Recognition and Prominence Ranking of Alphanumeric Number Sequences in Images

Alex Cummaudo
BSc *Swinburne*

Supervised by Prof. Rajesh Vasa, Assoc. Prof. Andrew Cain

A thesis submitted in partial fulfilment of the requirements for the
Bachelor of Information Technology (Honours)

Deakin Software and Technology Innovation Laboratory
School of Information Technology
Deakin University, Australia

October 2017

Abstract

Text detection in natural images is a growing area with increasing applications, including traffic sign and license plate recognition, and text-based image search. Robustly detecting and recognising text is especially challenging when text is deformed, such as the photometric and geometric distortions of text worn by a moving subject in unstructured scenes. Existing methods of text detection in such cases are classified as learning-based or connected component (CC)-based, applying a mix of enhanced detection techniques—such as stroke width transformation (SWT), canny-edge detection and maximally stable extremal regions (MSERs)—and feeding candidates into optical character recognition (OCR) engines or neural networks to recognise the text. This study proposes applying a learning-based approach using deep-learning strategies to automate the recognition of racing bib numbers (RBNs) in a natural image dataset of various marathons, and then ranking detected subject's photos in order of prominence. Experimental results showed that these deep-learning strategies performed favourably against other methods using a consistent dataset, prompting further investigation in the generality of the technique developed to other similar subject material.

Declarations

I certify that the the thesis entitled "Recognition and Prominence Ranking of Alphanumeric Number Sequences in Images" submitted for the degree of Bachelor of Information Technology (Honours) is the result of my own work and that where reference is made to the work of others, due acknowledgement is given. I also certify that any material in the thesis that has been accepted for a degree or diploma by any university of institution is identified in the text.

Alex Cummaudo, BSc *Swinburne*
October 2017

We certify that the thesis prepared by Alex Cummaudo entitled "Recognition and Prominence Ranking of Alphanumeric Number Sequences in Images" is prepared according to our expectations and that the honours coordinator can proceed to accept this submission for examination.

Prof. Rajesh Vasa
October 2017

Assoc. Prof. Andrew Cain
October 2017

Dedicated to Tom Fellowes.

Acknowledgements

I thank everyone at DSTIL for fostering a highly supportive and motivating environment to work in: you make every day enjoyable and are all my second family. I especially thank Simon Vajda and Rodney Pilgrim for their direction throughout my honours year, the exceptional research guidance provided by Raj Vasa and Nicola Pastorello and our fruitful discussions and well-valued feedback on this thesis, and Jake Renzella and Reuben Wilson for their much-needed support and friendship throughout the last two years—and the increasing number of coffees shared at Fusion as the hand-in date drew nearer. I also strongly thank Andrew Cain for his extraordinary teaching efforts over many years, inspiring thousands of students to cherish the art of programming (myself included) and with whom I have developed a highly-valued mentorship with in guiding me throughout life. And lastly, I thank my loving family and patient partner Tom Fellowes, without whom this thesis (indeed this year) would not be possible.

Contents

List of Abbreviations

ACL Argus Constraint Language. 42–46

ADF Argus Data Format. xxi, 42, 44, 46–48, 51, 60–62, 112–115

AI Artificial Intelligence. 27, 29, 42, 46, 48, 56, 62, 67

AMT Amazon Mechanical Turk. 60, 63, 64, 67

APEP Annotation and Performance Evaluation Platform. 60, 63

API Application Programming Interface. 64

CC Connected Component. 2, 4, 5, 9–14, 18, 22, 25, 26

CNN Convolutional Neural Network. 5, 11, 16, 17, 26

COCO Common Objects in COntext. 60, 61, 63, 100, 102, 112

CSV Comma-Separated Values. 46

CVC Computer Vision Center. 60, 63

DSTIL Deakin Software and Technology Innovation Laboratory. 4, 29

ETL Extract-Transform-Load. 42

HIT Human Intelligent Task. 63

HOG Histogram of Oriented Gradient. 15, 20, 25

HTML HyperText Markup Language. 42

ICDAR International Conference on Document Analysis and Recognition. 18, 23, 26, 60–63, 103, 105, 113

IoU Intersection over Union. 24

JSON JavaScript Object Notation. 46, 60, 61, 65, 100, 102

LBP Local Binary Patterns. 15, 25

LoP Likelihood of Purchase. xii, 36, 37, 39, 47, 48, 56

LPR License Plate Recognition. 2, 4, 67

MDE Model-Driven Engineering. 28

MLP Multilayer Perceptron. 15, 19

MSER Maximally Stable Extremal Region. 11, 12, 19, 25

NLP Natural Language Processing. 67

NN Neural Network. 2, 4–6, 9, 15, 19–21, 25, 26, 33, 42, 46, 56, 57

OCR Optical Character Recognition. 1, 2, 5, 9, 18–21, 25, 26, 65

PASCAL Pattern Analysis, Statistical Modelling and Computational Learning. 60, 61, 106, 108, 114

PNN Probabilistic Neural Network. 20

RBN Racing Bib Number. 2–5, 7, 9, 18, 20–22, 26, 29, 30, 33, 34, 39, 41, 53, 57

RLE Run Length Encoding. 61, 63

SaaS Software as a Service. 63, 64

SIFT Scale Invariant Feature Transform. 15

SUN Scene UNderstanding. 60, 62

SURF Speeded-Up Robust Features. 15

SVHN Street View House Numbers. 20, 60, 62, 115

SVM Support Vector Machine. 15, 16, 25

SWT Stroke Width Transformation. 10, 11, 21, 22, 25

TSR Traffic Sign Recognition. 2, 19, 20, 26, 67

UI User Interface. 43, 44, 52, 55, 56

UML Unified Modelling Language. 29, 40, 44, 46

VATIC Video Annotation Tool from Irvine, California. 64, 66

VOC Visual Object Classes. 60, 61, 106, 108, 114

XML eXtensible Markup Language. xvii, 42, 43, 46, 60–62, 103

XSD XML Schema Definition. 103

YAML YAML Ain't Markup Language. 46

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Ever since the camera and phone were unified into smartphones, we have seen an increasing interest for image understanding (specifically to identify the content of an image) but text recognition still faces challenges within images of unstructured scenes. While successes in character recognition have a long history with Optical Character Recognition (OCR) engines [122], these are typically applied under strict conditions (e.g., flatbed scanners for documents without distracting backgrounds). Once applied within the context of a natural scene, real-world discrepancies pose serious shortcomings, such as illumination conditions, viewpoint and perspective differences, blur and glare variations, geometric and photometric distortion, and differences in font size and style [64, 150]. Overcoming these issues has motivated a variety of techniques to realise potential applications that make use of text recognition at scale.

With the ubiquity of smartphone cameras, practical applications of natural image processing have increased. In the last two decades, we have seen the development of point-and-shoot product recognition [42, 132], object detection in videos [119], building recognition [130], image feature extraction to improve visual-based search engines [5, 90], and translation services of American Sign Language gestures [60]. Nonetheless, embedded text within images contains indexable data on the image's semantics [120]; if text extraction is therefore not robust, information extraction suffers.

Text detection robustness is a factor that severely limits a text recognition pipeline. Research in overcoming such limitations have been competed numerous competitions [54, 91, 92, 116], where robustness is the key focus in the image processing pipelines proposed. This focus was reiterated by Chen et al. [19], who state the primary prerequisite for text-based recognition (especially within natural scenes) is the text location must be robustly located.

As with any data processing pipeline, false negatives increase where early stages of the

pipeline fail, and therefore detection of these potential candidates must be robust. We can reduce errors, and thus robustness, in a pipeline where: (1) there are unwarranted stages (*excluding* unnecessary stages may also assist in reducing error cases) and (2) by piping through unmatched candidates to further pipelines, which can increase the detection.

Without the construct of robustness, we restrict these pipelines to very confined conditions, and its usefulness in products is not warranted. Therefore, the robustness of text extraction pipelines are imperative to gapping the semantic extraction of information from an image [120], and solving this issue can assist in applications of image processing and data indexing of content within images [36] of paramount proportions.

1.1 Background

This study focuses on character recognition in unstructured scenes (Figure 1.1): specifically, short, alphanumeric number sequences. Previous works present methods to extract these sequences in various areas, namely: License Plate Recognition (LPR) systems [1, 14]; Traffic Sign Recognition (TSR) [30, 72, 81, 114]; and, street number recognition, specifically a study by Netzer et al. [104], using Google Street View[1] to determine the numerical value of street numbers. Figure 1.2 highlights typical usage of these sequences.

Different applications apply varying methods to parse short alphanumeric characters. There are typically two stages of any parsing method: *detection* and *recognition*. Detection refers to locating possible candidates and recognition refers to the representation of the text itself. Detection techniques usually are categorised as either Connected Component (CC)-based or learning or texture-based. CC-based detection will typically use a set of distinct properties on the image to detect relevant areas (such as width, stroke and colour) while learning-based feed images into a classifier that can distinguish candidates from false positives. The recognition phase can typically be achieved using Optical Character Recognition (OCR) engines (such as Tesseract[2]) [7], machine learning algorithms [72, 76, 104] or deep Neural Network (NN) to classify the detected regions [61, 81, 115].

