in this thesis.

5.2.1 Foundation: Relevance Assessment

The Saliency metric is derived as a simplification of earlier work, Relevance Assessment, intended to identify neural units for removal as part of a pruning algorithm. In the work by Mozer and Smolensky (1989b; 1989a), they describe a method for assessing the relevance of a unit within a multi-layered feed-forward neural network. In this model, they conceive each neural unit to have a additional coefficient, α_i, that represents "attentional strength" (1989a, p. 2). Given a typical computation unit, Eq. 1 (Sec. 2.1.1), the output of the computational unit can be conceptualised as

$$o_j = \text{norm}(\sum_i w_{ij}\alpha_i o_i). \tag{12}$$

The value of α_i "can be thought of as gating the flow of activity from the unit" (1989a, p. 2), used to enable ($\alpha_i = 1$) or disable ($\alpha_i = 0$) unit i. From this, the relevance of the unit i, ρ_i, can be described as

$$\rho_i = E_{\alpha_i=0} - E_{\alpha_i=1}, \tag{13}$$

where E represents the computed error under the two attentional conditions.

Mozer and Smolensky approximate the value of ρ_i by "using the derivative of the error with respect to α" and assume that the resulting equality "holds approximately for γ=0" (1989a, p. 3).

$$\lim_{\gamma\to 1} \frac{E_{\alpha_i=\gamma} - E_{\alpha_i=1}}{\gamma - 1} = \left.\frac{\delta\, E}{\delta\, \alpha_i}\right|_{\alpha_i=1} \tag{14}$$

$$\frac{E_{\alpha_i=0} - E_{\alpha_i=1}}{-1} \approx \left.\frac{\delta\, E}{\delta\, \alpha_i}\right|_{\alpha_i=1} \tag{15}$$

$$\rho_i \approx \hat{\rho}_i = -\left.\frac{\delta\, E}{\delta\, \alpha_i}\right|_{\alpha_i=1}. \tag{16}$$

Here, $\hat{\rho}$ is used to identify the approximated value of ρ.

Mozer and Smolensky found that their definition varied considerably in time, so a "exponentially-decaying time average of the derivative"(1989a, p. 4) was used to produce a smoother resultant curve,

$$\hat{\rho}_i(t) = \begin{cases} 0 & t = 0 \\ 0.8\hat{\rho}_i(t-1) + 0.2\,|\frac{\delta\,E(t)}{\delta\,\alpha_i}| & t > 0, \end{cases} \tag{17}$$

where t here represents time. Mozer and Smolensky's equation shown here uses an 80% weighting towards past values for the stability they desired. Such a high weighting would make such an algorithm slow to react to sudden changes. For brevity, this exponential decay will be omitted from described values of $\hat{\rho}$ in context of *Relevance Assessment*.

For the purpose of approximating ρ_i for output units, they recommend that the error is defined by the linear error function

$$E = \sum^{P}\sum_{i}^{N}|t_i - o_i| \tag{18}$$

where P is the set of patterns, t_i is the target or training value for unit i for that pattern, and N is the set of output units.

For hidden and input units, the relevance is computed during an additional feed-back phase (in a similar manner to back-propagation) as

$$\hat{\rho}_i = \sum_{j}^{N_i} w_{ij}\hat{\rho}_j, \tag{19}$$

where N_i is the set of neurons that use unit i as an input and w_{ij} is the weight between them.

The issue with this definition is that the output units relevance is dependent on the latest input received by the ANN and the approximated relevance varies considerably with time. Mozer and Smolensky attempted to rectify this issue by

an "exponentially-decaying time average of the derivative" (1989b, p. 4),

$$\varrho_i(t) = 0.8\varrho_i(t-1) + 0.2\hat{\rho}(t)\,, \tag{20}$$

where $\hat{\rho}(t)$ is the approximated relevance from equations (19) or (17) at time t. This temporal smoothing makes the computed relevance dependent on the order in which the training set was presented, which can cause or exaggerate issue resulting from poor choices in the training set.

Other metrics, like 'Sensitivity Analysis' (Engelbrecht, Cloete and Zurada 1995), solve the temporal noise problem by parsing the entire data-set, then computing a metric for each unit. This still leaves the sensitivities generated dependent on the patterns in the data-set; such that a poor choice of test patterns can still result in useless sensitivities. Other issues may arise due to requirements of processing time and memory usage for such an algorithm.

5.2.2 Saliency: Definition

For use in the preprocessors presented here, the relationship with the current output value would need to be removed from the previous equations. For the used metric to provide valid insight into the ANNs usage of neural units in this context, the returned value must be related only to the structure of the ANN; it should not have any relationship to recent executions of the ANN, further enforced by the fact that evaluation by this metric may occur before any evaluation has occurred. An attempt to solve these problems resulted in a reworking of the ideas used in Relevance Assessment, and a small simplification to the implementation. This new definition is presented as 'salience'; a measure of the amount of contribution by each input or neural unit has to the output classification/prediction of an ANN.

As this model is not interested in the contribution of the inputs to the ANN's

predictive accuracy, but rather to the ANNs output function, it becomes independent from both the task and, by extension, the training set. Following this pattern, an output unit is described as having a saliency of exactly 1, based on the argument: 'if the output was not useful/salient, then the model would not be attempting to extract that piece/type of information.' This simplification to the output salience allows the design to stable and consistent evaluations of an ANN, avoiding the stability issues experienced by Mozer and Smolensky, allowing for the removal of the exponentially-decaying time average from the definition.

Additionally, to keep the values constrained, the approximated saliency of a hidden unit is normalised using the hyperbolic-tangent. This function was chosen over the typical *Logistic* function due to the increased demand for granularity (which may otherwise be lost to precision based rounding errors of floating-point numbers) of salience measures as they tend towards zero.

$$\hat{\rho}_i = \begin{cases} 1 & \text{when } i \text{ is an output,} \\ \tanh\left(\sum_{j}^{N_i} |w_{ij}| \hat{\rho}_j\right) & \text{otherwise.} \end{cases} \tag{21}$$

A problem to be aware of when using Saliency as a metric, is that the reported values can be influenced by the input domains. If an input has a wide domain, then the ANN has to perform the task of scaling and biasing the values in addition to predicting the classification of the feature set. The inclusion of scaling to the ANNs task can result in poor Saliency computations as the scaling sub task attempts to adjust the weight toward extreme values to compensate and the derivatives typically used in ANNs slow weight migration down. Different normalisation functions with Saliency-compatiable derivatives may be a solution to this, but due to the prevalence of continuous normalisation functions with derivatives that produce bell curves, the simplest solution is the normalise all of the input domains. Target ranges (for finite domains) of [0..1] or [-1..1] or normal distributions with a standard deviation of 1 centred on 0 are the preferred input domains when using

Saliency. For this thesis, all image data is converted from the range [0..255] for each colour channel and normalised to the range [0..1] by simple division.

5.2.3 Alternate Uses

In this thesis, the primary focus is on the usefulness of an input, and thus the term Saliency is used to assign that notion to the model (i.e "How salient is that section of images to the task?"). An alternative uses for the same metric is similar to relevance assessment: a review of an ANN to determine how influential any intermediate neural unit is to subsequent units. In these cases, the neural units of interest have their saliency set as 1 and extra output units to 0. The application of the saliency function to each neural unit without an assigned saliency backwards through the network would generate a measure of how much effect the neural units have on the units of interest.

Systems, such as Google's DeepDream project (Mordvintsev, Olah and Tyka 2015a,b), were originally developed to identify the usage of individual neural units within a CNN. In this system, the supplied image would be altered ever so slightly as to increase the confidence level of a targeted neural unit. Over a number of iterations the 'concept' that the targeted unit encodes for would be come visually apparent in the generated output. In a similar vein, the Similarity metric presented may provide useful in identifying the relationship between neural units encoding simpler concepts and units further along the network encoding for higher level concepts.

5.3 Research Question

To explore this idea, the following research questions are proposed:

1. Can a Voronoi Diagram be used as a sound basis to partition a 2D space for the purpose of image preprocessing to usable features for an ANN?

2. Can such a preprocessor allow the associated ANN to achieve an improved measure of accuracy at a given task?

3. Do the increased computational overheads in the preprocessor allow for a greater reduction in computation in the ANN?

To answer these questions, an implementation of the preprocessor will be constructed and tested against the test domains presented in chapter 3. To answer the first question, the preprocessor will be tested against Test Domain 1: Object Presence Detection, Section 3.2. The traditional preprocessors will also act as a control group in this testing domain to give early insight onto the likely outcome of questions 2 and 3. Failure of the preprocessor to compete with the control group will be viewed as a contradiction to questions 2 and 3, whilst an equivalence or improvement at this stage will require confirmation using Test Domain 2: Baseline Face Detection, 3.3.

5.4 Preprocessor: Cellular Downsampling

With the descriptions of the VSP conceptual model and testing environment presented, this section will focus on the implementation of the preprocessor portion of the VSP. This section will discuss three methods that process an image into a feature set. The following section will then test the three models to identify the most productive. Subsequent sections will then focus on potential models for the adaptive portion of the VSP algorithm.

The most obvious method of down-sampling based upon Voronoi Diagrams, is the generation of a single feature for each cell in the Voronoi Diagram by the aggregation of pixels within the Cell's coverage. As the base algorithm will be independent of the size of the source image, the centre of each pixel will be mapped to fit within a unit square, onto which the Voronoi diagram will be projected. All the pixel centres that fall within the area covered by each Cell will be aggregated

Figure 14: This figure demonstrates the downsampling of an image from the Training Set. The source image is reduced to a vector of 300 values (100 cells, each with three colour channels) and the cellular image reproduced from the Voronoi Diagram used to perform the downsampling.

and the resulting colour assigned to the related site as the reported feature of the algorithm. Three variations will be tested, each using one of the following weighted average methods for aggregating pixel colours:

Flat Each pixel will be averaged equally to all other pixels.

Inverse Distance Each pixel will be weighted inversely proportional to the distance between the mapped pixel centre and the sites location.

Inverse Distance-Squared Each pixel will be weighted inversely proportional to the square of the distance between the mapped pixel centre and the sites location.

Figure 14 shows an extracted initial state from one of the testing runs discussed in the next section. The colour aggregating method used in this figure is Inverse Distance-Squared.

If the image source is constant and known ahead of time, a large portion of the computations can be precomputed for each assignation of Voronoi sites to improve run-time speed at the expense of increased memory cost.

5.4.1 Initial Site Positions

For the given testing domain, the Control group will down-sampling to a 6×5 image. For comparison, the VSP based preprocessors used in the following test will use 30 sites. For the first test they be positioned randomly within the unit square. The second test is performed with the 30 sites arranged in a Centroidal Voronoi Tessellation[1] of the unit square. The final test will position the 30 sites in a rectilinear formation, 6 horizontally and 5 vertically, to match the rectilinear nature of the traditional preprocessors.

5.5 Testing the Downsample

This section presents the results of performing the 'Object Presence Detection' test against the VSP algorithms. The results for the results for the control group is also presented to show context and provide a target 'desired' result. Figure 15 and Table 3 present the accuracy per epoch and average runtime respectively of each algorithm tested. Each of initialisation methods discussed in section 5.4 were also tested. In the same section, it was also suggested that the down-sampling algorithms could be optimised for computational runtime if the image size for the task was known and constant. As this was the case, a brief second test was also performed using pre-computed variants of the VSP algorithms. This test only used the Rectilinear initialisation and was not tested against a control group as only computation time was of interest. Figure 16 presents the accuracy per epoch of the precomputed VSP test; and Table 4 presented the runtimes.

Of the 11 tests performed, only three show learning taking place en mass. As expected, Bilinear and Bicubic quickly converge on a potential solution in about 5-6 epochs. Inverse Distance-Squared initialised by the Rectilinear method show

[1] A Voronoi Tessellation of a space is considered centroidal when the location of a site also equals the mean of the cell it defines.

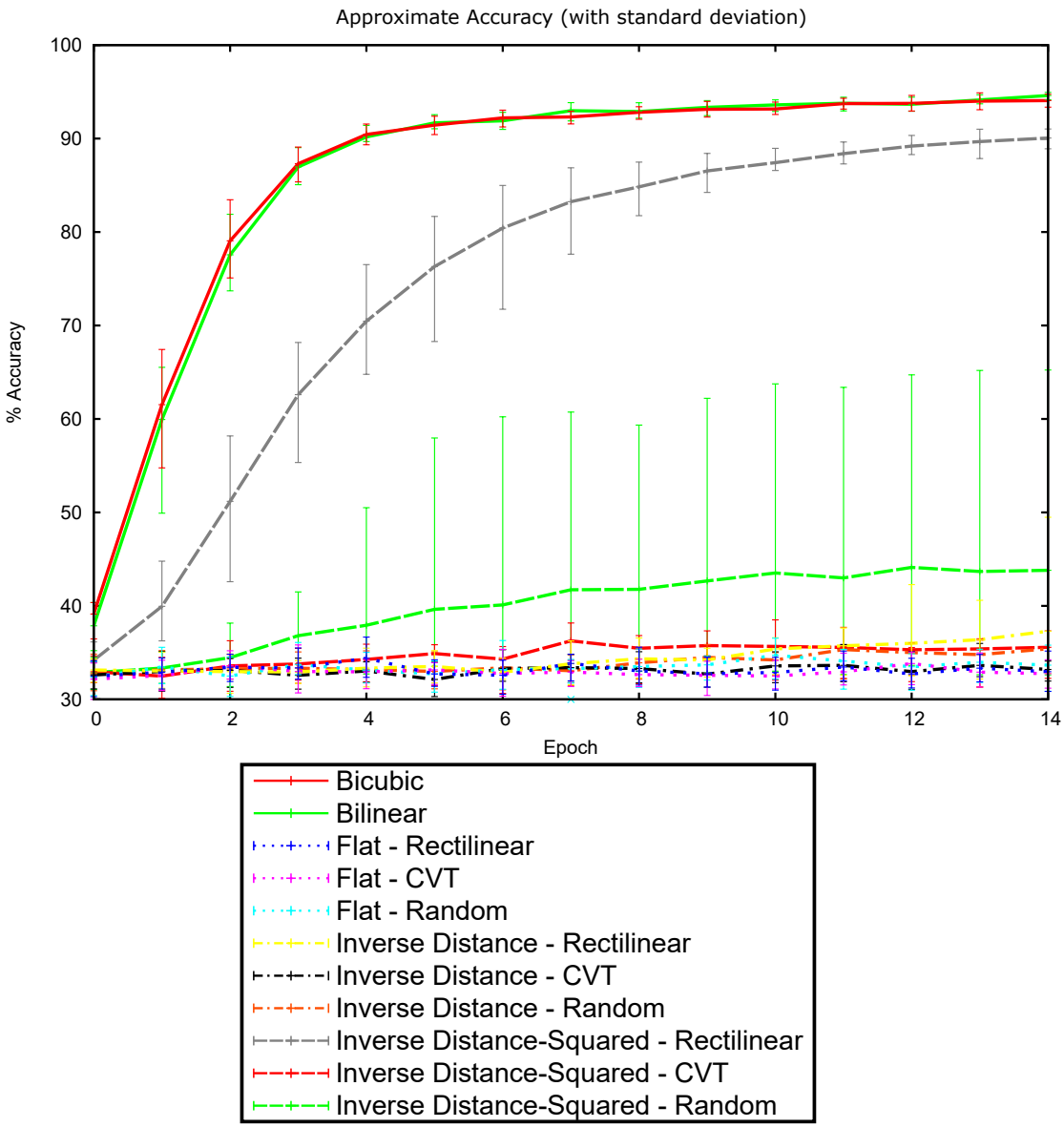


Figure 15: Figure showing the accuracy per epoch of the various VSP downsampling algorithms (Flat [Dotted Lines], Inverse Distance [Dot-Dashed Lines] and Inverse Distance-Squared [Dashed Lines]), their initialisation method (Rectilinear, CVT or Random) against the two control preprocessors (Bilinear and Bicubic [Solid Lines]). Standard deviation above and below the mean for each epoch are presented.

Table 3: Average computational Runtime ($\overline{time}$) for the VSP Down-sampling Algorithms and Control Group with standard deviation (σ)

Method	Rectilinear		CVT 30		Random 30	
	$\overline{time}$	σ	$\overline{time}$	σ	$\overline{time}$	σ
Bilinear (Control)	1.74ms	0.48ms				
Bicubic (Control)	2.29ms	0.74ms				
Flat	4.74s	0.38s	4.50s	0.92s	4.52s	1.11s
Inverse Distance	4.95s	0.52s	4.73s	1.02s	4.81s	1.11s
Inverse Distance-Squared	4.77s	0.28s	4.55s	0.96s	4.59s	1.01s

Table 4: Computational Runtime for the precomputed VSP Down-sampling Algorithms

Method	Rectilinear
Flat	673.5ms
Inverse Distance	667.2ms
Inverse Distance-Squared	671.9ms

learning having taken place, but not as fast as the two traditional methods; by epoch 14, its average accuracy is equivalent to the traditional methods at epoch 4. The variance bars and anomalous entries show some individuals of Inverse Distance-Squared when initialised by CVT or Random could also learn part of the task (asymptoting to about 60% accuracy by around epoch 7). A few individuals of Inverse Distance - Rectilinear, Inverse Distance - Random and Flat - Random can be seen starting to learn from epoch 6. The rate of ascent of these three implies that they were also only capable of learning part of the task, but the test did not continue on long enough to provide a definite conclusion.