This study proposes the development of a learning-based detection and recognition pipeline using deep-learning neural networks within the context of unstructured photos, with a focus on marathon Racing Bib Numbers (RBNs)[3], as shown in Figure 1.1.

[1] `https://www.google.com/streetview/` last accessed 13 May 2017.

[2] `https://github.com/tesseract-ocr/tesseract` last accessed 14 May 2017.

[3] While referred to as numbers, some RBNs have alphabetic identifiers in them.

Figure 1.1: Four RBNs in a sample marathon photo.

(a) Successful LPR character segmentation [1]. *Left to right*: original image; region segmentation; character segmentation after negation, height and orientation measurements.

(b) Successful recognition of speed sign digits shown in Eichner and Breckon [30].

(c) Localisation of digits found from varying street view house numbers using the worker described in Netzer et al. [104].

Figure 1.2: Various sample alphanumeric sequences observed in literature.

1.2 Motivation

Detection is harder when the photo is unstructured. Early investigations in License Plate Recognition (LPR) systems were systematic in the subject material assessed; a detailed survey by [2] showed that they work best with consistent lighting, specific colour and typeface detection, fixed detection regions, and non-noisy backgrounds. When applied in the context of images with unstructured backgrounds, these systematic approaches begin to have limitations as the text components cannot be easily determined.

While further investigations in the area utilise enhanced Connected Component (CC)-based detection [19, 31, 118], performance is likely to degrade as image complexity increases [79]. This is especially relevant when text is geometrically obfuscated, such as malformed Racing Bib Numbers (RBNs) as worn on a marathon runner's torso. Malformed, in this sense, is caused by non-flat bib sheets that tend to follow the runner's body shape, in addition to images that are taken in dynamical contexts. Some studies have shown to overcome this by using facial recognition to find a more distinct candidate area [7], but nonetheless rely on a person's face to detect a number. Similarly, typical recognition techniques interpret text as segmented characters, rather than a single string, though there are exceptions such as in Zhu et al. [153].

We also identify subject prominence ranking within natural scenes as an area that has little exploration within literature. (For example, the prominence of a *specific* marathon runner within a scene of many runners.) Prominence ranking is an important field in the context of RBN recognition: runners typically choose not to purchase photos where they have been recognised in an image but are not in the foreground. There are also varying factors that influence purchase likelihood, such as face visibility, eye contact with the camera, and blurriness. An assessment into how the prominence of a runner can be ordered in hundreds of identified photos (based from their recognised RBN) can be used by use of a Neural Network (NN).

This study forms part of an industry project under the Deakin Software and Technology Innovation Laboratory (DSTIL). As a part of the research project, access has been made to a labelled dataset of hundreds of thousands of marathon photos.

1.3 Research Goals

This study aims to develop a processing pipeline that both detects and recognises RBNs on a marathon runner, and then ranks the prominence of each runner detected in the photo. The in-

tention is to explore the viability of artificial deep-learning NNs—such as Convolutional Neural Networks (CNNs)—in the pipeline. Previous studies in RBN recognition [7] and similar areas [30, 72, 131] were heavily heuristic and rule driven.

This primary aim is developed into three key objectives:

Goal 1: ***Detect RBNs using a CNN***

Literature has shown that heuristic-based detection algorithms (that are CC-based) are able to detect text within photos [19, 30, 79]. We propose to apply these rule-based techniques to a large labelled dataset within the context of RBNs, and contrast them against a learning-based detection and recognition algorithms (using NNs). By benchmarking a against existing libraries and open source tools, we explore if heuristic-based detection algorithms (focusing namely on CC-based detection) outperforms learning-based detection methods. For this goal the research question is framed as:

RQ1) Do CNNs detect RBNs with equal or higher recall and precision rates than CC-based methods?

Goal 2: ***Design a CNN that can recognise RBNs***

Typically, traditional alphanumeric sequence parsing can be performed by character segmentation, and then piping those characters into Optical Character Recognition (OCR) engines. In the context of marathon photos, we explore answers to the following:

RQ2) Does a CNN-based OCR approach outperform or is at parity with traditional OCR approaches with higher or equal recall and precision rates?

RQ3) Does a CNN-based OCR algorithm perform *without* the use of character segmentation?

Goal 3: ***Rank prominence of alphanumeric sequences***

Our research objective is aimed to compare if humans are always better at ranking the prominence of an RBN than a NN. We can therefore propose the followings research questions:

RQ4) Can a deep-learning NN be trained to rank marathon runners by prominence?, and if so

RQ5) Does a trained deep-learning NN rank prominence of a runner better or equal to a human?

1.4 Thesis Organisation

This thesis is organised into the chapters as outlined below. An appendix follows with additional supplementary material.

Chapter 2 - Background Provides an overview of prior studies broadly around the areas of number detection and recognition in image processing and artificial NNs.

Chapter 3 - Dataset Describes the dataset to be used, data treatment steps, possible techniques in closer depth to develop a number recognition pipeline, and explores ways to develop a metamodel of gathering large datasets.

chapter 4 - Processing Pipeline Discusses the proposed processing pipeline developed that satisfies the aims of this study.

Chapter 5 - Evaluation Strategies Discusses several evaluation strategies used to assess the accuracy of our processing pipeline.

Chapter 6 - Findings Outlines the method used for validation and presentation of our results and compares this amongst existing open source tools and pipelines presented in previous work.

Chapter 7 - Discussion Presents implications that were found from the results of our findings and limitations.

Chapter 8 - Conclusions and Future Work Draws a number of conclusions and alleviates gaps in the findings of this work by presenting future studies.

Summary

In this chapter we identified some shortcomings in text recognition, developed the context of the study—namely RBN detection. We discussed the general stages that exist for text parsing within natural scenes, detection and recognition, and introduced typical techniques that are applied in this context. We outlined the research aims this study achieves, and how the thesis is organised. The following chapter will detail applications of image processing, using neural networks for image processing, and outline what techniques have been used in previous studies to achieve this.

Chapter 2

Background

Text capturing within unstructured photos from the wild are typically achieved in two stages: detection and recognition. Detection techniques are classified as either CC- or learning-based. The recognition phase uses traditional OCR engines or, more recently, artificial NNs.

In this chapter, we survey different applications where RBN recognition (and related works) are investigated. Various detection and recognition techniques discussed in literature are detailed.

2.1 Detection Strategies

Text extraction strategies have seen continuing interest in the literature, with many comprehensive surveys assessing the state of the art [15, 63, 64, 82, 150]. It is widely demonstrated that if text within an unstructured scene is *detected* reliably, then existing OCR engines can suitably extract these characters [121] once they exist in a structured context; thus not every extraction pipeline needs to self-contain a recognition strategy if commercial OCR packages suffice. A survey into the two prominent detection strategies is given in Sections 2.1.1 to 2.1.2.

These two prominent strategies have a varied nomenclature: (1) the CC-based (or *region*-based) approach, that utilise different region properties (e.g., colour, edges, CCs) [19, 31, 59, 68, 77, 79, 88, 89, 117, 118, 127, 128, 151, 152] for unsupervised extraction; and, (2) learning-based (or *texture*-based) approach, which uses unique texture properties to supervise extraction text from its background [22, 24, 28, 45, 48, 74, 78, 84, 99, 108, 109, 133, 140, 148]. Some authors have proposed methods that mix both supervised and unsupervised techniques [8, 94, 103].

2.1.1 Connected Component-based techniques

Connected Component (CC)-based approaches generate separated CCs using properties such as stroke width, pixel colour and edges, typically applying geometric and texture filters to reduce false positives. Neighbouring pixels are then 'grouped' using an algorithm such as the one originally presented by Horn [52].

Previous work required the use of a scanning window [22, 62, 84] that is limited by a constant image scale and discrete orientations of the sliding (thereby preventing text strokes in non-linear directions). Subramanian et al. [127] overcame this limitation by implementing an algorithm to detect text strokes by scanning an image horizontally and looking for sudden changes in background intensity (Figure 2.1b). However, this algorithm assumes a darker text on a lighter background to find such intensity changes, and consequently there are numerous parameters that must be fine-tuned. Additionally, the algorithm is only able to detect horizontal text only, and detected strokes are not grouped into characters, words and sentences.

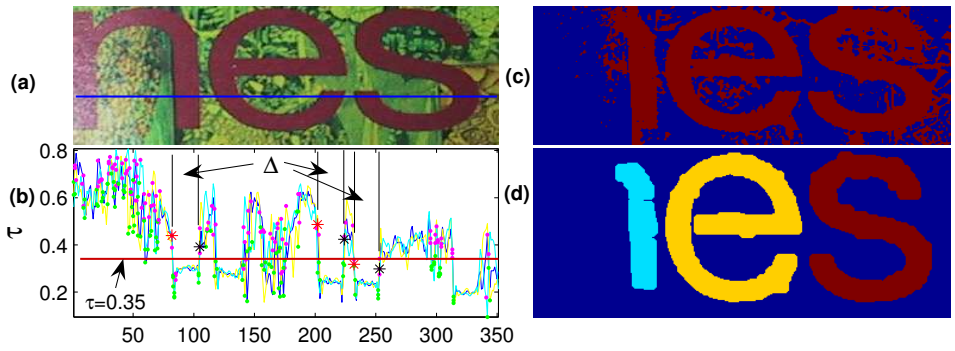


Figure 2.1: A study from Subramanian et al. [127] showed that stroke width could be determined from (a) the original image; (b) intensity plots of the image to determine stroke regions (τ is the intensity threshold and Δ is the stroke width); (c) the intensity at an optimal threshold; (d) the final thresholded image after morphological operations and CC analysis.