With a three orders of magnitude increase in computational runtime, the base versions of the VSP algorithms, as presented, perform very poorly. The precomputed versions mitigate slightly, reducing the computational runtime to only a two order of magnitude increase.

Overall, the Inverse Distance down-sampling algorithm, initialised by the Rectilinear methods, clearly out performs the other VSP candidate algorithms in terms of overall accuracy on this task. The large different in computational runtime

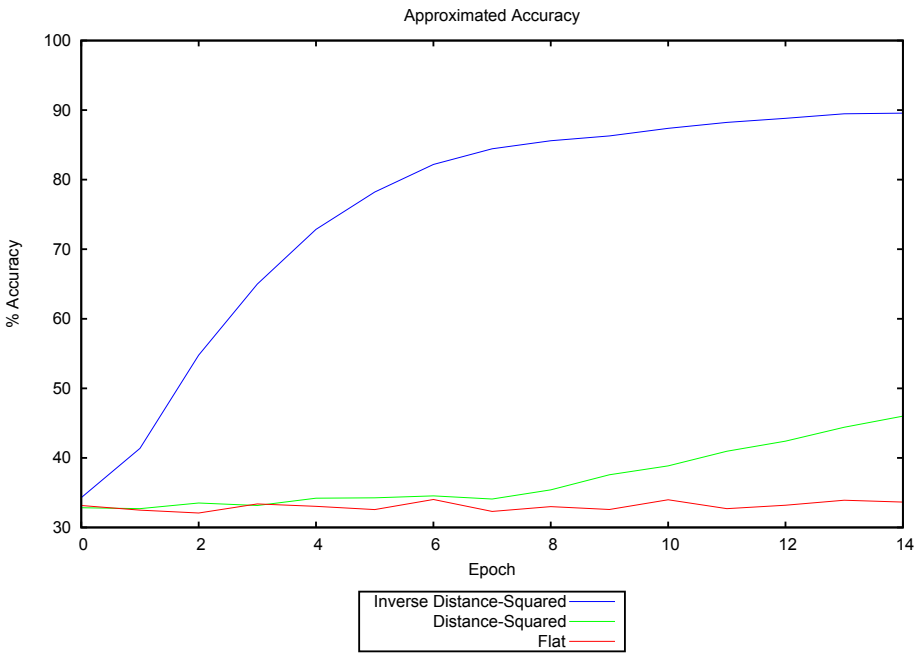


Figure 16: Figure showing the accuracy per epoch of the precomputed variations of the three presented VSP down-sampling algorithms, all using the Rectilinear initialisation method.

is a serious issue for this algorithm and can only be mitigated by optimisations/refinements to the down-sampling or the addition of the update portion of the VSP algorithm providing significant improvements to accuracy.

5.6 Update: Adaptive Site positioning

Using the Salience metric (defined Sec. 5.2), it is possible to determine which features provide low predictive value to the network. As there is a one-to-one relationship between each of the ANN's input units and the output features produced by the preprocessor, this salience of the input unit can also be said to describe the feature itself. It is easy to argue that sites that generate high saliency features are situated in valuable locations (with regards to the task at hand), as the high saliency implies that the learning algorithm has increased the magnitude of those connections. The converse can also be argued for the sites that have low salience

scores. Using this knowledge, it is possible to identify inputs that are safe to alter as a low saliency scores means that any alterations to the Input Context of the related feature (such as the site's location) will result in negligible (if any) change in the output of the ANN; thus any inputs of sufficiently low saliency can be considered suitable candidates for alteration as part of the update process.

5.6.1 Particle Inspired Update

A challenge with this type of adaptive site updating is not only the identification of sites that are suitable to move location, but also by how much to move. Using 2D particles simulations to take advantage of the nature of sites (as 2D points on the unit euclidean plane), it is possible to reconstruct the problem into the definition of a custom force computation based upon properties of simple 2D particles (representative of the sites). In practise, the implementation bares a strong resemblance to the flocking simulations described by Reynolds, (Reynolds 1987), but is conceived as a simulation of high and low energy particles on some 2D field.

The only properties of consequence for this application is a particle's location and the saliency of the resultant input feature. The simulation can be performed at a rate of one 'tick', single application of the physics algorithms, per learning of the ANN such the update function is responsive to changes in the saliency of sites. Increasing this tick rate to an integral number of ticks per application of the learning algorithm could also be used as a way to smooth or steady any jitter in the particle simulation.

Two fuzzy classifications can be defined to easily describe the nature of the Sites: High Saliency (HS) and Low Saliency (LS) sites. As there names suggest, HS Sites have saliencies that are close to 1, where as LS Sites have saliencies close to 0. Using these terms, higher order behaviours can be defined that described what the apparent motion of the particles as produced by some unknown underlying

mechanics (to be defined later):

1. HS particles should move negligibly, to prevent them from migrating away from regions of predictive value (with respect to the ANN's task).

2. LS particles should move towards HS particles so that the resultant cells focus in on predictive areas.

3. LS particles should migrate away from other LS particles so that their cells consume as much of the un-predictive parts of the scene with as few cells as possible.

4. All particles should be prevented from becoming too close (where 'too close' is task, image resolution and site density dependent).

From these behaviours, the following three forces where developed:

- Firstly, F_1 is a saliency based attractive force, intended to draw particles towards other particles of higher saliency, satisfying the requirement of the second behaviour.

$$F_1(i,j) = s_j(s_i - 1)\theta, \tag{22}$$

  where s_x is the salience measure associated with site x, and $0 \leq \theta \leq 1$ is a tunable scaling factor used to adjust the speed of motion in the simulation.

- F_2 is a saliency based repulsive force that is strongest between sites of low saliency, satisfying behaviour three.

$$F_2(i,j) = (1 - s_i)(1 - s_j) \tag{23}$$

- F_3 is a proximity based repulsion, used to prevent sites getting to close and destabilising any partitions that they generate, as required by the last behaviour.

$$F_3(i,j) = (\mathrm{dist}(i,j) + 1)^{-\upsilon} \tag{24}$$

where $\text{dist}(i, j)$ returns the euclidean distance between sites i and j and $\upsilon \geq 1$ is a tuning factor.

As none of the forces are based directly on s_i but its negation, the first behaviour is satisfied.

The three forces are combined and scaled to produces a single force applied to the particle as

$$F(i, j) = \omega \sum_{x=1}^{3} F_x(i, j) \tag{25}$$

where ω is a final tuning parameter used to adjust the maximum force applied to any particle.

Note that, unlike traditional Newtonian physics, because of the definition of F_1, the force 'felt' by one particle in response to the presence of another particle is not necessarily equal to the force the other particle 'feels' from this one.

The change in the position of site i, ΔP_i, is the adjustment of its current location in response to the forces imposed by the other sites, in the direction of those sites,

$$\Delta P_i = \sum_{j}^{S} F(i, j)\overrightarrow{P_j P_i} \text{ where } i \neq j. \tag{26}$$

Here the notation $\overrightarrow{AB}$ is used to refer to the unit vector from A to B.

5.6.2 Neighbourhood Watch

During the implementation of the above update algorithm (and a few trial runs to confirm functionality) it was realised that a single Cell is not uniquely defined by a single Site, but also by the other Sites with whom the Cell's primary Site share edges.

The sites that belong to a Sites *Neighbourhood* set can be defined as the set of Sites whose Cells (as a subset of all points in the space) contain more than one point in common with the Cell of the primary site. The requirement of more than

one point in common is to prevent Cells that share a single vertex being included in the Neighbourhood as the excluded Site, in its current location, has no effect on the definition of the Cell. Subtle motion of the excluded Site towards the Cell's associated Site could cause it to become a neighbour depending on the motion of the other Sites that share the vertex.

$$\text{neighbour}(s_k) = \{s_j : s_j \in S. \left|\text{c}(s_k) \bigcup \text{c}(s_j)\right| > 1\}, \tag{27}$$

where $\text{c}(x)$ returns a set of points that represent the corners of the Voronoi cell of x.

An alternate definition for a Site's Neighbourhood is as the set of sites directly connected to the given Site in the Delaunay Triangulation of all the Sites. This definition is possible as the Delaunay Triangulation[2] and Voronoi Diagrams are dual graphs of each other. Figure 17 shows a sample Site/Cell and it's associated neighbourhood.

This concept is useful as it can allow for the identification of the sites that result in the definition of a cell's convex hull. i.e. All sites that are not part of $\text{neighbour}(s_k)$ could be removed from S and the result of $\text{c}(s_k)$ would remain unchanged; corollary, any changes to the sites in $\text{neighbour}(s_k)$ would result in changes to $\text{c}(s_k)$.

The main advantage of this design is the Low Saliency sites can be divided into two classifications: High Maximum Neighbourhood Saliency (HiMNS) and Low Maximum Neighbourhood Saliency (LoMNS). Here Maximum Neighbourhood Saliency (MNS) refers to the maximum saliency value of the site and all of its neighbours. The LoMNS sites, given the previous description, must have a low individual Saliency. The low individual and neighbourhood saliencies imply that the current location of this site is providing minimal predictive value for the

[2]The Delaunay Triangulation of a set of points is the "that triangulation of the convex hull of the points in the diagram in which every circumcircle of a triangle is an empty circle". Weisstein (2015)

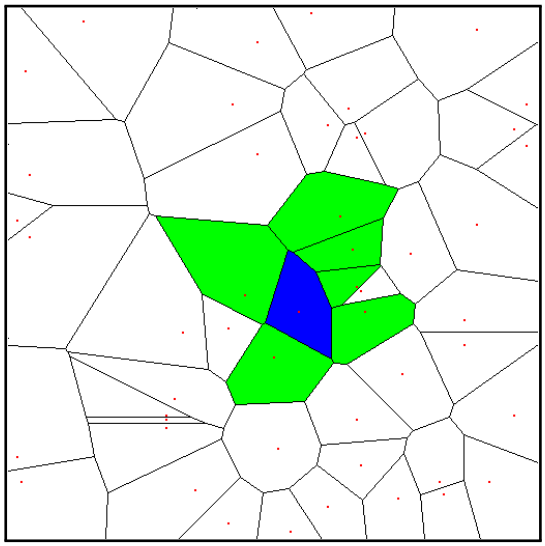


Figure 17: The figure shows the same Voronoi Diagram as Fig. 13, but shows the Voronoi Neighbourhood (green) of a selected cell (blue).

task, and make them good candidates for use to search the space for new potential regions of useful information from the source image. The term HiMNS is used explicitly to describe LS sites that are neighbours to a HS. Though HS sites technically belong to this category, the moniker High Saliency is more than sufficient and typically more apt. The HiMNS sites aid in the definition of high value cells and could act as buffers between the stable HS sites and the unstable LoMNS sites. Moving HiMNS sites towards a HS site performs the equivalent of focusing in on any object or feature that may be represented in that region of the image.

To implement this, the forces summation function, Eq. 25 was adjusted as follows:

$$F(i,j) = \omega(1 - neighbour(s_i)) \sum_{x=1}^{3} F_x(i,j), \tag{28}$$

where n_i is the neighbourhood saliency of site i.

An additional optimisation to computation time is that the force calculations can be computed only between neighbours. This reduces the computation (on average) only by a constant factor, but can be quite significant reduction if the VSP is defined by a large number of sites.

5.6.3 Powered Neighbourhoods

The final update algorithm is an alternate particle simulation with a simplified attraction/repulsion forces. The linear interpolation between these two forces, by the Neighbourhood Saliency is then proportioned by the sites Salience and inversely by the distance and biased by a proximity force:

$$F_a(i,j) = {s_j}^{\theta} \tag{29}$$

$$F_r(i,j) = (1 - s_j)^{\theta} \tag{30}$$

$$F_p(i,j) = (\mathrm{dist}(i,j) + 1)^{-\upsilon} \tag{31}$$

$$F(i,j) = \omega \left((1 - s_i) \frac{n_i F_a(i,j) + (1 - neighbour(s_i)) F_r(i,j)}{(\mathrm{dist}(i,j) + 1)^2} - F_p(i,j) \right) \tag{32}$$

where θ, υ and ω perform similar functions as tuning parameters to their equivalents in the previous two update algorithms.

This method is designed to also conform to the behaviours described in section 5.6.1.

5.7 Testing Site Update

To examine the addition of the update portion of the VSP algorithm, the previous 'Object Presence Detection' was re-run. Based on the performance of the previous test, the down-sampling portion of VSP, only the precomputed variant of Inverse Distance-Squared, initialised by the Rectilinear method, will be used in this test.

For each of the three potential update methods a variety of parameters will be trialled. The domain of these parameters is presented in Table 5.

The update procedure depends upon learning previously done by the ANN. To prevent issues arising from updating based on the randomised initial state of the ANNs weights and to allow sufficient time between updates for the ANN to learn

Table 5: Parameter domains for the Site Update tests.

Method	Parameter	Domain
Update 1 (Sec. 5.6.1)	υ	10
	ω	$2^{-1}, 2^{-2} \ldots 2^{-9}$
Update 2 (Sec. 5.6.2)	ω	
	υ	10
Update 3 (Sec. 5.6.3)	υ	10
	θ	0.2, 0.5, 0.7
	ω	0.5, 0.7

in response, site updates are not applied until the third epoch and then every odd epoch thereafter.

5.7.1 Results

To prevent the resultant graphs from becoming unreadable, each of the presented updated methods (and the control groups) will be presented in separate figures. Each graph will present the mean accuracy, the positive and negative standard deviations and anomalous individuals (identified as beyond twice the inter-quartile range) for each epoch. Figures 18, 19, 20, 21 present the results for Update Methods 1, 2, 3 and the control group, respectively.

For Update Algorithm 1, Fig. 18, soon after the update process begin to apply changes to the underlying sites, the accuracy drops off rapidly (the first updated occurs between epochs 3 and 4), only to stabilise around 33% (no better than chance).The drop-off itself was to be expected from the design, but the magnitude is far greater than expected and the inability to recover is disappointing. The apparent determining factor in the rate of drop-off is the update algorithms speed factor, ω. As ω tends towards 0, (the maximum effect of the update process on the location of the sites) the learning portion of the ANN is able to learn to solve the task. Figure 22 shows an initial and final VSP preprocessor and was chosen to provide an insight into the forms of failure for this algorithm. The specific instance is for Update Algorithm 3, but all failures of the Update process tended

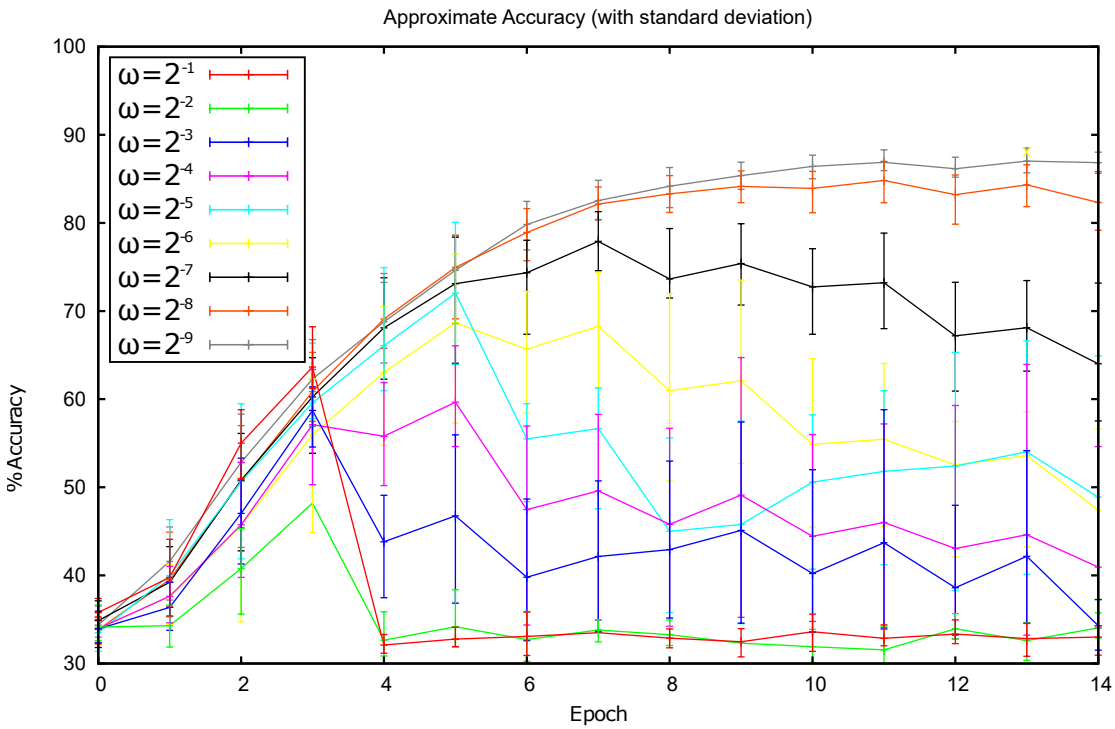


Figure 18: VSP using Update Algorithm 1

towards similar results.

Update Algorithm 2, Fig. 19, thought still worse than the controls, faired better then the previous algorithm. The lower asymptote demonstrates that this method allows for a lower maximum accuracy. The increased standard deviation coupled with the reduced performance from the two largest values for ω bring scepticism to the notion of stability for this update method.

Update Algorithm 3, Fig. 20, at first glance appears to have performed worse than algorithm 1. In the previous two algorithms, it has been shown that large values for ω can make the algorithms unstable. Due to the interplay with a second parameter, θ, and the long runtime required to perform the test only two values for ω were tested. Further evaluations with smaller values for ω were not under-taken for the reasons that follow.

The speed factor ω dictates how significant a change the the update algorithms can apply to the current state of sites. When the value of ω approaches 0, the

update algorithms are having negligable effect on the sites locations. Given this and the relationship between the rate of drop in accuracy and the ω implies that the rectilinear layout is the primary contributor to the predictive value of the preprocessor in the early stages of learning, not the VSP algorithm. As the Voronoi Diagram underlying the VSP departs from the preferred rectilinear grid, the ability of the ANN drop. Faster departures result in greater drop in predictive quality per epoch.

5.8 Conclusion

This section will first present a general discussion and overview of the results and other outputs of the VSP preprocessors on Test Domain 1. This will be followed up with the research questions (Section 5.3) presented earlier in this chapter and their responses.

As pointed out in the previous section, as the value ω tends towards 0, the negative effect of the VSP algorithm decreases. This is a result that <u>any</u> effect the update algorithms have is proportional to the value of ω. When testing the computational time of the downsample, this was found to be 3 orers of magnitude slower than the traditional methods. Pre-computing large portions of the preprocessor (trading computational time for memory capacity), this improved the runtime, but only to 2 orders of magnitude slower than the traditional methods.

The gathered results show that the VSP algorithms introduced in this chapter are unsuited to the task presented. It is believed that the rectilinear nature of the traditional methods Bilinear and Bicubic provide a more robust preprocessor than was previously expected. In response to this, it is recommended to the reader that, unless significant advancements and alterations are made, that the presented VSP algorithms are to be considered undeserving of future investigation.

Can a Voronoi Diagram be used as a sound basis to partition a 2D

> space for the purpose of image preprocessing to usable features for an ANN?

The VSP preprocessor was successfully used as a preprocessor for an ANN. The algorithm was able to present useful features to ANN, which in turn was able to learn to solve the required task. The flexibility of the sites to produce convex feature partitions could be useful in some niche circumstances. As the Update process was unsuccessful, the generation of these sites would be a manual task at current. Coupled with the three orders of magnitude decrease in computational speed, make the algorithm usable as opposed to useful.

> Can such a preprocessor allow the associated ANN to achieve an improved measure of accuracy at a given task?

The presented algorithms fail to produce improved or comparable results to the traditional methods.

> Do the increased computational overheads in the preprocessor allow for a greater reduction in computation in the ANN?

The presented algorithms fail to reduce overall computational time when compared to the traditional methods.

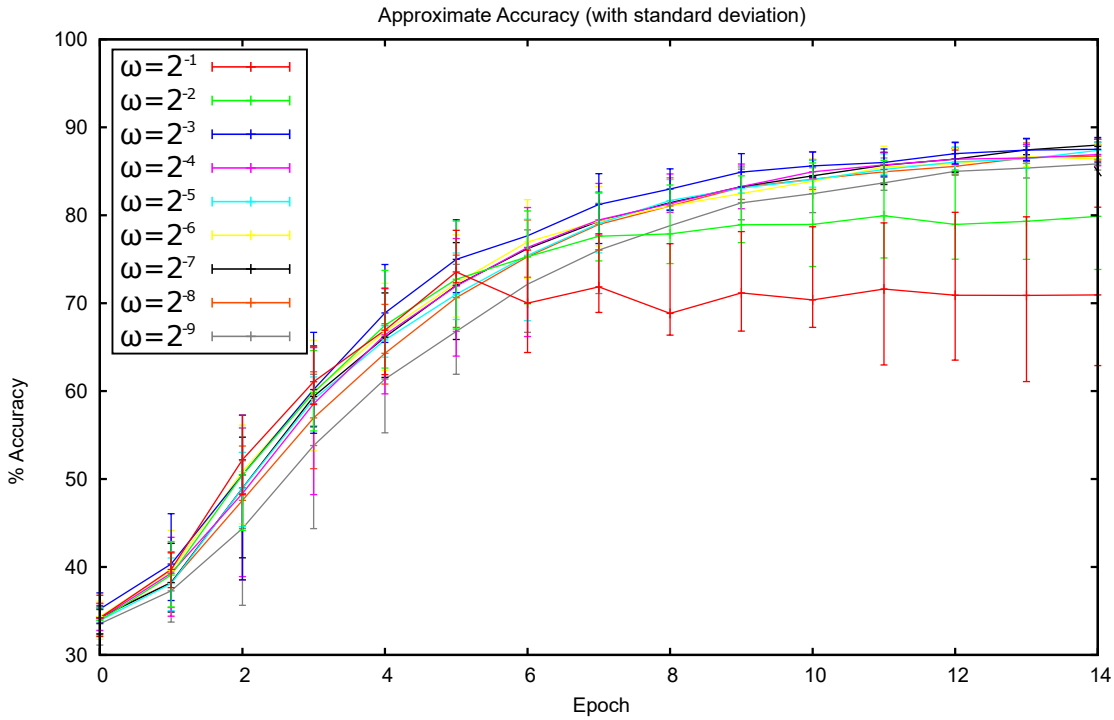


Figure 19: VSP using Update Algorithm 2

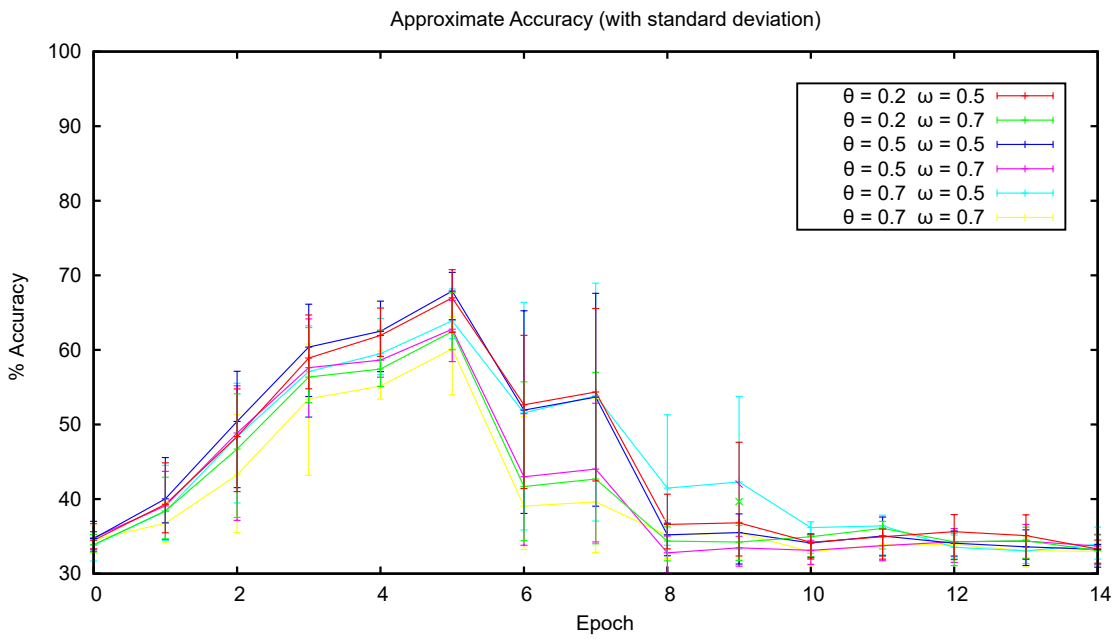


Figure 20: VSP using Update Algorithm 3

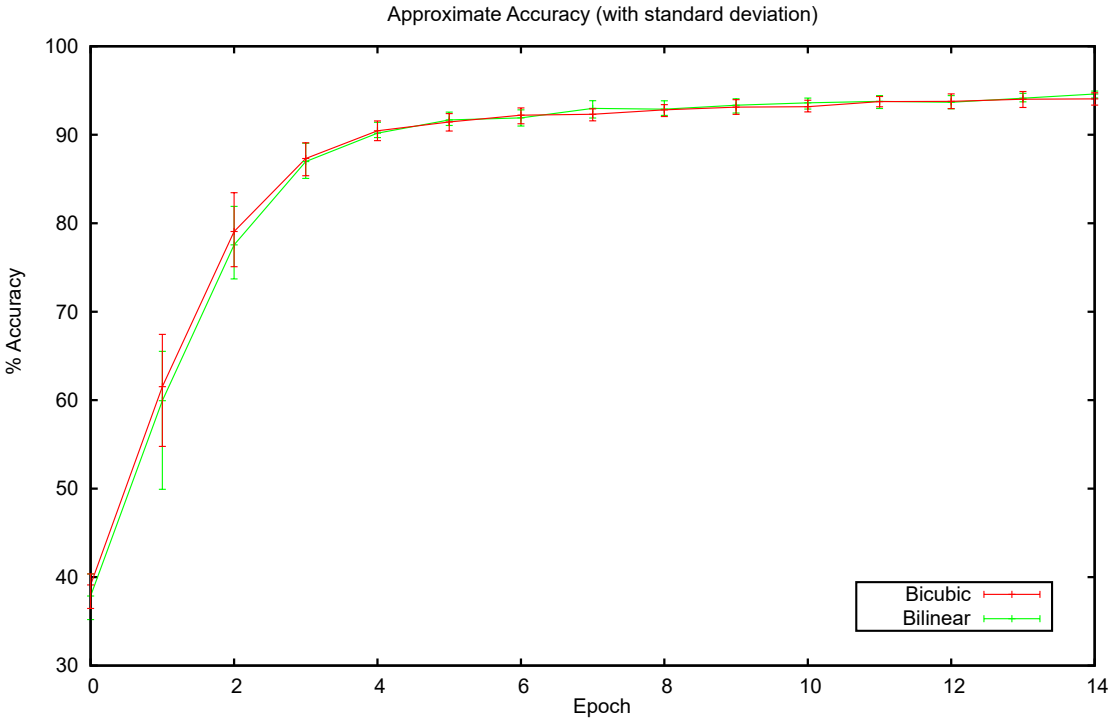


Figure 21: Control Group: Traditional Methods

Figure 22: An ANN system (using Update Algorithm 3) early in its training (left) and at completion (right). In this sample, The system failed to stabilise until all the Voronoi sites clustered around point (0,0).

Chapter 6

Regional Downsampling Algorithms

The concept of *Regional Downsampling Algorithms* (RDAs) was formed in response to the failings of VSP. As was noted in the previous chapter, the model underlying VSP requires significant computation in comparison to the simpler methods of Bilinear and Bicubic interpolation and performed worse than the control groups. In review of the design, the reduced decrease in accuracy from the rectilinear VSP instances presented an idea, maybe the rectilinear resampling of an image was better for a computational algorithm than had been previously expected. From this, the RDA model was designed to leverage the advantages of the rectilinear regularity of down-sampling and the many years of refinement that have been applied to the standard implementations.

First, the additional term *Region* will be defined in the context of an RDA; followed by how an RDA, using a collection of unique regions, can be used as an image preprocessor and how the collection of Regions can be obtained.

6.1 Terminology: Region

The term *Region* is used to describe an axis-aligned rectangular subsection of a source image, used as single pixel sampling window as part of RDA algorithms. The Region is described by two 2-dimensional values. The size value is used to describe the number regions required to cover the entire source in terms of tiled columns and rows; hence inferring the size of each region. The second provides the offset from the top-left corner of this regional grid to the individual region.

This reduces the complexity of implementation as

- Each Region can be uniquely defined, and each definition refers to a single Region.
- All values of each Region instance can be stored as integral types, negating Precision-Errors from floating-point numbers.
- Bounds checking can be simply to checking if $0 \leq pos_x < res_x$ (similarly for the y axis), as opposed to bounds checking with floating-point checks with more tedious implementation specific edge-casing.
- Base-line implementations can, at the point of down-sampling, resize source Image to the specified resolution and then sample the pixel at the given location for the Region's top-left corner.

Fig.23 provides a simple visual representation of how Regions are defined, presenting a variety of Regions with different values for all four parameters. For the sake of the reader, these Regions do not overlap, though there is no constraint that mandates this for groups of Regions. The notation used to in describing a Region will always use (P_x, P_y) 2-tuple to refer to it's location and $R_x \times R_y$ to refer to its resolution.

The term *Region Set* is used as shorthand for 'a stable ordered set of regions'. If the collection of regions contained duplicates, this would be providing redundant

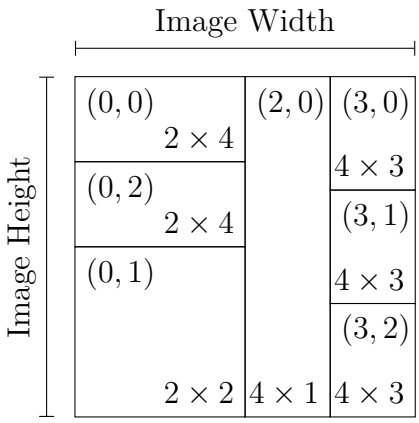


Figure 23: An example layout of 7 regions at a variety of resolutions (bottom left corner of each region) and different positions (top left corner). In this example none of the regions overlap and all the space is covered by a region for clarity and simplicity; these are not restriction imposed by regions themselves.

information or add undesirable bias to the duplicated region. Additionally, as the intended use for the related algorithms is as input sources for ANN, the order of features presented must remain stable. (i.e. Using hashes is a common practise in the implementation sets as these structures are efficient at identifying duplicates but are liable to re-order the presentation of their content in the event of any changes in contents. For this reason, solely using hash based sets implementations is insufficient.)

6.2 RDA as a Preprocessor

A RDA is a novel image preprocessing algorithm, designed to reduce the dimensionality of an image using a sequence of regions (as defined above). The specific mechanism of obtaining the sequence of regions is dependent on the selection method, which will be discussed in the following section. The feature set output by the algorithm is equivalent to a sequence of pixels, equal in length to the number of regions. In a base-line implementation of a RDA, for each region, R, the

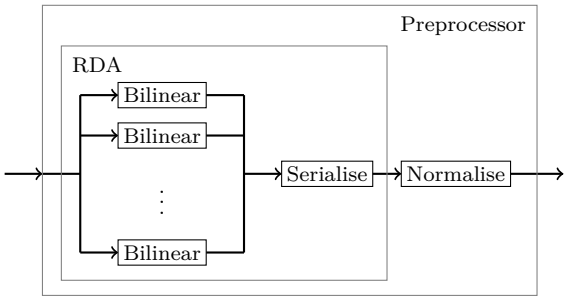


Figure 24: Representation of the implementation of a preprocessor using an internal RDA. This diagram also displays the internal component topology and data flow through the RDA. Each Bilinear block computes the equivalent output of the Bilinear Interpolation algorithm for it the single pixel represented by the associated Region.

image is resized to $R_h \times R_v$ by simple rectilinear interpolation method[1] and then filtered to return just the pixel at coordinate (R_x, R_y) for use as the feature value for the region. Figure 24 shows the generic structure of a RDA and how it would be incorporated into a preprocessor.

6.2.1 Comparison to CNNs

Section 2.1.2 provided an overview of Convolutional Neural Networks (CNNs), how they function and their advantages and disadvantages as shown by the literature. The design of the Convolution layer (specifically the receptive fields) of a CNN share a similarity with the Regions of RDAs. The contrast between them is in what part is "learned" by the system and which part defined by the developer. In a CNN, the area of a receptive field is specified and construction and the kernel applied to each receptive field is refined over many training sessions. Conversely, the area a region refers to is extracted from the accumulation of many training sessions (Bespoke) or refined over many training sessions (Adaptive), whilst the down-sampling function (equivalent to the kernel) is defined by the developer.

[1]Bilinear interpolation is used in these implementations due to the reduced processing time and comparable quality to other interpolation methods at low resolutions.

6.3 Research Question

To explore RDAs, the following research questions are proposed:

1. Do the Region Selection Methods presented allow for the construction of viable RDA Preprocessors (preprocessors that allow an ANN to learn its intended task)?