A study by Epshtein et al. [31] (and coincidentally Zhang and Kasturi [152]) built on the idea presented by Subramanian et al., and introduced the concept of Stroke Width Transformation (SWT), a local image operator that determines the most likely stroke of a given pixel by computing the per-pixel width. This was later expanded in Srivastav and Kumar [126]. The SWT approach overcame previous limitations by introducing a system that can detect text regardless of size, typeface, direction and language, making it one of the first widely cited multilingual text detection algorithms. Additionally, SWT overcame methods that required the use of an OCR filtering stage to reduce false positives [22, 23, 148]. A sample of SWT is shown in Figure 2.2.

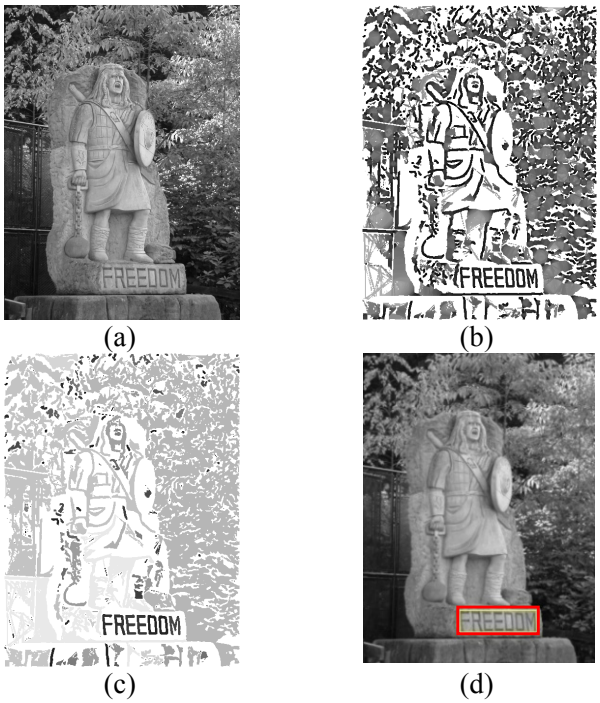


Figure 2.2: The Stroke Width Transformation (SWT) approach introduced in [31]. The original image (a) is converted to a binarised array with the most likely stroke width per-pixel (b), piping the information into geometric filtering (c) as text maintains fixed stroke width (excluding false positives such as foliage). The resulting detected text is shown in (d).

It is common to see edges computed from a raw image using the Canny-Edge Detection algorithm [13]. This was successfully applied in various CC-based studies [19, 31, 151]. While different researchers have exploited SWT and adapted it further [118, 152], when opposite edges are not parallel, the SWT forms candidates with holes appearing in stroke curves or joints. This is due to candidates formed by shooting rays from detected the edges along the gradient found, removing the rays if terminated by another edge pixel of a perpendicular gradient. Further limitations include undetected stronger highlights, blurry text, and text with a wide curvature.

An alternate approach that overcomes this limitation was introduced by Chen et al. [19], where the complimentary properties of Canny-Edges [13] and Maximally Stable Extremal Regions (MSERs) [95] were combined. MSER is a detection mechanism suited for region-based detection, is robust against varying viewpoints, scales and illuminations [98], and can be extracted from images efficiently [105]. A limitation of MSER is its sensitivity to image blur [98], but Chen et al. demonstrated that MSER can be edge-enhanced using Canny-Edges on a contrast-enhanced image (Figure 2.3), achieving comparable results to SWT presented by Epshtein et al. [31]. Multiple works have utilised MSERs in a wide range of applications, such as their use in teaching CNNs and real-time text extraction [46, 55, 72, 79, 149].

Figure 2.3: Extracting text from a natural image shown in Chen et al. [19]. *From left to right:* Detected Maximally Stable Extremal Regions (MSERs) of black-on-white objects; text candidates grouped to formed text lines after geometric and stroke width filtering; false positives rejected using text verification showing detected text in the blue box.

A significant requirement of all CC-based techniques are the requirements to cluster extracted components back together again. This, in turn, also helps to remove any false positives by removing properties that don't meet set criteria. Various proposals have been made:

- Epshtein et al. [31] use basic geometric filtering based on the stroke width detected and height ratios of candidates. Additionally, colours of candidates are averaged as it is expected that words be written in the same colour. These are then clustered into candidates pairs (of at least three letters), chained together if they share a similar direction.

- Zhang and Kasturi [152] investigate the spacial relationship and property similarity of two neighbouring candidates, computing their link energies to compute *text energy* (the probability a candidate is a true positive). The distance of the text energies are computed and, where beyond a set threshold, will be eliminated if not met. This is presented in Figure 2.4.

- Zhang and Kasturi [151] expand the use of graph spectrum, which has successfully been used in computer vision [113] to show image features in the form of a graph, use an adjacency matrix, then gather clusters of CCs based on the positive eigenvectors of the graph. This process is illustrated in Figure 2.5.

- Shivakumara et al. [118] propose the use of skeletal distance maps of a CC to remove small artefacts and reduce false positives. They define *simple* and *complex* CCs respectively as: (a) a single text string or false positive; (b) multiple text strings that are connected to each other. The skeletonisation process is shown in Figure 2.6.

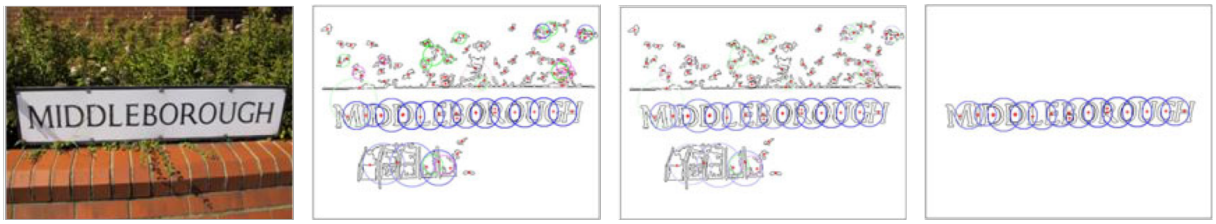


Figure 2.4: Using link and text energies for reconnecting character candidates shown in Zhang and Kasturi [152]. *From left to right:* original image; all link energies determined in a given image (note the false positives of background foliage); text energies calculated; all text energies greater than 0.5.

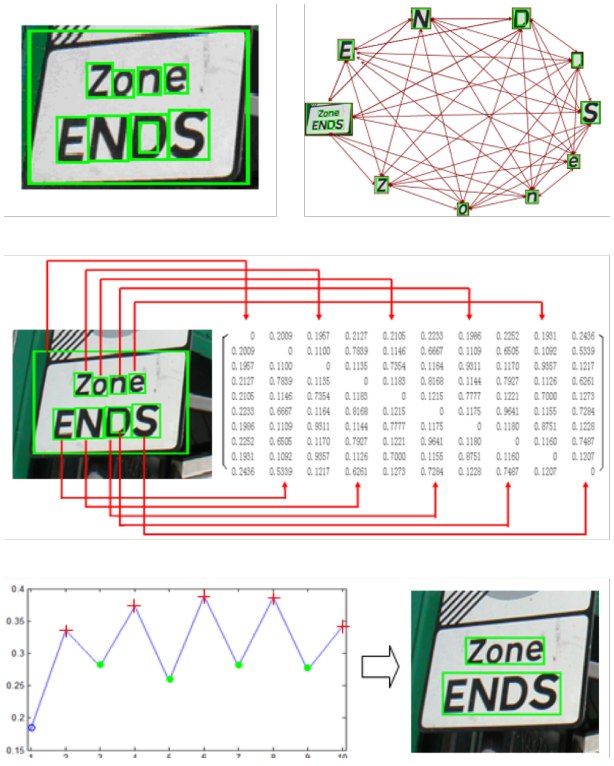


Figure 2.5: Process of grouping components via graph spectrum [151]. *Top-left:* 10 CCs detected. *Top-right:* generated graph from detected candidates. *Middle:* generated adjacency matrix. *Bottom-left:* Positive eigenvector resulting from the graph spectrum. *Bottom-right*: Resulting bounding boxes.

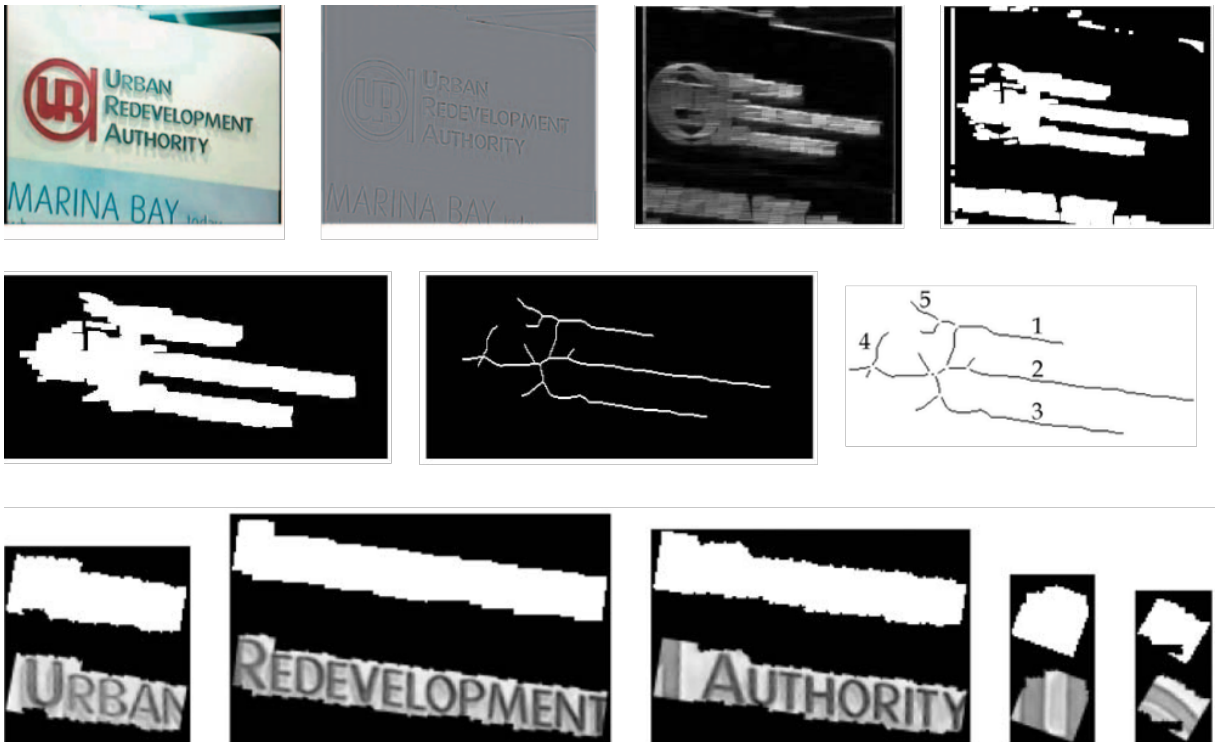


Figure 2.6: Developing a skeletal map using the process proposed by [118]. *Top row:* Original image is processed using Fourier-Laplacian filtering. A maximum distance map is developed and parsed through a morphological operation to remove smaller artefacts. *Middle row:* CC classification and further skeletonisation, showing five labelled subcomponents. *Bottom row:* The five sample subcomponents extracted from the skeletonisation process (in order). Note that subcomponents 4 and 5 are false positives.

2.1.2 Learning-based techniques

The learning-based approach has had a varied popularity over the years. While also referred to as a *texture*-based approach, it typically utilises learning-based methods to train a classifier using these textures, considering text as a special texture within the image. This is done by extracting certain features over a portion of the image (i.e., the texture) either via heuristics or machine learning. The texture is typically extracted by scanning at varying scales, then classifying areas of pixels based on certain features. A classifier uses these features to identify text from non-text to extract text from its background.

Early works in the area focused primarily on how to classify a region as either text or non-text [69, 78, 84, 123]. However, while it may be easy to procure training samples of text, it is often more difficult to procure non-text training samples [51, 129] due to the wide variance of 'non-text' samples, which need to be well-represented in the training set of the classifier.

A varied range of features have been utilised to mark textures. A recent feature made popular is the Histogram of Oriented Gradient (HOG), a object detection technique that utilises gradient orientation, first conceptualised in [97] and later coined by Freeman and Roth [37]. However, its popularity did not become widespread until [28], where Dalal and Triggs showed its applicability was shown on pedestrian detection. Later, HOG was shown to be useful in text detection by Hanif et al. [49]. Further features include: Local Binary Patterns (LBP) [106], which was shown to improve detection when combined with HOG [139]; wavelet energy [100, 136], which were applied to extract subtitles [45] and worked optimally with HOG in text extraction [108] (when compared to other features); Gabor filters [86]; Scale Invariant Feature Transform (SIFT) descriptors [90]; grey scale features [69]; Speeded-Up Robust Features (SURF) [5], edge map features [16]; and shape contexts [6].

Many of these features can be combined and fed into a single or multiple classifiers [45, 48, 49, 108, 133, 140, 148]. Example classifiers include Support Vector Machines (SVMs) [12, 26, 134], the Adaptive Boosting (AdaBoost) classifier [38]—and variants thereof ([39, 48, 124])—or NNs, such as Multilayer Perceptrons (MLPs), though Chen et al. [17] report that the use of SVMs showed better text texture verification than MLPs.

As shown in the wide range of features and classifiers, a main limitation of learning-based methods is the difficulty in selecting which combinations of features to use, the inability to detect sufficiently slanted text, as well as its reported high computational complexity due to the need to scan the image multiple time at different scales [31, 79]. Furthermore, it is typical for

these classifiers to required thousands of training images [18].

More recently though, deep-learning CNNs have been utilised for instance classification and per-pixel segmentation, as emphasised in Figure 2.9. While CNNs are a relatively old concept (see LeCun et al. [73]), they lost interest within image processing to Support Vector Machines (SVMs) and AdaBoost throughout the late 1990s and 2000s. In 2012, however, Krizhevsky et al. [70] sparked interest in their use by demonstrating far increased classification accuracy within images in the ImageNet Large Scale Visualisation Recognition Challenge [29], with only a few modifications made to the CNNs proposed by LeCun et al. more than a decade prior.

This has since led to the rise of enhanced CNNs such as FICS+++ [80], R-CNN [43], Fast/-Faster R-CNN [44, 110], and Mask R-CNN [50]. A comparison of two of these networks using images in the Microsoft Common Objects in Context [**?**] dataset are shown in Figure 2.9, with the Mask R-CNN framework illustrated in Figure 2.8. The rise of these recent developments have sparked interest in multi-language machine learning libraries, such as MXNet[1], which are able to combine these academic works for use in large-scale industry-focused production code [20].

These methods are increasingly improving per-pixel classification of objects within natural scenes. However, applicability of these deep-learning networks in the sole context of *text extraction* is yet to be widely explored, though a 2016 paper by Jaderberg et al. [58] did show the viability of a real-time text reading pipeline based on CNNs (Figure 2.7) that successfully extracts any text query from 2.3 million frames of BBC News footage[2].

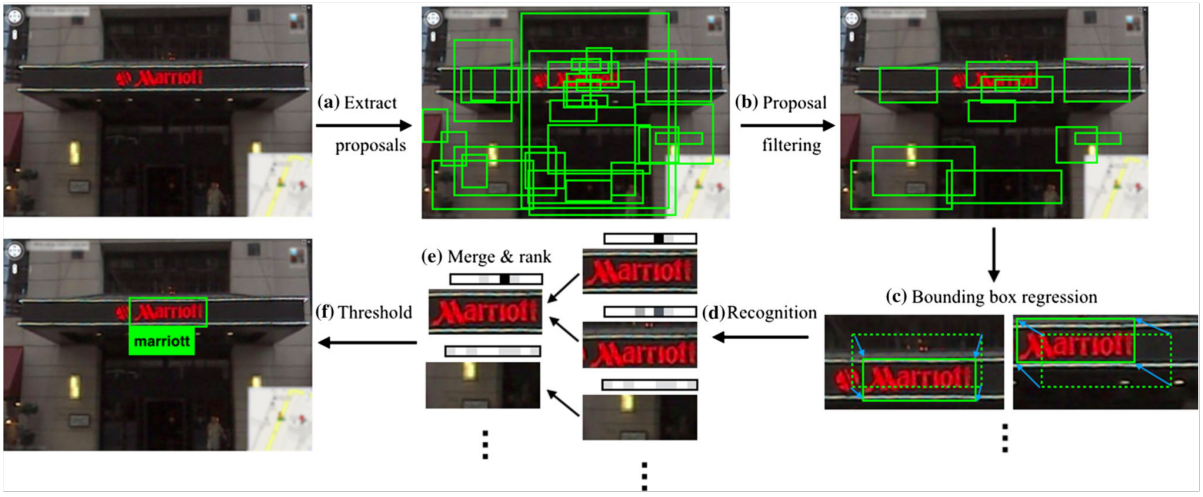


Figure 2.7: The pipeline for text extraction proposed in [58].

[1] `http://mxnet.io` last accessed 30 June 2017.

[2] `http://zeus.robots.ox.ac.uk/textsearch` last accessed 14 August 2017.