2. Can such a preprocessor allow the associated ANN to achieve an improved measure of accuracy at a given task?

3. Do the increased computational overheads in the preprocessor allow for a greater reduction in computation in the ANN?

To answer these questions, a similar work flow from that used in chapter 5 will be used. First the preprocessor, constructed from each selection algorithm, will be tested against Test Domain 1: Object Presence Detection, Section 3.2. Upon demonstrating the ability to learn the first task, the tests will progress to Test Domain 2: Baseline Face Detection, 3.3.

6.4 Region Selection

Region Selection is the process by which the Region set, used by a RDA implementation, is generated. This section will present three different algorithms for the generation of these Region sets. Their use constitutes the production of a Bespoke Preprocessor. The three algorithms function similarly, only differing in how restrictive the original selection pool is. Each algorithm is expected to produces RDAs with Region sets of differing sizes. Assuming multiple algorithms are successful, this allows for an evaluation of the trade off in computation time costs and accuracy improvements per region.

6.4.1 Setup: Saliency Computation

As stated above, the three following algorithms share a lot of their functionality. The setup to all three is one of these similarities. All three algorithms require knowledge of the Saliency of various potential Regions for selection. *Saliency Computation* (Alg. 1) is used as a brute-force implementation that tests all potential Regions, in a given range of resolutions, and provides that information to the three selection algorithms.

Algorithm 1 Saliency Computation: This function pre-computes the per pixel salience over a range of resolutions or a given training regime.

Require: res_{min} & res_{max}, the range (inclusive) of the resolutions to test.
Require: $train(p, n)$, a function that will perform a very short training regime on a given ANN, n, with the provided preprocessor, p, and return the accuracy against a test set.

```
1:  function salComp(res_min, res_max, train)
2:      res_range ← res_max − res_min + 1
3:      sal ← Uninitialised 2D Array, sized res_range × res_range
4:      for res_h = res_min to res_max do
5:          for res_v = res_min to res_max do
6:              ρ ← Uninitialised 1D Array, sized repetitions
7:              pp ← Bilinear Preprocessor (res_h, res_v)
8:              for x = 0 to (repetitions − 1) do
9:                  n ← Randomly Initialised ANN
10:                 acc ← train(pp, n)
11:                 flatNetSal ← Saliency (n)
12:                 matNetSal ← Uninitialised 2D Array, sized res_h × res_v
13:                 for i = 0 to (res_h − 1) do
14:                     for j = 0 to (res_v − 1) do
15:                         matNetSal [i, j] ← flatNetSal [i + j · res_h]
16:                 ρ [x] ← {acc : acc; mat : matNetSal}
17:             sal [res_h, res_v] ← average (ρ)
18:     return sal                ▷ A 2D array of {accuracy; 2D salience matrices}
19: function Saliency(n)
20:                 ▷ Implementation of the Saliency metric for ANN n (Sec. 5.2).
21:         ▷ Returns an array of saliencies, equal in length to the input layer of n.
```

6.4.2 Saliency Heat Map Selection

Saliency Heat Map Selection (SHM Selection) starts by takes the image matrices provided by the Saliency Computation stage and resizes them via nearest-neighbour interpolation, such that their dimensions are $2 \times res_{max} \times res_{min}$. The resultant matrices are then averaged together, weighted by their accuracies, generating a single Saliency Heat Map. Fig. 25 shows an example of a heat map generated by the SHM algorithm. The heat map is resized to each allowed resolution, where each pixel in the heat map represents an estimated saliency of the potential region that would have the same coverage as that pixel. From these, each pixel is sampled, and the single pixel over all the resolutions with the highest estimated saliency is chosen. A Region is generated to cover the same region (using the pixels location and the resolution of the matrix from which it is draw) and added to the Region set. The equivalent area of the heat map is then clear out to prevent re-selection. This process is repeated multiple times, until there are no remaining potential Regions with estimated saliencies above a specified threshold. For the implementation used in the later tests, a selection threshold of 0.65 was found to be sufficient by informal experimentation.

Algorithm 4 show this is pseudo-code, using Algo. 2 and 3.

6.4.3 Restricted Saliency Heat Map Selection

A criticism with SHM selection comes from fact that the heat map is generated by a weighted average of matrices, with the weight based on the accuracy of the ANN against a test set, and the space of potential Regions being that of all Regions within the allowed resolution range. This means that if a potential resolution results in a untrained ANN (one that has a accuracy no better than chance), then candidate Regions will still be put forward for selection and can be successful depending on the flattened heat map.

This new method works in the same manner as SHM Selection, but applies

Algorithm 2 Saliency to SHM: This support function is used to simplify the returned value from Alg. 1 to a single matrix. This is obtained by first computing the weighted average of all the matrices, weighted by their associated accuracies. The flattened result is up-sampled by Nearest-Neighbour the be $res_{max} \cdot res_{min} \cdot 2$ in both dimensions and is finally normalised to the range $0 \ldots 1$ before being returned. The provided pseudo-code implementation combines the up-sampling and weighted average stages to reduce the memory footprint.

Require: sal, the result of the saliency computation.
Require: res_{min} & res_{max}, the range (inclusive) of the resolutions used.

```
function matsToSHM(sal, res_min, res_max)
    res_range ← res_max − res_min + 1
    res_size ← res_max · res_min · 2
    SHM ← 2D Array of numbers, sized res_size × res_size
    for each (x, y) ∈ SHM do
        sum ← 0
        weight ← 0
        for each (i, j) ∈ sal do
                ▷ A combined Nearest-Neighbour resize and weighted averaging
            x_nn ← x(i + res_min)/res_size
            y_nn ← y(j + res_min)/res_size
            sum ← sum + sal[i, j]_mat[x_nn, y_nn] · sal[i, j]_acc
            weight ← weight + sal[i, j]_acc
        SHM[x][y] ← sum/weight
                                    ▷ Normalise the SHM before returning
    return (SHM − min(SHM))/(max(SHM) − min(SHM))
```

Algorithm 3 Saliency Estimation: This support function takes the flattened SHM and returns the estimated salience of a given region.

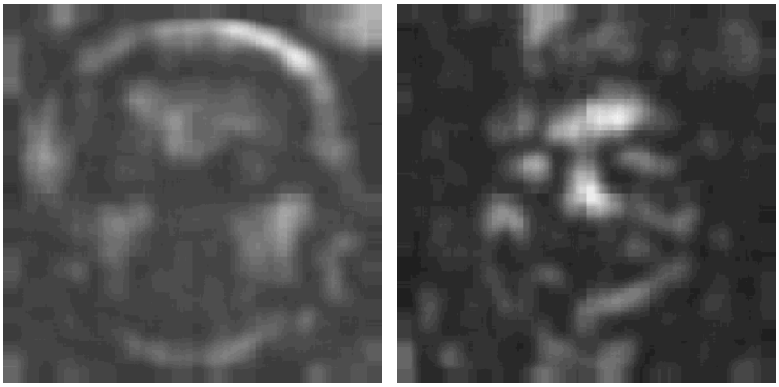
Require: x, y, the location of the prospective region (P_x, P_y).
Require: h, v, the resolution of the prospective region (R_x, R_y).
Require: sal, the result of the saliency computation.

```
function estimateSaliency(x, y, h, v, SHM)
    resampled ← Downsample SHM to the size h×v by Bilinear Interpolation
    return resampled[x, y]
```

Algorithm 4 SHM Selection: Performs a region selection based on the computed salience of multiple resolutions using an heat-map approach.

Require: sal, the result of the saliency computation.
Require: $threshold$, the selection threshold.
Require: res_{min} & res_{max}, the range (inclusive) of the resolutions used.

```
1:  function SHMSELECTION(sal, threshold, res_min, res_max)
2:      SHM ← matsToSHM(sal, res_min, res_max)
3:      potentialRegions ← Empty collection
4:      for each (h, v) ∈ (res_min, res_min) ... (res_max, res_max) do
5:          for each (x, y) ∈ (0, 0) ... (h, v) do
6:              s ← estimateSalience(x, y, h, v, SHM)
7:              if s ≥ threshold then
8:                  Push potentialRegions ⇐ {x : x; y : y; h : h; v : v; salience : s}
9:                  ▷ Properties x, y, h and v encode P_x, P_y, R_x and R_y of a region
10:     selectedRegions ← Empty collection
11:     while potentialRegions is not empty do
12:         best ← remove potential region with highest salience property
13:         Mask out area of SHM covered by best region
14:                                         ▷ Resize best using Nearest-Neighbour
15:         for each pr ∈ potentialRegions do
16:             s ← estimateSalience(pr_x, pr_y, pr_h, pr_v, SHM)
17:             if s ≥ threshold then
18:                 pr_salience ← s
19:             else
20:                 Remove pr from potentialRegions
21:     return selectedRegions
```



(a) Caltech (b) CMU

Figure 25: The SHM generated for the baseline face finding task (Section 3.3). Both images have been normalised, with the Caltech image (25a) originally ranging 0.55 to 0.67 and the CMU image (25b) ranging 0.59 to 0.69. For the Caltech data set, the algorithm has focused on the perimeter of the face, cheeks, forehead and an anomalous region in the top right corner. Against the CMU data set the algorithm has highlighted the forehead, eyes, noes and jawline.

the new additional criteria to pixel selection. Prior to selecting pixels by highest predicted saliency, the pixels are pre-filtered so that only pixels from resolutions that had positive accuracy scores from the Validation tests (i.e. the ANNs that were used to generate the saliencies managed to learn the task at hand, even to a minor degree) can be chosen. The modifications can be seen by the additions of lines 5 and 6 in Algo. 5.

6.4.4 Pruned-Bilinear Selection

This final selection method is a further restriction on SHM. Pruned-Bilinear is named for the fact that the resultant Region set only contain Regions taken from the same resolution. The resolution sampled is the single resolution with the highest computed accuracy. Once selected, the preprocessor can be implemented as a single call the to Bilinear Interpolation algorithm, and the returned image sampled for the specified pixels from the Regions location.

Due to this extreme restriction, a large portion of the selection algorithm and

Algorithm 5 Restricted-SHM Selection: Performs a region selection based on the computed salience of multiple resolutions using an heat-map approach, restricted to resolutions that have been shown to allow for above random accuracy of trained ANNs.

Require: sal, the result of the saliency computation.
Require: $threshold$, the selection threshold.
Require: res_{min} & res_{max}, the range (inclusive) of the resolutions used.

```
1:  function RshmSelection(sal, threshold, res_min, res_max)
2:      SHM ← matsToSHM(sal, res_min, res_max)
3:      potentialRegions ← Empty collection
4:      for each (h, v) ∈ (res_min, res_min) ... (res_max, res_max) do
5:          if sal[h − res_min, v − res_min]_acc ≤ 0 then
6:              skip resolution (h, v)
7:          for each (x, y) ∈ (0, 0) ... (h, v) do
8:              s ← estimateSalience(x, y, h, v, SHM)
9:              if s ≥ threshold then
10:                 Push potentialRegions ⇐ {x : x; y : y; h : h; v : v; salience : s}
11:                 ▷ Properties x, y, h and v encode P_x, P_y, R_x and R_y of a region
12:     selectedRegions ← Empty collection
13:     while potentialRegions is not empty do
14:         best ← remove potential region with highest salience property
15:         Mask out area of SHM covered by best region
16:                                         ▷ Resize best using Nearest-Neighbour
17:         for each pr ∈ potentialRegions do
18:             s ← estimateSalience(pr_x, pr_y, pr_h, pr_v, SHM)
19:             if s ≥ threshold then
20:                 pr_salience ← s
21:             else
22:                 Remove pr from potentialRegions
23:     return selectedRegions
```

be simplified, see Algo. 6 for a pseudo-code implementation.

Algorithm 6 Pruned-Bilinear Selection: Performs a region selection based on the computed salience of multiple resolutions, restricted to all the single resolution with the highest above-random accuracy for trained ANNs

Require: sal, the result of the saliency computation.
Require: $threshold$, the selection threshold.
Require: res_{min} & res_{max}, the range (inclusive) of the resolutions used.

```
1: function PRUNEDBILINEARSELECTION(sal, threshold, res_min, res_max)
2:     (i, j) ← Such that sal[i, j]_acc is maximal
3:     mat ← (sal[i, j]_mat − res_min)/(res_max − res_min)        ▷ Normalise
4:     selectedRegions ← Empty collection
5:     for each (x, y) ∈ mat do
6:         if mat[x, y] ≥ threshold then
7:             Push selectedRegions ⇐ {x : x; y : y; h : i + res_min; v : j + res_min}
8:     return selectedRegions
```

The type of results provided by Pruned-Bilinear Selection show similarity to the results from the MIPs method presented by Lew and Huijsmans (1996) (see section 2.5.1 for an overview of their model) with a variation on the pixel/region quality metric (Kullback relative information as opposed to Salience) and selection qualification (Markov random field approximation verses Pruned-Bilinear selection).

6.5 Optimising for Runtime

The direct usage of the Region Set as returned by the selection method for the RDA would perform poorly (in terms of computation time) compared to Bilinear Interpolation preprocessor. This is due, primarily, to the set-up and tear-down costs associated with the internal calls to the underlying Bilinear Interpolator for each Region. This costs can be minimised by grouping compatible Regions and using only a single call to the Bilinear algorithm and filtering out the desired pixels to be presented as the intended features.

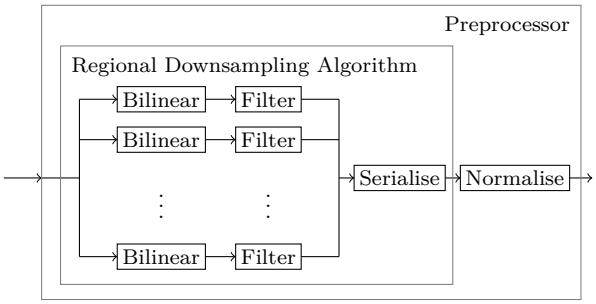


Figure 26: A diagram of an RDA constructed using the Region Set optimiser for reduced computational overheads.

A simple variant of this is used in the implementation of RDAs used in this thesis. In this version, all Regions of the same resolution are grouped together and a filter. A more intelligent version would take into consideration the distance between regions of the same resolution to identify clusters of regions, with the goal of minimising the computation of redundant pixels. A preliminary investigation into this idea was performed, but due to a limitation of time, a successful segmenting method was not found.

6.6 Inter-comparisons of RDAs

When producing RDAs via selection, it may be beneficial to compare how similar two different RDAs are, such as for identifying similar families of RDAs for further study. Two metrics are provided here, RDA Similarity and RDA Likeness.

6.6.1 Similarity

A simple description, anthropomorphising the presence of a Region as providing 'attention' to the section of the source image at a specific granularity, then Similarity and Likeness can be described as heuristic measures of convergence in the attention of the two preprocessors being compared. Likeness approximates for space in which the attention is provided, while Similarity approximates for both

space and granularity of this attention.

To compute the Similarity of two RDAs, the ratio of intersection area to union area is computed for each of the grouping of regions of the same resolution against the equivalent collection of regions for the opposed RDAs. Resolutions that do not have an equivalent collection in the opposing RDA can be ignored as the intersection between them will equal 0. The resultant Similarity is the average of these ratios.

Algorithm 7 Computes the Similarity between two RDAs, A and B. In this function, r_h, r_v, r_x and r_y refer to the horizontal resolution, vertical resolution, X-offset and Y-offset properties of the Region r.

Require: A, the set of regions from the first RDA,
Require: B, the set of regions from the second RDA.

```
1:  function SIMILARITY(A, B)
2:      group_A ← group(A, λr : (r_h, r_v))   ▷ Group all regions by their resolutions
3:      group_B ← group(B, λr : (r_h, r_v))
4:      sim ← 0
5:      for each res ∈ keys(group_A) do
6:          if res ∈ keys(group_B) then
7:              regs_A ← group_A(res)
8:              regs_B ← group_B(res)
9:              locs_A ← map(regs_A, λr : (r_x, r_y))
10:             locs_B ← map(regs_A, λr : (r_x, r_y))
11:             I ← count(intersectlocs_A, locs_B)
12:             U ← count(regs_A) + count(regs_B) − I
13:             sim ← sim + (I/U)
        return sim
```

6.6.2 Likeness

Likeness applies the ratio of intersection to union between the union of all regions within each RDA. This can easily be approximated to N decimal places using a 1-bit square matrix of order $R \times 10^N$ where R is the maximum resolution used in the generation of either RDA. A value of 0 will be returned if the two preprocessors are distinct, and a value of 1 if the two are equal OR one is completely contained

by the other.

Similarity is most suitable when comparing highly dense[2] RDAs generated by the same (or possible highly similar) RDA Selection methods. Likeness, on the other hand, is appropriate for comparisons between sparse[2] RDAs and can also be useful in comparisons across RDA selection methods; in this case care needs to be taken as a likeness between two RDAs produced by the same seection method cannot be considered comparable to a likeness between two RDAs of differing origins.