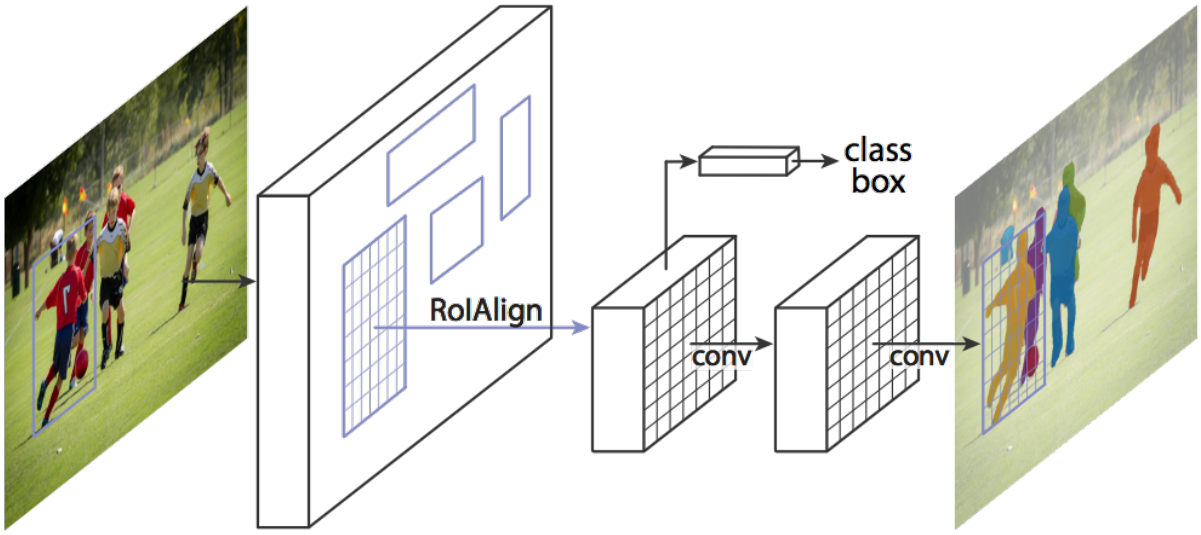


Figure 2.8: The Mask R-CNN framework for instance segmentation [50].

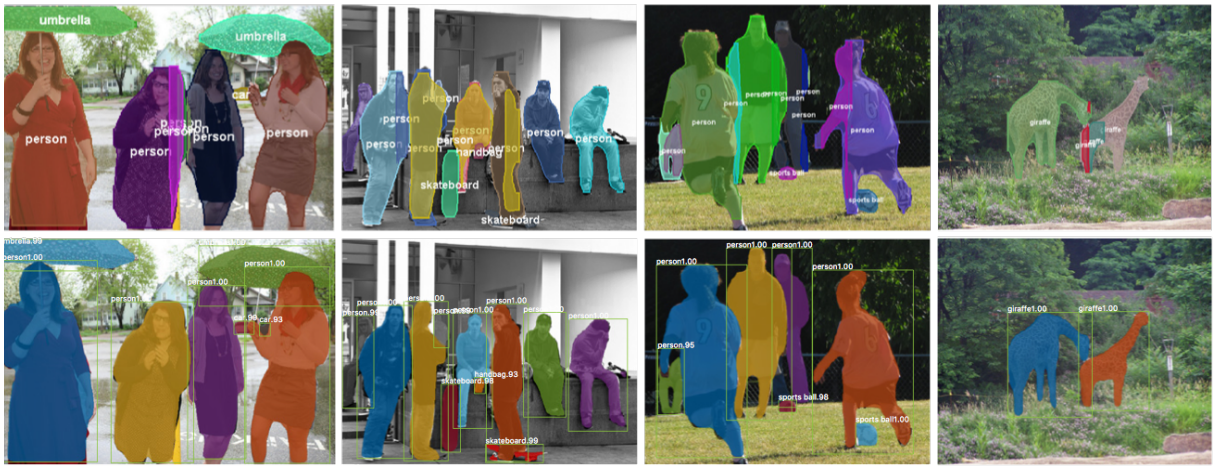


Figure 2.9: Use of CNNs have recently been shown to have accurate per-pixel object detection and classification. *Top row:* FCIS+++ [80]. *Bottom row:* Mask R-CNN [50]. Mask R-CNN outperforms FICIS+++ for overlapping instance segmentation.

2.2 Recognition Strategies

The International Conference on Document Analysis and Recognition (ICDAR) Robust Reading Competitions [65, 67, 91, 92, 116] broke down the issue of text extraction into two sub-problems: text locating and character recognition. Most of the literature discussed in Section 2.1 focused within the text locating sub-problem. For character recogntion, the *Focused Scene Text* word recognition task[3] received three entries in 2011 and four in 2013. In 2015, the recognition challenge was redesigned[4] using a new (and more challenging) *Incidental Scene Text* dataset of photos captured in-the-wild using Google Glass. The evaluation scheme uses the *Total Edit Distance* metric (described in [67]) and additionally the number of correctly recognised words for qualitative analysis. Table 2.1 summarises the top-scoring recognition rates from these competitions.

Table 2.1: Top-scoring word recognition results from ICDAR 2011–2015.

Year	Method	Dataset	Total Edit Distance	Correctly Recognised Words (%)
2011	TH-OCR System [87]	Focused	176.23	41.2
2013	PhotoOCR [10]	Focused	122.7	82.83
2015	MAPS [71]	Incidental	1128.0	32.93

It has been widely demonstrated that off-the-shelf commercial and open source OCR packages are able to correctly recognise text once the characters are extracted. In their conclusions of the ICDAR 2015 competition, Karatzas et al. note that the top performing methods will make use of commercial OCRs and conclude that conventional shape-based OCR engines can produce competitive results with pre-processed images.

This conclusion is emphasised in further works. The Open Source Tesseract OCR engine was used in Ben-ami et al. [7] to extract RBNs after preprocessed CC-based extraction. Similarly, leading commercial OCR engines used by Chen and Yuille [22] were able to achieve 93% recognition from binarised text regions after AdaBoost non-text classification, using ABBYY FineReader[5], TOCR[6] and Readiris Pro[7]. Similarly, Gatos et al. [41] used ABBYY FineReader and showed their extraction method showed a 50% improvement for indoor and outdoor scene images, an approach also applied by Chen and Yuille [22]. This is not to say that OCR engines

[3] See Challenge 2, Task 3 in [67, 116].
[4] See Challenge 4, Task 3 in [65].
[5] `https://www.abbyy.com` last accessed 3 July 2017.
[6] `http://www.transym.com` last accessed 3 July 2017.
[7] `http://www.irislink.com/readiris` last accessed 3 July 2017.

are always needed.

Recent interest in developing novel general character recognition strategies have also been investigated. Wang et al. [138] compared ABBYY FineReader with their novel PLEX approach and demonstrated that their text recognition system can outperform traditional OCR engines without the use of a text detector. This inspired more recent works: in 2016, Lee and Osindero [75] developed a lexicon-free photo OCR frameworks. Furthermore, application-specific recognition has been investigated using variant techniques.

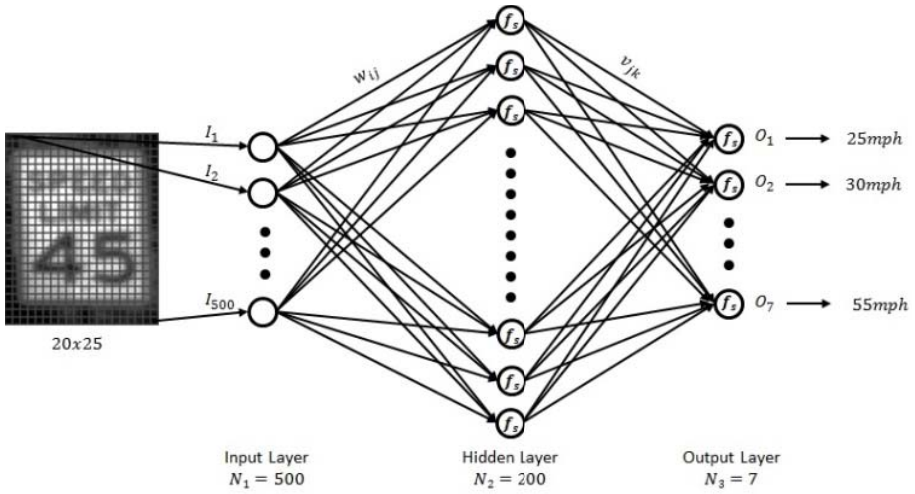


Figure 2.10: The artificial Multilayer Perceptron (MLP) NN designed in Kundu and Mackens [72] to recognise US-style speed limit signs.

A 2015 study into Traffic Sign Recognitions (TSRs) to detect US-style speed limit signs achieved recognition without the use of any OCR packages. In [72], Kundu and Mackens were able to extract a speed limit sign via the use of MSERs and template matching. The resulting detected signs were scaled to a grayscaled size of 20×25 pixels and fed into a Multilayer Perceptron (MLP) NN of 200 neurons in the hidden layer. The output layer of the network consisted of seven nodes, each representing the seven kinds of speed limit signs in US cities (25, 30, 35, 40, 45, 50, and 55 miles per hour). This architecture is shown in Figure 2.10. When trained with 13,289 images of text cases and 4,319 non-text cases, the results showed that their recognition classier was able to correctly recognise speed limit signs with an accuracy of 98.04%. Similar results were achieved using a feed-forward MLP in [30], using UK/Poland style speed limits scaled to 20×20 pixels (grayscale) and 12 output layer neurons ($10 \ldots 100$ kilometres per hour, the national speed limit sign, and non-sign neurons).

However, works in TSR systems that utilise networks are generally non-generalisable, and only work in a limited context (i.e., by classifying speed limit signs of known outputs). In our

context of RBN recognition, we have a known character output range of 36 possibilities: 0–9 and A–Z = 10 + 26. We do not consider lowercase letters.

Beyond TSR systems, however, we see the use of more generalisable networks: Netzer et al. [104] trained neural network to recognise street number characters from Google Street View (using the Street View House Numbers (SVHN) dataset) with higher precision and recall than that of HOG and the Tesseract OCR engine, showing that the applicability of NNs for recognition can outperform traditional means. Anagnostopoulos et al. [1] used a Probabilistic Neural Network (PNN) to recognise single characters of the same 36 possibility range (i.e., uppercase alphanumeric characters) corresponding to the input grayscale vector of 9×12 pixels ($9 \times 12 = 108$ input neurons) for a single character. Figure 2.11 illustrates the PNN architecture used in this study. Furthermore, investigations in comparing different architectures of NNs for this context is given in Lee and Osindero [75].

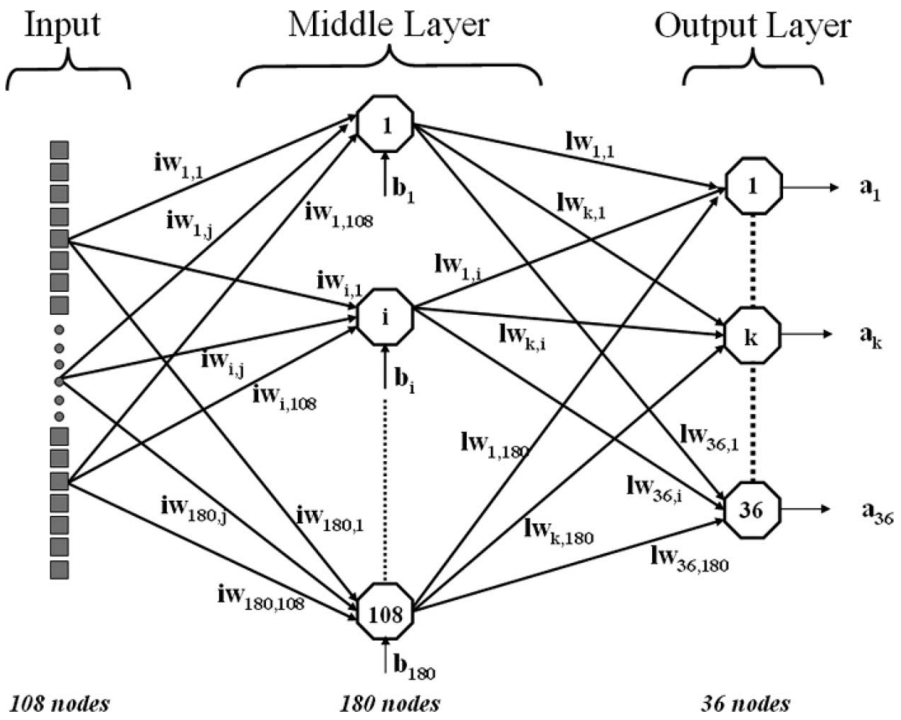


Figure 2.11: Anagnostopoulos et al. [1] developed the architecture of a PNN to determine a single character from within a license plate.

However, these pipelines are still entirely dependent on good detection strategies, and in the case where classifiers are used, quality training data must be supplied. The pipeline developed by Ben-ami et al. [7] for RBN detection (Figure 2.12) is dependent on quality facial detection: the OpenCV implementation [83] was used in this study. In order to detect the torso region—and thus detect the bib sheet itself—a heuristically-driven calculation is used to hypothesise

where the torso bounds ($T_b = T_h \times T_w$) are, given by the face height (F_h), face width (F_w):

$$T_b = (3 \times F_h) \times (\frac{7}{3} F_w)$$

The location of T_b is horizontally located at the centre of the face midpoint and vertically halfway below the face height ($0.5 \times F_w$). Hence, the dependency on these heuristics are heavily driven by a tolerant bounding box, which in itself depends on the face detection itself. Additionally, these heuristics are not always accurate—Figure 2.14 highlights a case where the face detection approach has missed the RBN entirely. As Fu et al. [40] have also noted, even if the face is detected, the bib sheet can be obfuscated (e.g., by another runner) or if the camera is capturing on the runner's side, the OCR engine or face-detection algorithm may produce poor results. In our study, we ignore these limitations for the intention of capturing prominent runners.

Furthermore, the use of Tesseract in the study meant significant filtering, separation and character alignment is needed for the OCR engine to read characters correctly (Figure 2.13). This is yet another step that can possibly be avoided via the use of well-trained NNs, as demonstrated in previous works. Investigation into applying such networks in this context (i.e., alphanumeric sequences on *human* subjects) is largely lacking.

Figure 2.12: The RBN recognition pipeline by Ben-ami et al. [7]. *From left to right:* Input image; face detection results in red and the estimated RBN region hypothesis in green; SWT of the hypothesis region; tag region detection in blue; tag region after digit processing.

Figure 2.13: Character processing in [7]. *From left to right:* A detected tag; separation via SWT CC analysis; binarised characters; CC analysis and filtering; separation and alignment.

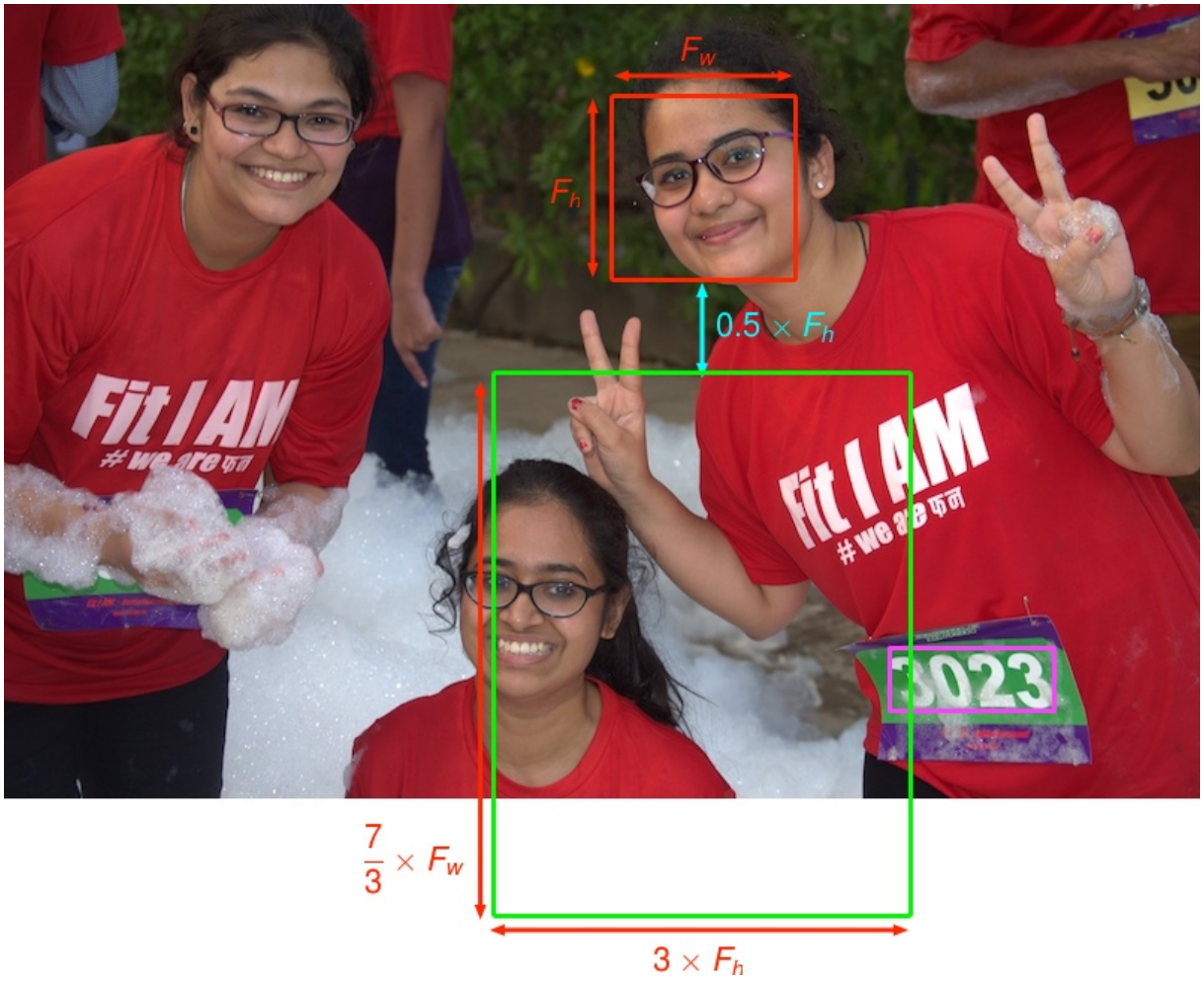


Figure 2.14: Cases where false negatives occur with the heuristic-based approach proposed in [7]. The expected RBN region (shown in magenta) is missed as the subject is leaning in the photo. Face detection in this photo uses the same OpenCV implementation [83] that was used in the study.