6.7 Situating

The use of the Selection Method makes the RDA model a member of the Bespoke preprocessors group. A pre-trial is performed to determine the state of the region set, as generated by the Selection method. The RDA is then used for training of the ANN, during which time the region set remains unchanged.

6.8 Results

6.8.1 Test Domain 1

When applied to the challenge presented in Test Domain 1, it can be seen from table 6 that the three variants RDA perform noticeably better than the traditional Bilinear and Bicubic preprocessors in terms of both Cohen's kappa and computation time. This sufficiently demonstrates that the algorithms functions as intended and can now progress to Test Domain 2 for statistical analysis.

[2]Density is used with respect to the variety of unique resolutions in use by the RDAs

Algorithm 8 Computes the Likeness between two RDAs, A and B. This version uses the assumption that a Region is always 1×1 at the given resolution. In this function, reg_h, reg_v, reg_x and reg_y refer to the horizontal resolution, vertical resolution, X-offset and Y-offset properties of the Region reg.

Require: A, the set of regions from the first RDA,
Require: B, the set of regions from the second RDA,
Require: $detail$, the level of detail for this approximation. A value of $R_{max} \cdot 10^N$ gives a precision of $\approx N$ decimal places.

```
1:  function LIKENESS(A, B, detail)
2:      mat ← 1-bit details × details matrix with all entries zeroed.
3:      union ← 0
4:      intersect ← 0
5:      for each reg ∈ A do
6:          x_min ← ⌊reg_x × (details/reg_h)⌋
7:          x_max ← ⌊(reg_x + 1) × (details/reg_h)⌋
8:          for x = x_min → x_max do
9:              y_min ← ⌊reg_y × (details/reg_v)⌋
10:             y_max ← ⌊(reg_y + 1) × (details/reg_v)⌋
11:             for y = y_min → y_max do
12:                 mat[x, y] ← 1
13:                 union ← union + 1
14:     for each reg ∈ B do
15:         x_min ← ⌊reg_x × (details/reg_h)⌋
16:         x_max ← ⌊(reg_x + 1) × (details/reg_h)⌋
17:         for x = x_min → x_max do
18:             y_min ← ⌊reg_y × (details/reg_v)⌋
19:             y_max ← ⌊(reg_y + 1) × (details/reg_v)⌋
20:             for y = y_min → y_max do
21:                 if mat[x, y] = 1 then
22:                     intersect ← intersect + 1
23:                 else
24:                     union ← union + 1
        return intersect/union
```

Table 6: Aggregate results for Test Domain 1.

Preprocessor	**Features**	**Cohen's kappa**		**Computation Time**	
		Average	**Standard Deviation**	**Average**	**Standard Deviation**
Bilinear (6×5)	90	0.967	0.0122	2.2ms	1.31ms
Bicubic (6×5)	90	0.959	0.0178	2.7ms	1.55ms
RDA + SHM	9	0.970	0.0069	2.1ms	1.28ms
RDA + Restricted-SHM	9	0.951	0.0117	2.4ms	1.42ms
RDA + Pruned Bilinear	6	0.970	0.0070	1.6ms	0.94ms

Table 7: Aggregate results for the tests against the Caltech Data Set.

Preprocessor	**Features**	**AUC**		**Computation Time**	
		Average	**Standard Deviation**	**Average**	**Standard Deviation**
Bilinear (24×22)	1584	0.943	0.0857	48.3ms	8.01ms
Bicubic (24×22)	1584	0.966	0.0654	60.0ms	8.74ms
RDA + SHM	36	0.977	0.0057	133.9ms	34.69ms
RDA + Restricted-SHM	39	0.981	0.0024	153.7ms	43.18ms
RDA + Pruned Bilinear	42	0.980	0.0113	27.8ms	5.32ms

6.8.2 Test Domain 2

Tables 7 and 9 show the results for the Face detection task against the Caltech and CMU-130 data sets, respectively.

Caltech Dataset

Using the Caltech data-set, there was a statistically significant difference in predictive accuracy depending on the preprocessor used, $\chi^2(4) = 49.739, p = 4 \times 10^{-10}$. Post hoc analysis using Wilcoxon signed-rank tests were performed, with Bonferroni correction applied.

Table 8: Post hoc Wilcoxon Signed-Rank tests for RDA Accuracy on the Caltech Data set

	Pruned-Bilinear - Bicubic	Pruned-Bilinear - SHM	Pruned-Bilinear - Bilinear	Pruned-Bilinear - Restricted-SHM	Bicubic - SHM
z	0.850	4.124	3.830	2.309	2.713
Asymp. Sig. (2-tailed)	0.39513	0.00004 (sig)	0.00013 (sig)	0.02096	0.00666

	Bicubic - Bilinear	Bicubic - Restricted-SHM	SHM - Bilinear	SHM - Restricted-SHM	Bilinear - Restricted-SHM
z	3.071	1.687	3.201	6.130	3.420
Asymp. Sig. (2-tailed)	0.00213 (sig)	0.09151	0.00137 (sig)	8.8×10^{-10} (sig)	0.00063 (sig)

Table 9: Aggregate results for the tests against the CMU Data Set.

Preprocessor	**Features**	**AUC**		**Computation Time**	
		Average	**Standard Deviation**	**Average**	**Standard Deviation**
Bilinear (25×19)	1425	0.571	0.1414	47.3ms	9.86ms
Bicubic (25×19)	1425	0.469	0.1668	58.2ms	10.67ms
RDA + SHM	48	0.623	0.0769	174.4ms	71.79ms
RDA + Restricted-SHM	48	0.595	0.1119	173.4ms	72.84ms
RDA + Pruned Bilinear	21	0.596	0.0178	18.3ms	6.47ms

Table 8 suggests that the following hierarchy describes the expected accuracy of the preprocessors: Pruned-Bilinear (98.1%) $\approx$ Restricted-SHM (98.1%) $>$ SHM (97.7%) $>$ Bilinear (94.4%), with Bicubic (96.6%) difficult to place, possibly due to the larger variance.

For the comparison of computation times, the results are much clearer than anticipated, there was a statistically significant difference depending on the preprocessor used, $\chi^2(4) = 67406.6, p = 0$. Table 10 show the post hoc analysis (as before) for the computation times.

The clarity of these significance tests give the hierarchical ordering (with overall average times) of: Pruned-Bilinear (27.8ms) $<$ Bilinear (48.3ms) $<$ Bicubic (60.0ms) $<$ SHM (133.9ms) $<$ Restricted-SHM (153.7ms).

CMU Dataset

Using the CMU data-set, there was a statistically significant difference in predictive accuracy depending on the preprocessor used, $\chi^2(4) = 40.757, p = 3 \times 10^{-8}$. Post hoc analysis using Wilcoxon signed-rank tests were performed, with Bonferroni correction applied.

Table 8 suggests that the following hierarchy describes the expected accuracy of the preprocessors: SHM (62.3%) $>$ Pruned-Bilinear (59.6%) $\approx$ Restricted-SHM

Table 10: Post hoc Wilcoxon Signed-Rank tests for RDA Overall Computation Time on the Caltech Data set

	Pruned-Bilinear - Bicubic	Pruned-Bilinear - SHM	Pruned-Bilinear - Bilinear	Pruned-Bilinear - Restricted-SHM	Bicubic - SHM
z	122.5	122.5	122.5	122.5	122.4
Asymp. Sig. (2-tailed)	0 (sig)	0 (sig)	0 (sig)	0 (sig)	0 (sig)

	Bicubic - Bilinear	Bicubic - Restricted-SHM	SHM - Bilinear	SHM - Restricted-SHM	Bilinear - Restricted-SHM
z	122.4	122.4	122.5	122.2	122.5
Asymp. Sig. (2-tailed)	0 (sig)	0 (sig)	0 (sig)	0 (sig)	0 (sig)

Table 11: Post hoc Wilcoxon Signed-Rank tests for RDA Accuracy on the CMU Data set

	Pruned-Bilinear - Bicubic	Pruned-Bilinear - SHM	Pruned-Bilinear - Bilinear	Pruned-Bilinear - Restricted-SHM	Bicubic - SHM
z	5.349	6.943	2.346	4.917	5.612
Asymp. Sig. (2-tailed)	0 (sig)	0 (sig)	0.018	0 (sig)	0 (sig)

	Bicubic - Bilinear	Bicubic - Restricted-SHM	SHM - Bilinear	SHM - Restricted-SHM	Bilinear - Restricted-SHM
z	3.359	4.391	2.729	3.412	1.340
Asymp. Sig. (2-tailed)	0.001 (sig)	0 (sig)	0.006	0.001 (sig)	0.180

(59.5%) > Bicubic (46.9%), with Bilinear (57.1%) comparable to the top three.

Like those against the Caltech dataset, the comparison of computation times are much clearer than anticipated. There was a statistically significant difference depending on the preprocessor used, $\chi^2(4) = 54869.4, p = 0$. Table 10 show the post hoc analysis (as before) for the computation times.

The clarity of these significance tests give the hierarchical ordering (with overall average times) of: Pruned-Bilinear (18.3ms) < Bilinear (47.3ms) < Bicubic (58.2ms) < SHM (174.4ms) < Restricted-SHM (173.4ms).

6.9 Conclusion

This section will first present a general discussion and overview of the results and other outputs of the RDA preprocessors on Test Domains 1 & 2. This will be followed up with the research questions (Section 6.3) presented earlier in this chapter and their responses.

Test Domain 1 shows that the features provided by the preprocessors are sufficient enough to allow for learning. The traditional methods were able to reach an AUC of approximately 96% and three RDA variants were in the range 95% to 97%. All the systems tested are also comparable in computational time. Due to the minimal complexity of this test domain, it is suspected that most of this computational time is due to the ANN and preprocessors overheads. Test Domain 2 increased complexity with provide clearer answers in this case.

In Test Domain 2, against both data sets, RDA with Pruned Bilinear out performed the other methods presented. It is suspected that reduced features provided by the preprocessor is the most likely cause of the noticeable increase in performance and the low deviations in both cases, due to the reduced amount of noise the ANN has to contend with during training.

Looking at the Region Sets generated by SHM and Restricted-SHM and in

Table 12: Post hoc Wilcoxon Signed-Rank tests for RDA Overall Computation Time on the CMU Data set

	Pruned-Bilinear - Bicubic	Pruned-Bilinear - SHM	Pruned-Bilinear - Bilinear	Pruned-Bilinear - Restricted-SHM	Bicubic - SHM
z	122.5	122.5	122.5	122.5	122.5
Asymp. Sig. (2-tailed)	0 (sig)	0 (sig)	0 (sig)	0 (sig)	0 (sig)

	Bicubic - Bilinear	Bicubic - Restricted-SHM	SHM - Bilinear	SHM - Restricted-SHM	Bilinear a - Restricted-SHM
z	122.4	122.5	122.5	122.2	122.5
Asymp. Sig. (2-tailed)	0 (sig)	0 (sig)	0 (sig)	0 (sig)	0 (sig)

Table 13: The Likeness measure between each Region Set

Likeness	Pruned Bilinear	Restricted-SHM	SHM
SHM	0.366	0.698	-
Restricted-SHM	0.358	-	0.698
Pruned Bilinear	-	0.358	0.366

Table 14: The Similarity measure between each Region Set

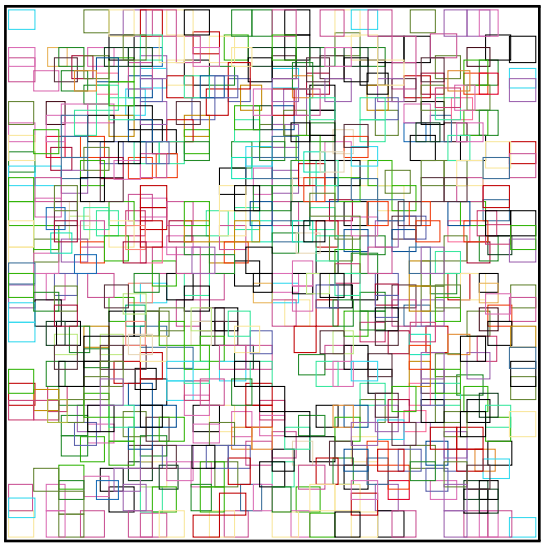
Similarity	Pruned Bilinear	Restricted-SHM	SHM
SHM	0.000	0.610	-
Restricted-SHM	0.000	-	0.610
Pruned Bilinear	-	0.000	0.000

response to the Caltech data set (Figure 27) shows some subjective similarity. The two metrics Likeness and Similarity (See section 6.6) are used to quantify this perceived similarity and are reported in tables 13 and 14. The Region Set returned by the Pruned Bilinear selection method contained a resolution that was not utilised by SHM and Restricted-SHM method. The high Likeness and Similarity between SHM and Restricted-SHM and low Likeness of each to Pruned Bilinear seem to suggest that difference observed in AUC can be attributed to those selection methods pruning features from potential pool that are of a larger benefit than is currently quantified.

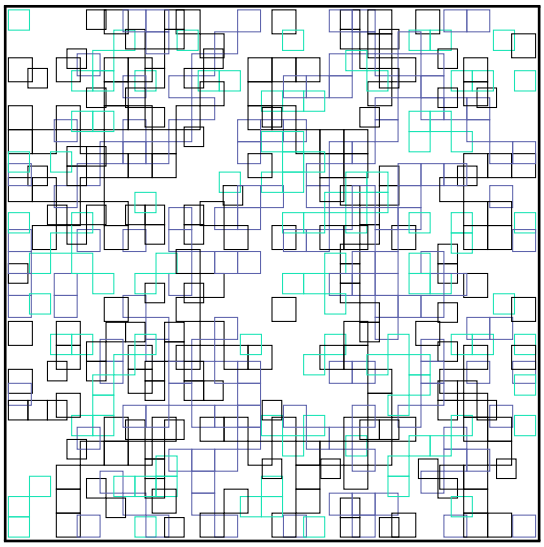
> Do the Region Selection Methods presented allow for the construction of viable RDA Preprocessors (preprocessors that allow an ANN to learn its intended task)?

Test Domain 1 shows that the three models presented here are able to provide features of sufficient quality that a suitable trained ANN will perform better than random.

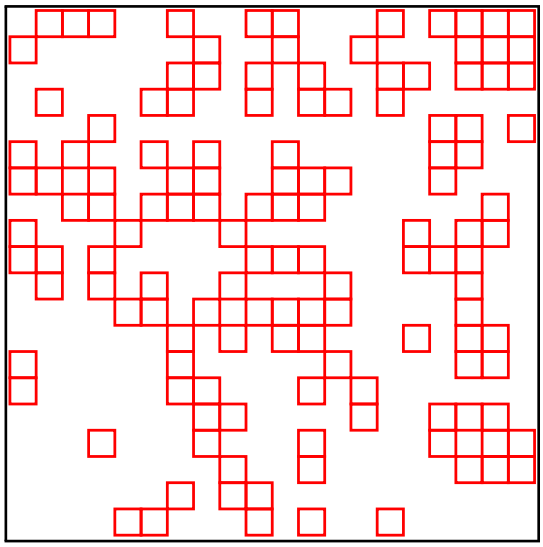
> Can such a preprocessor allow the associated ANN to achieve an improved measure of accuracy at a given task?



(a) Restricted SHM



(b) SHM



(c) Pruned Bilinear

Figure 27: The Region sets produced by (a) SHM, (b) Restricted-SHM and (c) Pruned-Bilinear. All regions at the same resolution share the same colour.

Test Domain 2 showed the three RDA preprocessors presented here produce significantly improved ANN system when compared to Bicubic preprocessor based systems and are comparable to those produced by Bilinear preprocessors.

> Do the increased computational overheads in the preprocessor allow for a greater reduction in computation in the ANN?

Test Domain 2 showed the RDA generated by the Pruned-Bilinear selection method performed significantly faster then both traditional preprocessors against both dataset.

Chapter 7

Subdividing Preprocessor

The Subdividing Preprocessor is a novel variant on the Regional Downsampling Algorithm. Instead of requiring an additional Region Construction stage like Regioned Downsample Preprocessor, this algorithm performs its alterations whilst the neural network is learning. The method behind the Subdividing preprocessor is to start with a near-sufficient bilinear interpolation preprocessor, and gradually (whilst training the ANN) improve the regions when certain criteria are met.

The motivation behind this addition is a further attempt to provide a useful Adaptive Preprocessor. Unlike VSP, this algorithm is designed to leverage the benefits of rectilinear downsampling as discussed at the beginning of chapter 6.

The underlying concept is to reallocate preprocessor outputs encapsulating features of negligible use to the ANN, and replace them with data of higher predictive value. The method presented here is to base this on the assumption that if the feature has a low saliency then it has little value to the ANN. It is expected that if a feature of low saliency were to be replaced with a constant low value, (with doing so effectively negating the input context of the feature,) the ANN would function as effectively as before. This concept can be extended such that the same preprocessor output can be attached to a new feature produced by the preprocessor, with a newly associated Input Context. The weights attached to

this feature would still be low enough that any data it does output would have little effect on the predictions of the ANN at first, but as future training progresses, correlations between the feature and the ANN's behaviour would increase the magnitude of the weights (assuming the new data feature is itself predictive).

7.1 Subdividing as an Adaptive Preprocessor

The start of an Adaptive Preprocessor is to perform a simpler process, similar to those of the construction phase of the Bespoke Preprocessors, but to a lower requirement for confidence/accuracy. This can then be used to determine the domain to search by the iterative process, *Update*, that operates along side the ANN's Learning. An Evaluation function is applied to the ANN to generate a quantified representation of the ANN's behaviour or conceptual usefulness of the features generated by the preprocessor, independent of the current quality of the ANN. Figure 12 shows how the traditional ANN learning feedback loop is altered to support the additional of the Update process and its associated Evaluation Function.

7.2 Research Question

To explore the Subdividing extension of the RDA algorithm, the follwing questions are proposed:

1. Does the Subdividing Preprocessor function as a viable Adaptive Preprocessors? (i.e. Does the preprocessor allow an ANN to learn its intended task whilst adjusting to the needs of the ANN?)

2. Can such a preprocessor allow the associated ANN to achieve an improved measure of accuracy at a given task?

3. Do the increased computational overheads in the preprocessor allow for a greater reduction in computation in the ANN?

As the generation of features from a source image is based on the same multi-resolution process that underlay the RDA algorithm, the implementation tests (Test Domain 1) will be skipped. Evaluation of this algorithm will start with Test Domain 2: Baseline Face Detection, 3.3, in the attempt to answer the above questions.

7.3 Region Selection and Subdivision

As part of a single subdividing event, a single region is divided into nine smaller regions (laid out in an equally spaced 3×3 grid within the region's coverage). Subdividing a region into nine smaller regions would increase the dimensionality of the returned value from this algorithm, so to counter this, the 'least useful' regions are recycled to keep the dimensionality constant. As mentioned in footnote 1 (Sec. 6.2), this algorithm is backed by Bilinear interpolation; the value of a pixel produced by Bilinear interpolation is slightly biased towards source pixels towards the centre of the processing area. To account for this, the algorithm uses a 3×3 sub-grid when subdividing.

As subdividing changes the functionality of the preprocessor, some training must occur before the ANN training is considered complete to provide ample time for the ANN to adjust to the changes that have taken place. For this reason a subdividing event was only checked for every third epoch, excluding the first three and last three epochs.

When to subdivide

The subdividing algorithm requires additional parameters compared to the traditional methods; these are res_{max}, T_{upper} and T_{lower}. These three parameters

are used to determine the suitability to subdivide. res_{max} represents a maximum resolution (both horizontally and vertically) that a subdividing process may zoom into. Any region that has an equivalent resolution greater than or equal to res_{max} may not be subdivided further. The T_{upper} and T_{lower} pair define the bounds on saliency at which subdividing can occur; formally,

$$\begin{aligned} \text{iff} \quad & \text{width}(R^{max}) \leq res_{max} \wedge \\ & \text{height}(R^{max}) \leq res_{max} \wedge \\ & \text{sal}(R^{max}) \geq \mathrm{T}_{upper} \wedge \\ & \max(\text{sal}(R^{min})) \leq \text{sal}(R^{max}) \cdot \mathrm{T}_{lower}, \end{aligned} \tag{33}$$

where $sal(X)$ returns the salience of region X, R^{max} is the single region with the highest saliency and R_n^{min} is the set of eight regions with minimal salience (where $n = 0 \ldots 7$).

First, all regions that have a resolution less than res_{max} in both the horizontal and vertical directions have their saliency computed (Sec. 5.2). If at least one region has a saliency above T_{upper} and at least eight have saliencies below T_{lower}, then the region with the highest saliency, R^{max}, is a valid target for subdivision, recycling the eight regions with the lowest saliencies, R_n^{min}, in the process. Regions that take part in a subdividing event cannot be considered for subsequent subdividing events until training of the ANN has occurred at least once. The determination of the values used for these is covered in section 7.5.

How to subdivide

Prior to the subdividing event, the region R^{max} has values $(R_x^{max}, R_y^{max}), R_h^{max} \times R_v^{max}$ (similarly for each of the minimally salient regions). For all nine regions, the values of R_h and R_v are set to $3 \cdot R_h^{max}$ and $3 \cdot R_v^{max}$ respectively. Temporary values X and Y are set to $3 \cdot R_x^{max}$ and $3 \cdot R_y^{max}$. The new coordinate (X, Y) represents the top left corner of the 3×3 sub-grid. The region R^{max} has it's coordinates

set to $(X + 1, Y + 1)$ such that it remains in the centre of the sub-grid. The remaining eight regions fill the remaining eight places around region R^{max}. This implementation places the regions R^{min} in ascending order of salience clockwise around region R^{max} starting at coordinate (X, Y).

An optional step can be used to speed up learning for the ANN by manually adjusting the weights connected to the regions altered in a subdividing event. For each neuron/unit in the hidden layer, identify the weight that is connected to the input of R^{max} and divide this value by 9. The weights that connect the hidden neuron/unit to the R^{min} regions are also set to this value. This was not applied to our implementation.

Subdivide Example

This example will look at a Subdividing Preprocessor that is initialised to a size of 4 regions wide, by 3 regions high. T_{upper} will be set to 0.9, T_{lower} will be set to 0.2 and res_{max} will be set sufficiently high that it isn't reached in this example. After a few rounds of training the saliency of the ANN is computed and used to determine the saliency of the preprocessed image. This is show in the left grid of figure 28. In this case, the subdividing conditions have been met, with region 6 having a saliency greater than 90%, and regions 0, 1, 4, 5, 8, 9, 10, 11 each have a saliency less than 20% of region 6. The space that was previously covered by region 6 is now divided up into 9 smaller spaces. Region 6 is reassigned to the centre of these new spaces, and the 8 regions with the lowest saliencies are assigned to each of the 8 remaining spaces.

7.4 Situating

Like RDAs, this model is backed by a set of Regions. The variation in obtaining the regions moves this model from the Bespoke group to the Adaptive group. The

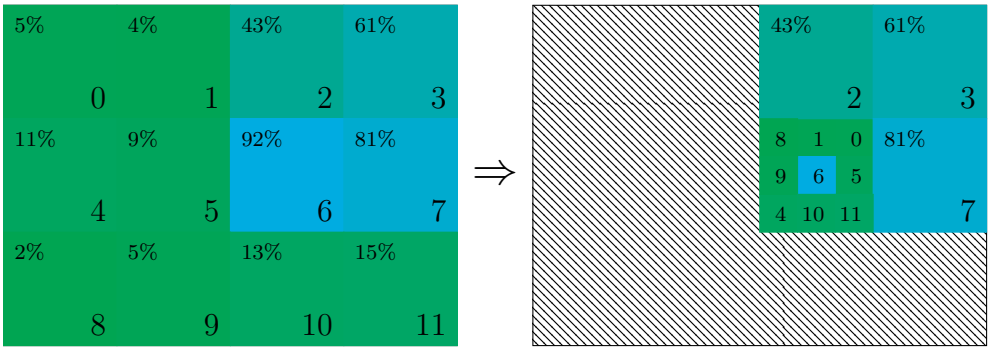


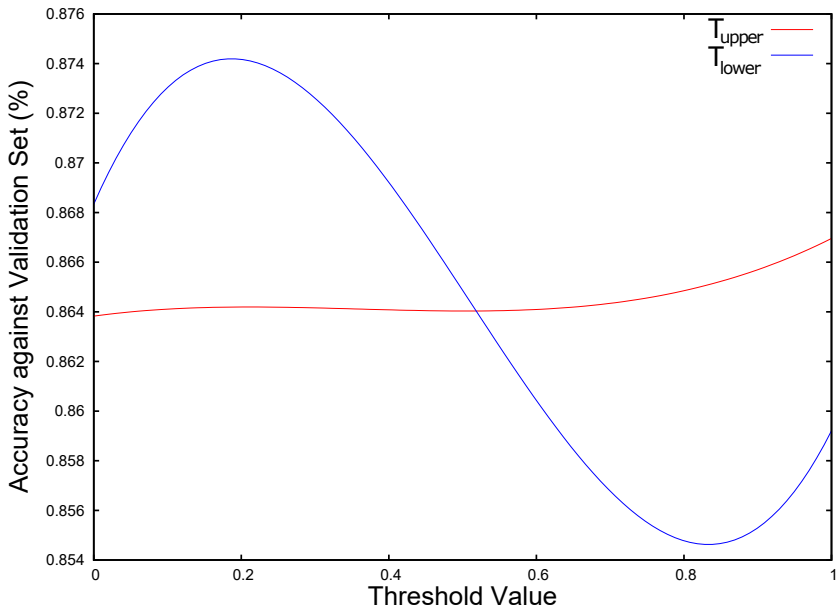
Figure 28: A visual representation of the subdividing event example.

initial regions are laid out as if representative of the output pixels of a traditional interpolation method and then iteratively refined according to the saliency of each region during the training of the ANN.

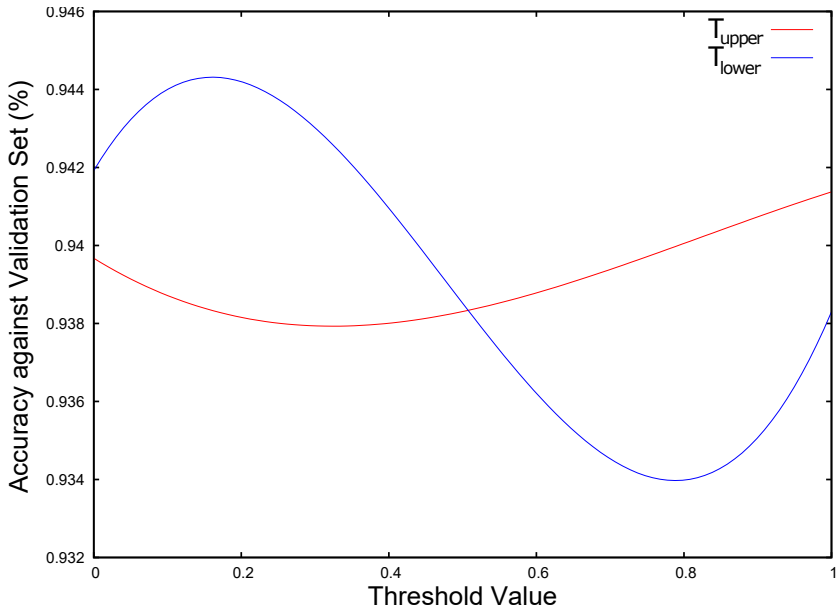
7.5 Parameter Selection

As part of the testing environments set-up, parameters need to be selected for the Traditional and Subdividing Preprocessors. To make for fair comparison between the various methods, optimal parameters are desired.

For the $width_{trad}$ and $height_{trad}$ parameters of the traditional methods, the single resolution that has the highest Cohen's kappa against the validation set is selected. This value is determined as a side-effect during the construction of the RDP (See sec. 6.4.1). The subdividing algorithm requires five parameters, $width_{subdivde}$, $height_{subdivde}$, res_{max}, T_{upper} and T_{lower}. Two trials of the Subdividing algorithm where performed, the first having the initial width and height equal to the traditional, the second group initialised with a reduced pool to reduce the number of output features (therefore a decrease in ANN complexity) and to



(a) CMU-130



(b) Caltech

Figure 29: The results from a brute-force parameter optimisation for the Subdividing Algorithm. Initially a linear best-fit was performed showing a slight positive correlation for T_{upper} and a strong negative correlation for T_{lower}. This was later replaced with a cubic best-fit (shown) to determine some finer grain details in the results. The graphs for both data sets appear highly similar, allowing for the attribution of the data-set independent parameters $\mathrm{T}_{upper} = 0.95$ and $\mathrm{T}_{lower} = 0.15$.

increase the difficulty of the task;

$$\begin{aligned} width_{subdivde} &= \lfloor \frac{3 \cdot width_{trad}}{4} \rfloor \quad \text{and} \\ height_{subdivde} &= \lfloor \frac{3 \cdot height_{trad}}{4} \rfloor \end{aligned} \tag{34}$$

were used, resulting in approximately half the output features of the traditional methods. res_{max} was set to RES_{max} to allow for a fair comparison between the novel algorithms in the future. The two threshold values, T_{lower} and T_{upper} were determined by a brute-force approach. For each point on a grid $0 \ldots 1 \times 0 \ldots 1$, at intervals of 0.01, a 20 epoch training run of a ANN with Subdividing Preprocessor was performed using the location on the grid as the parameters for T_{upper} and T_{lower}. This was repeated a total of 100 times, with the average accuracy against the training set being records. To select the final parameters, the data was parsed using gnuplot and a cubic best-fit performed (See Fig.29). From this the values $\mathrm{T}_{upper} = 0.95$ and $\mathrm{T}_{lower} = 0.15$ were selected.

7.6 Results

The underlying preprocessor for Subdividing is identical to that of RDA; for this reason, Test Domain 1 is skipped. This section will present the result for Test Domain 2.

Results from the comparison trials are presented in tables 15 and 18 in the same format as those of the previous chapter (Section 6.8). Fig. 30 presents an example preprocessor from each of the four Subdividing trials, with sub-figures 30a and 30d show-casing the preprocessors selecting for the curvature of the head.

Tables 15 and 18 show the results for the Face detection task against the Caltech and CMU-130 data sets, respectively.

Table 15: Aggregate results for the tests against the Caltech Data Set.

Preprocessor	**Features**	**AUC**		**Computation Time**	
		Average	**Standard Deviation**	**Average**	**Standard Deviation**
Bilinear (24×22)	1584	0.906	0.0613	63.4ms	4.83ms
Bicubic (24×22)	1584	0.897	0.0807	137.9ms	12.05ms
Subdividing (24×22)	1584	0.893	0.0324	7711ms	4024ms
Subdividing (18×16)	864(55%)	0.920	0.0116	6825ms	1477ms

7.6.1 Caltech Dataset

Using the Caltech data-set, there was a statistically significant difference in predictive accuracy depending on the preprocessor used, $\chi^2(4) = 29.124, p = 2 \times 10^{-6}$. Post hoc analysis using Wilcoxon signed-rank tests were performed, with Bonferroni correction applied.

Table 16 shows that reduced feature count Subdividing (18×16) (92.0%) method is comparable to the traditional Bilinear (90.6%) method; both of which show significant accuracy improvement over the matched feature count Subdividing method (89.3%). The Wilcoxon analysis was unable to identify any significance between Bicubic (89.7%) and the other methods, possibly due to the larger variance ($\sigma = 0.0807$).

For the comparison of computation times, the results are much clearer than anticipated, there was a statistically significant difference depending on the preprocessor used, $\chi^2(4) = 67406.6, p = 0$. Table 17 show the post hoc analysis (as before) for the computation times.

The clarity of these significance tests give the hierarchical ordering (with overall average times) of: Bilinear (63.4ms) $<$ Bicubic (137.9ms) $<$ Subdividing (Reduced Features) (6825.3ms) $<$ Subdividing (Matched Features) (7711.1ms).

Table 16: Post hoc Wilcoxon Signed-Rank tests for Subdividing Accuracy on the Caltech Data set

	Bilinear - Bicubic	Bilinear - Subdividing (24×22)	Bilinear - Subdividing (18×16)
z	0.556	3.187	1.636
Asymp. Sig. (2-tailed)	0.5785[1]	0.0014 (sig)[1]	0.1019[1]

	Bicubic - Subdividing (24×22)	Bicubic - Subdividing (18×16)	Subdividing (24×22) - Subdividing (18×16)
z	2.250	1.985	6.074
Asymp. Sig. (2-tailed)	0.0244	0.0471[1]	0 (sig)

Table 17: Post hoc Wilcoxon Signed-Rank tests for Subdividing Overall Computation Time on the Caltech Data set

	Bilinear - Bicubic	Bilinear - Subdividing (24×22)	Bilinear - Subdividing (18×16)
z	57.787	75.002	75.002
Asymp. Sig. (2-tailed)	0 (sig)	0 (sig)	0 (sig)

	Bicubic - Subdividing (24×22)	Bicubic - Subdividing (18×16)	Subdividing (24×22) - Subdividing (18×16)
z	75.002	75.002	54.420
Asymp. Sig. (2-tailed)	0 (sig)	0	0 (sig)

Table 18: Aggregate results for the tests against the CMU Data Set.