2.3 Metrics

Within the field of text detection, various evaluation protocols exist. For an extensive survey of these protocols, see [143, 147]. Throughout our survey, we note the standard the evaluation scheme for text extraction first proposed for use in image processing in the International Conference on Document Analysis and Recognition (ICDAR) competitions [65, 67, 91, 92, 116]. This scheme was designed to be easy to understand and compute, reward text extraction useful for natural scenes, and heavily punish trivial solutions. The intention behind these metrics were to develop a measure of 'robustness' a text extraction pipeline can achieve.

In the ICDAR competitions, two are used: the initial ICDAR 2003 evaluation protocol proposed by Lucas et al. [92], and, more recently, the *DetEval* evaluation protocol [118] (based on [143]), as used in the ICDAR 2011 and 2013 robust reading competitions. The simplicity and continued wide use of the ICDAR 2003 for text localisation evaluation is chosen over the *DetEval* approach, especially as it complements our use case.

2.3.1 Precision and Recall

Generally in information retrieval, the precision (p) and recall (r) metrics are used, first defined in the six evaluation criteria for information retrieval systems by Cleverdon et al. [25]. Precision refers to the proportion of relevant matches actually retrieved in the results, while recall refers to the proportion of relevant matches retrieved in total relevant instances. We use recall and precision metrics to assess the *effectiveness* of an information retrieval system [111].

In the context of image processing, systems that over-estimate are punished with a low precision score, while systems that under-estimate are punished with a low recall score [92]. Therefore, precision is the number of correct candidates (c) divided by the number of total estimates found (E):

$$p = \frac{c}{|\,E\,|}$$

And recall is defined as the number of correct estimates divided by the total number of ground-set truth targets (T):

$$r = \frac{c}{|\,T\,|}$$

However, it is not realistic for a given text extraction pipeline to *exactly* agree with the rectangle bounds manually tagged by a human. Lucas et al. [92] first proposed changes to these calculations to better suit their usage in the context of information extraction from within images. They adopt a more flexible notion of what a 'match' is. They define a new match measure,

the match area m_a, between two rectangles (i.e., the ground truth and the system's detected candidate) as the area of intersection of both rectangles divided by the minimum bounding box containing both rectangles (i.e., the union of both) [91–93]. This metric is otherwise commonly referred to as the Intersection over Union (IoU) [58, 65, 85]. Using $a(r)$ to denote the area of the rectangle, we can represent this as:

$$m_a(r_1, r_2) = \frac{a(r_1 \cap r_2)}{a(r_1 \cup r_2)}$$

This allows for a match value of one when the candidate is identical to the ground truth, and zero where the candidate has no intersection at all to the ground truth. Therefore, the best match, $m(r, R)$, of a rectangle r in a set of rectangles R is:

$$m(r, R) = \max \{ m_a(r, r') \mid r' \in R \}$$

Lastly, we can redefine the recall and precision metrics to be more forgiving in the image extraction context:

$$p' = \frac{\sum_{r_e \in E} m(r_e, T)}{\mid E \mid}$$

$$r' = \frac{\sum_{r_t \in T} m(r_t, E)}{\mid T \mid}$$

One issue with the calculation is that it assumes a rectangular bounding box. Words of non-rectangular nature, such as curved or rhomboidal text, are not well-represented.

2.3.2 The f-score

A common metric used when developing text extraction pipelines is the f-score, a single measure of quality that combines both precision and recall values. We are able to compute this metric using the standard measure across many studies, as contrasted in Table 2.2.

The f-score algorithm is given in the context of image processing in Lucas et al. [92]. Relative weights controlled by an α value of 0.5 give equal weight to both precision and recall metrics:

$$f = \frac{1}{\frac{\alpha}{p'} + \frac{1-\alpha}{r'}}$$

As Chen et al. [19] report, even when all text is correctly localised, it is likely that the f-score will vary between 0.8–1.0. This is because E boundaries are unlikely to match *exactly* with the manually labelled T boundaries. An illustration of this is phenomena is shown in Figure 2.15.

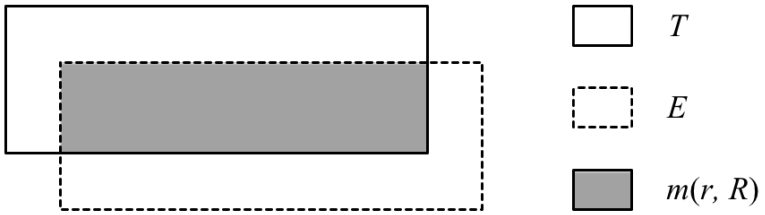


Figure 2.15: Overlapping areas of the ground truth targets, T, the estimated target boundaries E and the best match $m(r, R)$. (Adapted from [148].)

Table 2.2: A survey of text extraction literature, separated into CC- and learning-based detection methods.

Ref.	Year	Scientific Background		Precision	Recall	f-score	Dataset(s)	Performance	
		Detection	Recognition					Platform	Time (s)
[7]	2012	*CC-based*: SWT; Canny-Edges; Binary Conversion; Geometric Filtering; CC-Alignment	OCR (Tesseract)	0.65	0.62	0.63	N/A	N/S	N/S
[19]	2011	*CC-based*: Canny-Edges; MSER; SWT, Geometric, Template Matching	OCR (N/S†)	0.73	0.60	0.66	[91, 92]	2.5 GHz	0.20
[79]	2012	*CC-based*: MSER; geometric and SWT filters; skeletal distance mapping	N/S	0.59 [92] 0.59 [116]	0.59 [92] 0.62 [116]	0.59 [92] 0.61 [116]	[91, 116]	N/S	N/S
[152]	2011	*CC-based*: character detection with HOG and SWT; link energies via spacial relationships between characters	N/S	0.73	0.62	0.67	[92]	N/S	N/S
[118]	2011	*CC-based*: Fourier-Laplacian Filtering; clustering based on maximum distance via K-Means; skeletal distance mapping	N/S	0.76 [92] 0.81 [54]	0.86 [92] 0.93 [54]	0.81 [92] 0.87 [54]	[54, 92]	2.0 GHz	7.80
[31]	2010	*CC-based*: Canny-Edges; SWT; Modified CC algorithm; Geometric Filtering	OCR (N/S)	0.73	0.60	0.66	[91, 92]	N/S	0.94
[151]	2010	*CC-based*: Canny-Edges; HOG; geometric filtering; graph spectrum	N/S	0.67	0.46	0.55	[92]	N/S	N/S
[148]	2005	*Learning-based*: SVM to reject non-text; wavelet movement; HOG; OCR filtering	N/S	N/S	0.97	N/S	[54]	1.6 GHz	8.30
[108]	2010	*Learning-based*: Waldboost Classifier; HOG; LBP; Gabor Wavelets	N/S	0.56	0.70	0.68	[92]	3.4 GHz	0.37
[45]	2004	*Learning-based*: Wavelet transformation; feature estimation; pixel-block classification	N/S	0.87	0.90	0.88	[53]	N/S	N/S
[48]	2009	*Learning-based*: Mean Difference, Standard Deviation and HOG features; AdaBoost and CAdaBoost classifiers; NN-baed localiser	N/S	0.25	0.35	0.35	[92]	2.2 GHz	N/S

†Not Specified

Conclusion

In this chapter, we have presented a survey of literature in various application contexts: RBN and TSR recognition, recognition of alphanumeric sequences 'in the wild', and additionally object instance segmentation. We also present the varied range of techniques used to both detect and recognise the text, using both heuristic-based and NN-based approaches. This said, we acknowledge that datasets within the survey are not always recent (ranging as far back as 2003) due to tendencies for researchers to use the more popular (though aged) ICDAR datasets.

The state of the art of learning-based detection approaches such as CNNs for have gained wide popularity, albeit for object segmentation. These approaches are yet to be applied within the context of alphanumeric sequences. Recent years have had a heavier focus on heuristic-based detection strategies using CC-based methods, while a majority of learning-based detection methods have had far fewer recent investigations. Furthermore, recognition of characters using NNs are not yet widely used, and off-the-shelf OCR packages are still standard.

Chapter 3

Dataset

How do we capture large and annotated datasets to train an Artificial Intelligence (AI) model at scale? What is an annotation, and how can we constrain them? These are important questions when supplying data to train graphics-based AIs. In this chapter, we discuss a theoretical data tagging metamodel that enables us to concisely describe the architecture of how quality labelled data is captured for these purposes. Additionally, we discuss a partial implementation of this metamodel that we have used to gather our training dataset.

3.1 A Data-Capturing Architecture

Why should we care about the architecture of our dataset capturing, so long as the AI is well-trained? This question fundamentally leads to the *provenance* or *lineage* of our data: not all training data is initially perfect. At times, we will need to modify our training data, or in the cases of some AI systems, training data is re-fed back into the system from the AI itself.

Hence, it is imperative to understand how the transition and flow of data occurs [11, 27, 56] in these AI systems. Where does it comes from? How is it derived and updated, and how might this affect our trained AI models over time? A lack of understanding in training data provenance for large-scale AIs implies difficulty in sourcing the cause of errors due to these mutations. It may be a difference in the data training format, or a mismatch of values on the same training data. Overcoming this without systematic record-keeping is challenging.