Preprocessor	**Features**	**AUC**		**Computation Time**	
		Average	**Standard Deviation**	**Average**	**Standard Deviation**
Bilinear (25×19)	1425	0.781	0.0946	47.3ms	9.86ms
Bicubic (25×19)	1425	0.749	0.1079	58.2ms	10.67ms
Subdividing (25×19)	1425	0.934	0.0244	10591ms	3112ms
Subdividing (18×14)	756(53%)	0.948	0.0124	6094ms	1124ms

7.6.2 CMU Dataset

Using the CMU data-set, there was a statistically significant difference in predictive accuracy depending on the preprocessor used, $\chi^2(4) = 235.787, p = 7 \times 10^{-51}$. Post hoc analysis using Wilcoxon signed-rank tests were performed, with Bonferroni correction applied.

Table 19 suggests that the following hierarchy describes the expected accuracy of the preprocessors: Subdividing (18×14) (94.4%) > Subdividing (25×19) (93.4%) > Bilinear (78.1%) > Bicubic (74.9%)

Like those against the Caltech dataset, the comparison of computation times are much clearer than anticipated. There was a statistically significant difference depending on the preprocessor used, $\chi^2(4) = 53487.4, p = 0$. Table 20 show the post hoc analysis (as before) for the computation times.

The clarity of these significance tests give the hierarchical ordering (with overall average times) of: Bilinear (47.3ms) < Bicubic (58.2ms) < Subdividing (18×14) (10591ms) < Subdividing (25×19) (61828ms).

Table 19: Post hoc Wilcoxon Signed-Rank tests for Subdividing Accuracy on the CMU Data set

	Bilinear - Bicubic	Bilinear - Subdividing (25×19)	Bilinear - Subdividing (18×14)
z	3.233	8.725	8.725
Asymp. Sig. (2-tailed)	0.001 (sig)	0 (sig)	0 (sig)

	Bicubic - Subdividing (25×19)	Bilinear - Subdividing (18×14)	Subdividing (25×19) - Subdividing (18×14)
z	8.463	8.463	5.376
Asymp. Sig. (2-tailed)	0 (sig)	0 (sig)	0 (sig)

Table 20: Post hoc Wilcoxon Signed-Rank tests for Subdividing Overall Computation Time on the CMU Data set

	Bilinear - Bicubic	Bilinear - Subdividing (25×19)	Bilinear - Subdividing (18×14)
z	122.392	122.476	122.476
Asymp. Sig. (2-tailed)	0 (sig)	0 (sig)	0 (sig)

	Bicubic - Subdividing (25×19)	Bilinear - Subdividing (18×14)	Subdividing (25×19) - Subdividing (18×14)
z	122.476	122.476	122.418
Asymp. Sig. (2-tailed)	0 (sig)	0 (sig)	0 (sig)

7.7 Conclusion

This section will first present a general discussion and overview of the results and other outputs of the Subdividing preprocessors on Test Domain 2. This will be followed up with the research questions (Section 7.2) presented earlier in this chapter and their responses.

Against Test Domain 2, for both datasets, the reduced feature count variant of Subdividing out performed the variant with a feature count matching the traditional methods. In the case of the CMU dataset, both variants were shown to have significantly improved accuracy compared to the traditional methods with the reduced variant being further able to improve performance. Against the Caltech dataset, the reduced feature variant was only able to show significant improvement to Bilinear. This may be due to a large variance in the Bicubic results. Repeating this study with a larger sampling from this dataset may clarify these results. The improvement identified in the reduce feature variant of Subdividing is suspected to be the result of the ANN being able to identify patterns in the feature set faster from data due to the reduced noise.

Computational time was not a a primary goal for this implementation of Subdividing; as such, it is not a surprise to find that it is significantly slower than the traditional methods.

The conclusion to draw from these to insights is that the Subdividing algorithm presented here is only justifiable solution in limited cases. Until improvement in computational run time are applied to the algorithm, its use should be limited to situations where an ANN is unable to learn due too excessive noise obscuring relevant patterns in the input images and runtime complexity is not a significant issue.

This technique could be used as a tool to identify where the predictive value in the data set could be used in the discovery or construction of future preprocessors. As a demonstration, figure 31 show a sample of three region sets produced as a

Table 21: Distance metrics (Sec. 6.6) of the three random selected region sets generated as part of the Baseline Face Detection test against the CMU-130 data set. See figure 31 for their visual appearance.

Likeness	(a)	(b)	(c)
(a)	-	0.470	0.492
(b)	0.470	-	0.503
(c)	0.492	0.503	-

Similarity	(a)	(b)	(c)
(a)	-	0.468	0.470
(b)	0.468	-	0.502
(c)	0.470	0.502	-

result of testing the Subdividing algorithm (with matching region counts to the traditional methods) against Test Domain 2 using CMU-130 data set. A quick viewing of the samples show an expected bias towards the boundary of the cranium and background, with some focus towards the eyes and bridge of the nose. The samples also show some interest towards the top right corner of the scene and this may be evidence to an undesired coincidentally correlating patterns in that region. (See Section 2.1.3 for a discussion on Hughes Phenomenon.) When compared by the Similarity and Likeness metrics (Table 21) presented in section 6.6, the samples shows a strong similarity in the selection of regions. This leads to the implication that the these regions are strongly linked to the given task and other preprocessors that focus on these areas may also prove beneficial. As this opinion is based on a sample of just three convergent preprocessors from a small pool of subdividing preprocessors generated as a side effect to an alternative task, this statement can only be considered as an intriguing observation, but demonstrates the potential extrapolations that could be made.

> Does the Subdividing Preprocessor function as a viable Adaptive Preprocessors? (i.e. Does the preprocessor allow an ANN to learn its intended task whilst adjusting to the needs of the ANN?)

In all tested situations, the ANN when supported by a Subdividing preprocessor was able to learn the task presented. In each case, its was show to

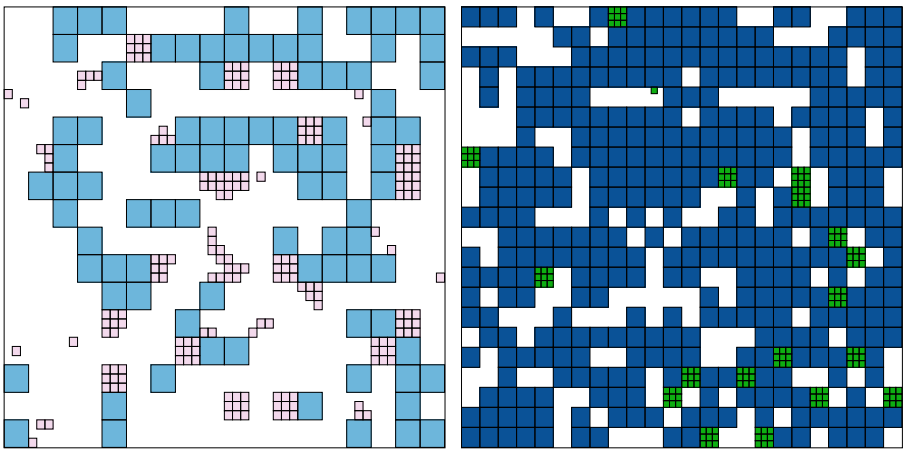
have significantly improved or equivalent AUC compared to the traditional methods presented, and in all cases the Standard Deviation was lower when compared to the traditional preprocessors.

> Can such a preprocessor allow the associated ANN to achieve an improved measure of accuracy at a given task?

The analysis presented here show that Subdividing can provide significant increases to accuracy, but the improvements may be task dependent.

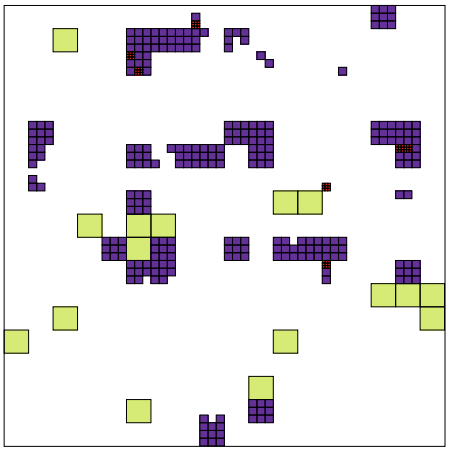
> Do the increased computational overheads in the preprocessor allow for a greater reduction in computation in the ANN?

The timing results present show that this is not the case.

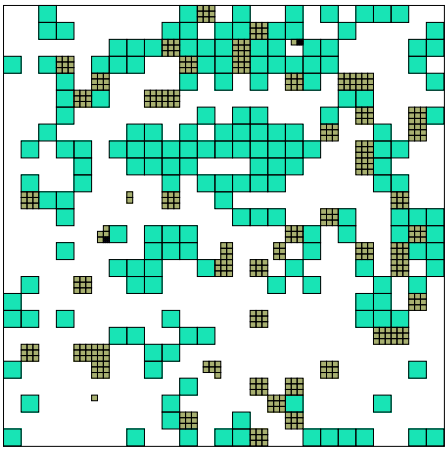


(a) Caltech (reduced feature count)

(b) Caltech (matching feature count)



(c) CMU-130 (reduced feature count)



(d) CMU-130 (matching feature count)

Figure 30: A sample of preprocessors from each of the Subdividing trials. Figures (a) and (d) where selected as representative of the typical density (though layout is highly variable). Figure (b) is an example of a Preprocessor-ANN system that failed to learn the task to a high degree of accuracy, and thus little has occurred in the way of subdividing events. Figure (c) is an example of the ANN over-fitting, affecting the preprocessor produced. Each colour represents a unique resolution

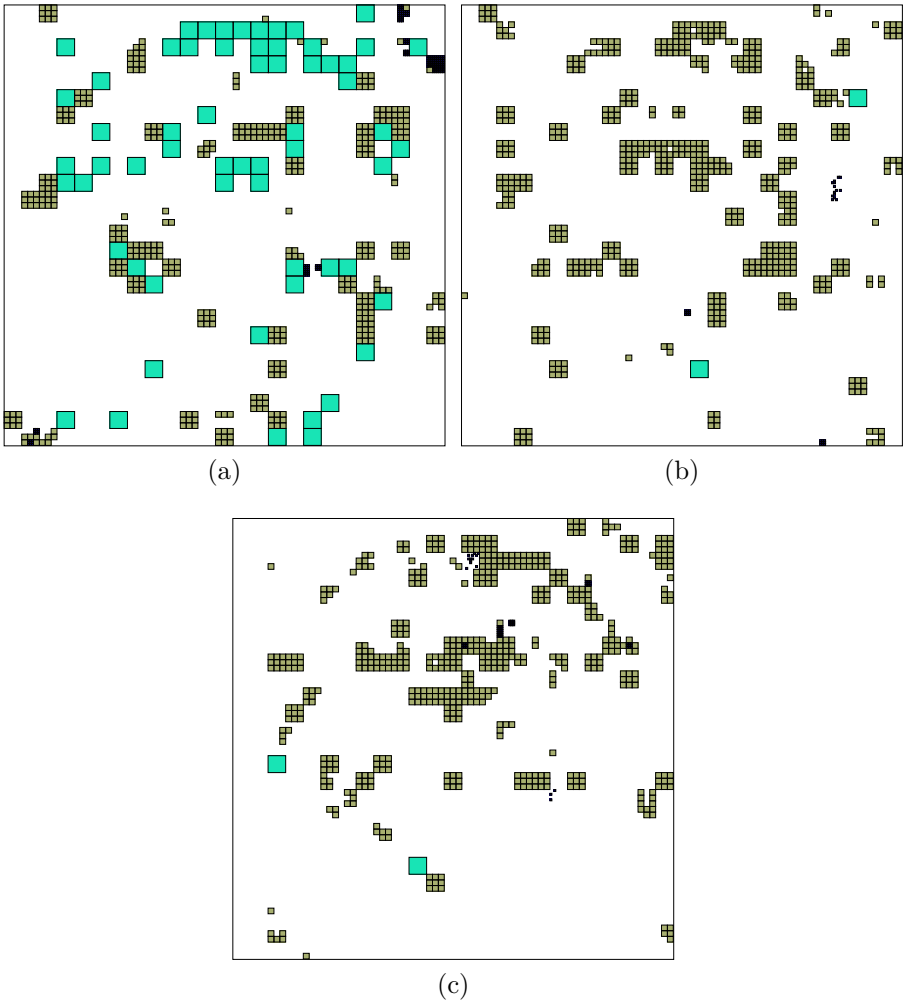


Figure 31: A random sample of three resultant Region Sets from the produced against the CMU-130 data set.

Chapter 8

Evaluation and Discussion

This chapter is provided to give an overview of the novel preprocessors presented in this thesis, and response to the results gathered as part of the studies discussed. For the reader's benefit, tables 22 and 23 have been provided that display the complete results of the Test Domain 2 as performed on both classes of preprocessor. As the comparisons between the novel preprocessors and the traditional methods have already been covered in earlier chapters, this will be passed over for discussion in this chapter. As such, this chapter will be focused on the comparison between RDA and Subdividing. The tests presented here were performed many weeks apart. Due to this, different processes were occurring on the machines during the execution of the Test Domains (as can be seen in the slight change in the traditional preprocessors runtime), making direct numerical comparison difficult. To counter this, comparisons will be made by the relationship between the algorithm in question and the traditional methods processed within the same execution.

For Test Domain 2 using the Caltech dataset, Subdividing has shown to have statistically significant increase in AUC for some cases (when it has a reduced feature set compared to the Bilinear traditional methods, an absolute increase in AUC of between 0.014), but the results were unable to reject the null-hypothesis

when comparing against Bicubic or using an equivalent feature count. The presented RDA methods show statistically significant improvements compared to Bilinear (an absolute increase of between 0.034 to 0.038). Focusing on the accuracy measures for this task, RDA shows a greater improvement than the Subdividing algorithms. Given the analysis in section 6.8.2, the recommendation of using RDA with Restricted-SHM (if computation time is not a concern), or RDA with Pruned Bilinear (when reduction in computational runtime is required) are justifiable.

Test Domain 2 using the CMU dataset provides a different conclusion. In this case, both Subdividing implementations provided significant improvement to accuracy (an absolute increase in AUC of between 0.153 to 0.199). If accuracy is the only concern, then Subdividing using a reduced feature set is a justified recommendation due to its statistical significance (Section 7.6.2). If run-time constraints apply to the task, then the recommendation of RDA using Pruned Bilinear is the most justified due to its comparable accuracy (no statistically significant improvement over the traditional methods) and statistically significant reduction in computational time as detailed in section 6.8.2.

Table 22: An aggregate of the results for both the RDA and Subdividing tests against the Caltech Data Set.

Preprocessor	**Features**	**AUC**		**Computation Time**	
		Average	**Standard Deviation**	**Average**	**Standard Deviation**
RDA Results					
Bilinear (24×22)	1584	0.943	0.0857	48.3ms	8.01ms
Bicubic (24×22)	1584	0.966	0.0654	60.0ms	8.74ms
RDA + SHM	36	0.977	0.0057	133.9ms	34.69ms
RDA + Restricted-SHM	39	0.981	0.0024	153.7ms	43.18ms
RDA + Pruned Bilinear	42	0.980	0.0113	27.8ms	5.32ms
Subdividing Results					
Bilinear (24×22)	1584	0.906	0.0613	63.4ms	4.83ms
Bicubic (24×22)	1584	0.897	0.0807	137.9ms	12.05ms
Subdividing (24×22)	1584	0.893	0.0324	7711ms	4024ms
Subdividing (18×16)	864(55%)	0.920	0.0116	6825ms	1477ms

Table 23: An aggregate of the results for both the RDA and Subdividing tests against the CMU-130 Data Set.

Preprocessor	**Features**	**AUC**		**Computation Time**	
		Average	**Standard Deviation**	**Average**	**Standard Deviation**
RDA Results					
Bilinear (25×19)	1425	0.571	0.1414	47.3ms	9.86ms
Bicubic (25×19)	1425	0.469	0.1668	58.2ms	10.67ms
RDA + SHM	48	0.623	0.0769	174.4ms	71.79ms
RDA + Restricted-SHM	48	0.595	0.1119	173.4ms	72.84ms
RDA + Pruned Bilinear	21	0.596	0.0178	18.3ms	6.47ms
Subdividing Results					
Bilinear (25×19)	1425	0.781	0.0946	47.3ms	9.86ms
Bicubic (25×19)	1425	0.749	0.1079	58.2ms	10.67ms
Subdividing (25×19)	1425	0.934	0.0244	10591ms	3112ms
Subdividing (18×14)	756(53%)	0.948	0.0124	6094ms	1124ms

Chapter 9

Conclusion and Further Work

At the start of this thesis, it was claimed that the following contributions would be made:

1. A framework with which to classify and aid discussion of image preprocessors,
2. A metric which can approximate the predictive accuracy of a neural unit or input feature to an ANN,
3. Three novel image preprocessors and their implementation procedures, and
4. A methodology to systematically compare these presented preprocessors.

It is believed that these contributions have been in the following chapters/ sections of this thesis:

1. Chapter 4 provides a framework to classify an image preprocessor as Bespoke or Adaptive based on the way in which concrete implementations (including parameters) are applied to the task.
2. Section 5.2 presents the Saliency metric for the use of qualitatively measuring an input feature's predictive value for an unknown task, and as a by-product can provide similar measure for all intermediate nodes in the ANN.

3. Chapters 5, 6 and 7 present three models for image preprocessor design, not previously presented in the literature, each with variations in implementation.

4. Section 3.3 introduced a Baseline Face Detection task as a variant of the work by Jain and Learned-Miller (2010).

Of the three novel preprocessors contributions, two (RDA and Subdividing) were found to have statistically significant improvements in accuracy for some domains. The third (VSP) has been found to have statistically poorer performance than the traditional methods with recommendations to disregard it as an avenue for further research.

9.1 Future/Further work

This final section presents a number of comments regarding to the limitations of this work. This includes the identification of: results that, in aggregate, hint at potential for improvements within a particular algorithm even when the individual results demonstrate poor performance, potential avenues of investigation for runtime optimisation and where the results obtained fall short of finding the extent at which an algorithm can perform.

Though RDA + Pruned Bilinear appears to have performed well in this study (Sec. 6.9), the results from RDA + SHM and RDA + Restricted-SHM hints at potential for improvement, whether it is to be realised through minor modification to the selection method itself or just refinement of the threshold parameters. Future studies that attempt to resolve this query are a potential avenue of research.

In this work, the RDA and Subdividing algorithms were only analysed with respect to a single applicable problem domain, face detection. This was due to the unexpected and significant time costs in the generation and testing of both the algorithms presented in this work and the testing environment. Further studies of

these algorithms to determine and/or demonstrate the generality and suitability of these to different tasks; along with the generation of newer Bespoke or Adaptive Preprocessors could prove beneficial.

Optimisation of the RDA algorithm runtime complexity has not been a focus of this research. In this research a naïve optimisation process was used and is a prime location for improvement. An increasingly complex optimiser could weigh the costs of a smaller number of bilinear interpolation components (reducing overheads), or more numerous smaller bilinear interpolation components, in attempts to minimise the runtime costs involved with preprocessing.

In these tests, there was no investigation into how aggressive the reduction in Subdividing Features could be pushed before there was a significant reduction in the preprocessors and associated ANNs accuracy. It is also expected that further investigations into this lower bound would also provide large savings in computational run time.

The Subdividing algorithms current implementation uses Bilinear interpolation to process each region that it encapsulates. Due to this, the runtime of a Subdividing Preprocessor is expected to be slower than that of the traditional methods. Expanding out the functioning of the interpolations may reduce overheads caused by duplicate computations and may allow for pre-computation of some necessary values. This is expected to reduce some runtime cost and transfer others from runtime to learning/training time.

Bibliography

Achanta, R. et al. (2012). Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11), pp. 2274–2282.

Buia, C. and Tiesinga, P. (2006). Attentional modulation of firing rate and synchrony in a model cortical network. *Journal of Computational Neuroscience*, 20, pp. 247–264, 10.1007/s10827-006-6358-0.

Campbell, C. (2007). The perils of image.getscaledinstance() blog. https://community.oracle.com/docs/DOC-983611.

Canon (2009). Digital image processor digic 4. [Online Video; accessed 12-March-2018].

Catmull, E. and Rom, R. (1974). A class of local interpolating splines. In R. E. BARNHILL and R. F. RIESENFELD, eds., *Computer Aided Geometric Design*, Academic Press, pp. 317 – 326.

Chandrashekar, G. and Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1), pp. 16 – 28, 40th-year commemorative issue.

Cheng, I. et al. (2015). Foveated high efficiency video coding for low bit rate transmission. In *2015 IEEE International Symposium on Multimedia (ISM)*, IEEE, pp. 547–552.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1), pp. 37–46.

Cohen, J. (1968). Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin*, 70(4), p. 213.

Colmenarez, A. and Huang, T. (1997). Face detection with information-based maximum discrimination. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pp. 782–787.

Covell, M. et al. (2017). Target-quality image compression with recurrent, convolutional neural networks. *arXiv preprint arXiv:170506687*.

Crick, F. and Koch, C. (1990). Some reflections on visual awareness. *Cold Spring Harbor Symposia on Quantitative Biology*, 55, pp. 953–962, `http://symposium.cshlp.org/content/55/953.full.pdf+html`.

Davies, S. et al. (2010). Interfacing real-time spiking i/o with the spinnaker neuromimetic architecture. In *Proceedings of the 17th International Conference on Neural Information Processing: Australian Journal of Intelligent Information Processing Systems*, pp. 7–11.

Dirichlet, P. G. L. (1850). Über die reduktion der positiven quadratischen formen mit drei unbestimmten ganzen zahlen. *Journal für die Reine und Angewandte Mathematik*, 40, pp. 209–227.

Engelbrecht, A., Cloete, I. and Zurada, J. (1995). Determining the significance of input parameters using sensitivity analysis. In J. Mira and F. Sandoval, eds., *From Natural to Artificial Neural Computation*, *Lecture Notes in Computer Science*, vol. 930, Springer Berlin Heidelberg, pp. 382–388.

Ester, M. et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, AAAI Press, pp. 226–231.

Feraud, R. et al. (2000). A fast and accurate face detector for indexation of face images. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pp. 77–82.

Forster, C. et al. (2015). Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 111–118.

Garg, R., Yang, S. and Scherer, S. (2018). Monocular and stereo cues for landing zone evaluation for micro uavs. *arXiv preprint arXiv:181203539*.

Geisler, W. and Kortum, P. (2001). Foveated image coding system and method for image bandwidth reduction. US Patent 6,252,989.

Greenhow, K. A. and Johnson, C. G. (2014). Region based image preprocessor for feed-forward perceptron based systems. In Z. Zeng, Y. Li and I. King, eds., *Advances in Neural Networks – ISNN 2014*, Springer International Publishing, Lecture Notes in Computer Science, pp. 414–422.

Greenhow, K. A. and Johnson, C. G. (2015). Regioned downsample for ann image classification: Alternate selection methods. In *2015 SAI Intelligent Systems Conference (IntelliSys)*, pp. 793–797.

Greenhow, K. A. and Johnson, C. G. (2017). Adaptive image downsampling preprocessor for artificial neural networks. In *2017 Computing Conference*, pp. 766–774.

Gu, Q. and Li, S. (2000). Combining feature optimization into neural network based face detection. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, vol. 2, pp. 814–817 vol.2.

He, S. et al. (2015). Supercnn: A superpixelwise convolutional neural network for salient object detection. *International Journal of Computer Vision*, 115(3), pp. 330–344.

Hughes, G. (1968). On the mean accuracy of statistical pattern recognizers. *Information Theory, IEEE Transactions on*, 14(1), pp. 55–63.

Itti, L. (2004). Automatic foveation for video compression using a neurobiological model of visual attention. *Image Processing, IEEE Transactions on*, 13(10), pp. 1304 –1318.

Itti, L., Koch, C. and Niebur, E. (1998). A model of saliency-based visual attention for rapid scene analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(11), pp. 1254 –1259.

Jain, V. and Learned-Miller, E. G. (2010). Fddb: A benchmark for face detection in unconstrained settings. Tech. rep., UMass Amherst.

Lang, R., Zhao, L. and Jia, K. (2016). Brain tumor image segmentation based on convolution neural network. In *2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pp. 1402–1406.

Lee, S., Pattichis, M. S. and Bovik, A. C. (2001). Foveated video compression with optimal rate control. *IEEE Transactions on Image Processing*, 10(7), pp. 977–992.

Lew, M. and Huijsmans, N. (1996). Information theory and face detection. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, vol. 3, pp. 601–605 vol.3.

Li, Y., Liu, H. and Chen, Z. (2016). Perceptually-lossless image coding based

on foveated-jnd and h. 265/hevc. *Journal of Visual Communication and Image Representation*, 40, pp. 600–610.

Lin, S.-H., Kung, S.-Y. and Lin, L.-J. (1997). Face recognition/detection by probabilistic decision-based neural network. *Neural Networks, IEEE Transactions on*, 8(1), pp. 114–132.

Lukas, S. et al. (2016). Implementing discrete wavelet and discrete cosine transform with radial basis function neural network in facial image recognition. *Journal of Image and Graphics*, 4(1).

March i Nogué, J. (2016). Schematic diagram of the human eye.

Mashito, K. (2001). Pedestrian observing device. [Patent 2006-104047 (Japan); Issued 30-June-1999].

Meyer, F. (1991). Un algorithme optimal de ligne de partage des eaux. *Actes du*, 2, pp. 847–859.

Mordvintsev, A., Olah, C. and Tyka, M. (2015a). Deepdream-a code example for visualizing neural networks. [Online; accessed 20-November-2018].

Mordvintsev, A., Olah, C. and Tyka, M. (2015b). Inceptionism: Going deeper into neural networks. [Online; accessed 20-November-2018].

Mozer, M. C. and Smolensky, P. (1989a). Skeletonization: a technique for trimming the fat from a network via relevance assessment. *Advances in neural information processing systems*, 1, pp. 107–115.

Mozer, M. C. and Smolensky, P. (1989b). Using relevance to reduce network size automatically. *Connection Science*, 1:1, pp. 3 – 16.

Nikon (2007). Electronic camera that detects and extracts faces. [Patent 8,379,108 (United States of America); Issued 3-April-2006].

Ooi, S. Y. et al. (2016). Image-based handwritten signature verification using hybrid methods of discrete radon transform, principal component analysis and probabilistic neural network. *Applied Soft Computing*, 40, pp. 274 – 282.

Osuna, E., Freund, R. and Girosi, F. (1997). Training support vector machines: an application to face detection. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pp. 130–136.

Pourghahestani, F. A. and Rashedi, E. (2015). Object detection in images using artificial neural network and improved binary gravitational search algorithm. In *Fuzzy and Intelligent Systems (CFIS), 2015 4th Iranian Joint Congress on*, pp. 1–4.

Press, W. H. et al. (1992). *Numerical Recipes in C - The Art of Scientific Computing*. Cambridge University Press, 1st edn., iSBN 0-521-43108-5.

Qazi, N. and Wong, B. (2016). Semantic based image retrieval through combined classifiers of deep neural network and wavelet decomposition of image signal. *Applied Soft Computing*.

Rashedi, E. and Nezamabadi-pour, H. (2014). Feature subset selection using improved binary gravitational search algorithm. *Journal of Intelligent & Fuzzy Systems*, 26(3), pp. 1211–1221.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, New York, NY, USA: ACM, SIGGRAPH '87, pp. 25–34.

Roth, D., Yang, M.-H. and Ahuja, N. (2000). A SNoW-based face detector. *Urbana*, 51, p. 61801.

Rowley, H., Baluja, S. and Kanade, T. (1998). Neural network-based face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(1), pp. 23–38.

Scassellati, B. (1998). Eye finding via face detection for a foveated active vision system. In *AAAI/IAAI*, pp. 969–976.

Scassellati, B. (1999). A binocular, foveated active vision system. Tech. rep., MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB.

Shibata, K. and Utsunomiya, H. (2011). Discovery of pattern meaning from delayed rewards by reinforcement learning with a recurrent neural network. In *The 2011 International Joint Conference on Neural Networks (IJCNN)*, pp. 1445 –1452.

Shipley, T. and Kellman, P. (2001). Gaze control for face learning and recognition by humans and machines. *From fragments to objects: Segmentation and grouping in vision*, 130, p. 463.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2), pp. 99–127, `https://doi.org/10.1162/106365602320169811`.

Subbotin, Y. (2001). Spline interpolation. `http://www.encyclopediaofmath.org/index.php?title=Spline_interpolation`, [Online; accessed 03-November-2016].

Sung, K.-K. and Poggio, T. (1998). Example-based learning for view-based human face detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(1), pp. 39–51.

Terzopoulos, D., Rabie, T. and Grzeszczuk, R. (1997). Perception and learning in artificial animals. In *Proc. of the 5th Int. Workshop on Artificial Life: Synthesis and Simulation of Living Systems (ALIFE-96)*, pp. 346–353.

Turkowski, K. (1990). Filters for common resampling-tasks. *Graphics gems*, pp. 147–165.

Voronoi, G. (1908). Nouvelles applications des paramìtres continus à la théorie des formes quadratiques. premier mémoire. sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik*, 133, pp. 97–102.

Wang, Z., Lu, L. and Bovik, A. C. (2003). Foveation scalable video coding with automatic fixation selection. *IEEE Transactions on Image Processing*, 12(2), pp. 243–254.

Weber, M. (1999). Caltech frontal face database. California Institue of Technology Online Archive.

Weisstein, E. W. (2015). Delaunay triangulation. `http://mathworld.wolfram.com/DelaunayTriangulation.html`, [Online; accessed 17-May-2017].

Wikipedia contributors (2018). Receiver operating characteristic — Wikipedia, the free encyclopedia. [Online; accessed 4-November-2018].

Yang, M.-H., Abuja, N. and Kriegman, D. (2000). Face detection using mixtures of linear subspaces. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pp. 70–76.

Zbontar, J. and LeCun, Y. (2016). Stereo matching by training a convolutional neural network to compare image patches. *Journal of Machine Learning Research*, 17(1-32), p. 2.

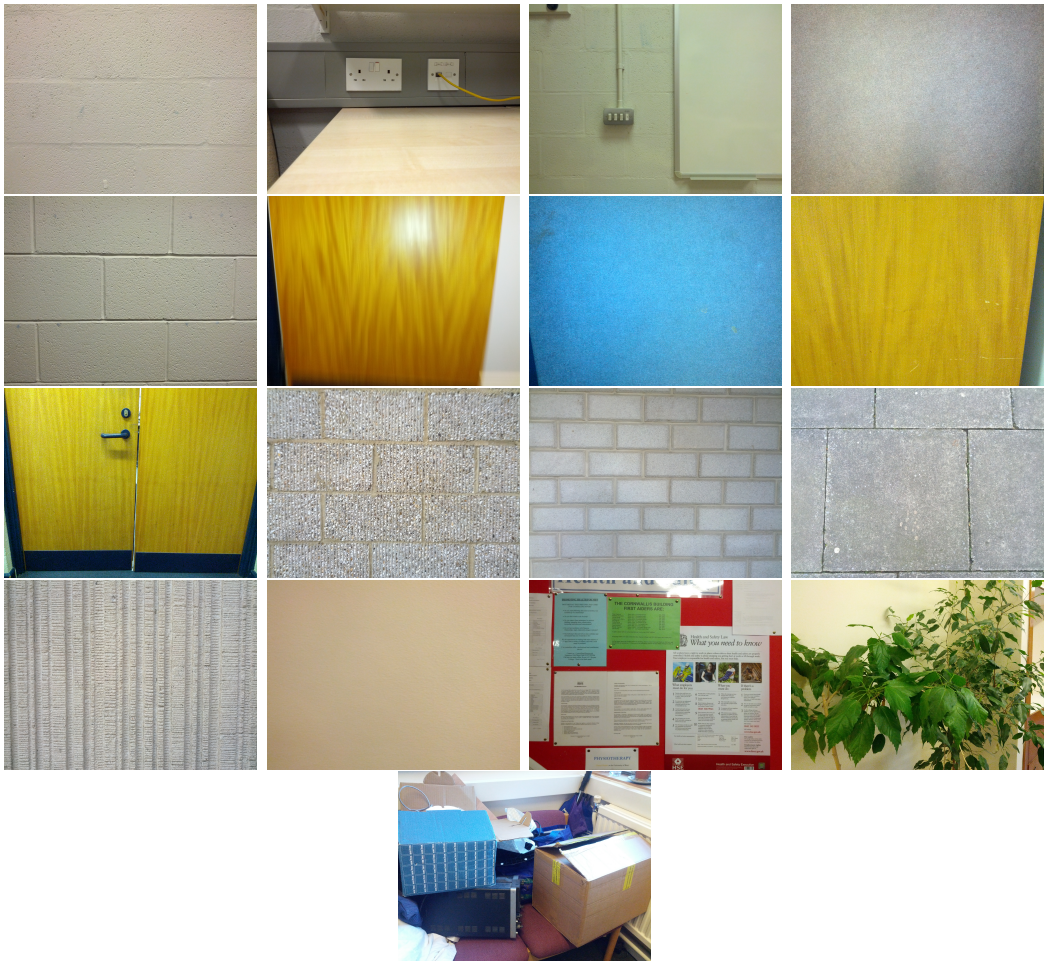
Appendix A

VSP Image Sources

The complete set of images used in the Presence Tests as described in section 3.2. Both the background and object images are stored in PNG format with 24-bit RGB colour encoding.

A.1 Stuffed Bear

The stuffed bear image used for the object presence testing domain as described in section 3.2. This image was selected due to the consistent colouration, complex shape and both high and low contrasts with some of the background images.

A.2 Backgrounds (17 images)

Images from around the Cornwallis building, University of Kent; used as backgrounds for generating the test and training set images for VSP (Sec. 3.2).