We therefore propose a grammar to describe systematic approach of a data-capturing architecture. As such a grammar is textual, it can be version controlled: therefore data provenance is recorded, which allows us to source how the mutations in training data can occur and and what effect this has our AI systems. From an exploratory analysis of developing this architecture (Section 3.1.1), we propose a theoretical and generalisable metamodel that can be used to

capture any form image data for training purposes (e.g., frames of a video, images of natural scenes). This metamodel is largely inspired by the works of Wickham [141, 142] and Moody [101].

3.1.1 Methodology

To understand the methodology on how we captured our dataset, we must first introduce the three key notions behind Model-Driven Engineering (MDE): technical spaces, models and systems. A system is a concrete "group or set of related or associated elements perceived or thought of as a unity or complex whole" [107]. Technical spaces were introduced Ivanov et al. [57] as a model management framework based on algebraic structures (e.g., trees, (hyper)graphs, categories). Technical spaces are usually based on a three-tier conjecture: metametamodels, metamodels and models. Whereas a model is an abstract representation a concrete system of specific purpose, a *meta*model, in contrast, describes the way to describe those models. A *metameta*model can be used to describe the representation structure of our metamodels and defines a type system that supports all underlying layers [9]. Figure 3.1 captures these concepts in further detail.

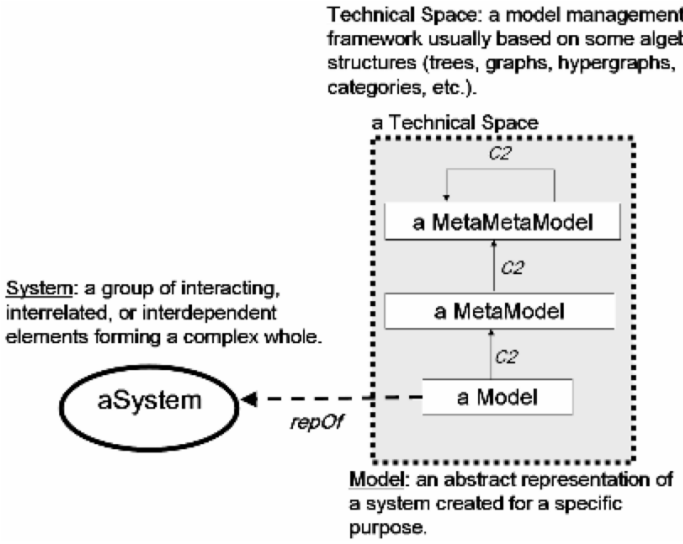


Figure 3.1: Systems, models and technical spaces. (From [9].)

We captured these concepts in a prototypical exploration process, first by developing a prototype system iteratively with a data tagging team who manually curated the data. Once this prototype was stable, we developed a model (Section 3.1.2) around the prototype. We generalised the system we had developed, expanding from a model we had designed that was marathon-specific, into a more generalisable metamodel (Section 3.1.3). We represent our process as as a

Unified Modelling Language (UML) activity sequence diagram, shown in Figure 3.2.

The initial phase in our dataset development was to create a prototype system with the specific purpose of tagging our marathon runner context (using the dataset provided by the industry client). We name this partial implementation *Argus*[1,2]. Refer to Section 3.4 for more about this implementation.

To develop this initial model (and thus to then what our metamodel would be), we began by discussing what requirements were needed—that is, the key features that we thought were necessary for extracting from our image. Five features were decided: (1) the crowdedness of the photo, (2) the visible bib sheets within the photo, (3) the faces corresponding to those bib sheets, (4) the prominence of runners of the photo and (5) the colours of runners' clothing in the photo.

Once an initial prototype was developed, we conducted informal usability tests with members from DSTIL, which captured minor flaws in the workflow (namely required conditions that were missing in annotations) as well as general usability enhancements. Once the internal testing had concluded, we developed instructional video guides on how to use Argus, and then deployed the tool to a data tagging team that made the data available to us.

After deployment, we ran four iterations of tagging with our external data tagging team. We use the term *iteration* to refer to the process of deploying Argus to the data tagging team, capturing a set of photos from various races, and sending the data back for quality assurance. These iterations consisted of photos from different marathon races in our dataset. (To ensure variance in tagging, there were some races at night and some races with alphanumeric components in the RBN.) After each iteration we assessed the tagging for quality. Feedback identified further restrictions that needed to be placed on Argus as poor approximations or incorrect tagging would cause our AI to learn poorly.

The following issues were identified:

- face region was too far away from the bib region,
- face region was overlapping the bib region,
- RBNs had spaces,

[1] `http://www.deakin.edu.au/~ca/argus` last accessed 5 July 2017.

[2] Whose name is inspired by the *all-seeing* giant of the same name from Greek mythology: "With his multiple sets of eyes, Argus could see nearly everything in his vicinity". See `http://www.loggia.com/myth/argus.html`.

- misidentified RBNs where the alphabetic ‘I’ was tagged as the numeric ‘1’.

Furthermore, we added geometric restrictions and conditions into the tool to prevent these errors from occurring at all (i.e., ensuring faces could only be marked above and horizontally near bibs). Some of these issues are highlighted in Figure 3.3.

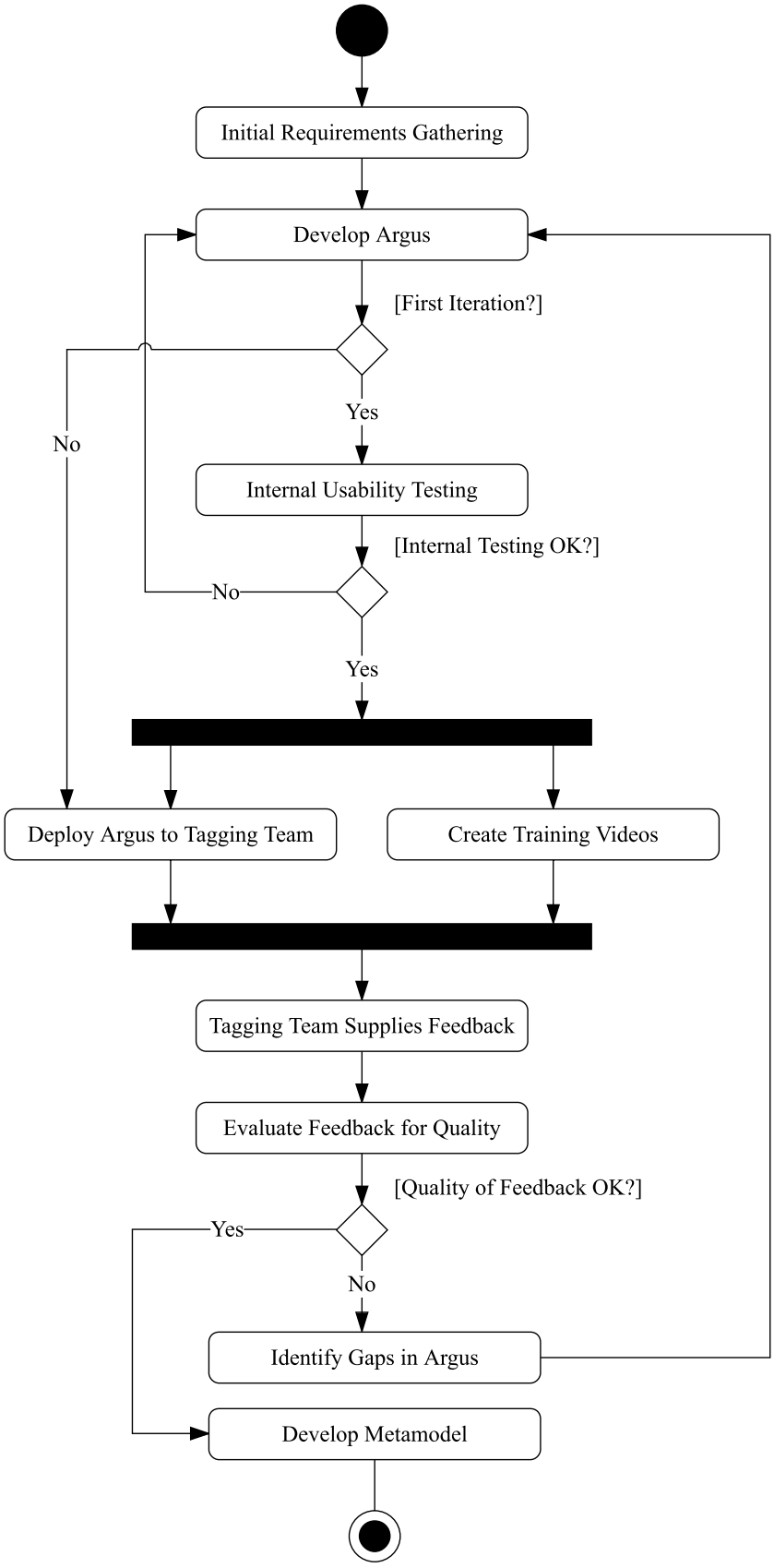


Figure 3.2: An overview of the methodology used to discover our metamodel.

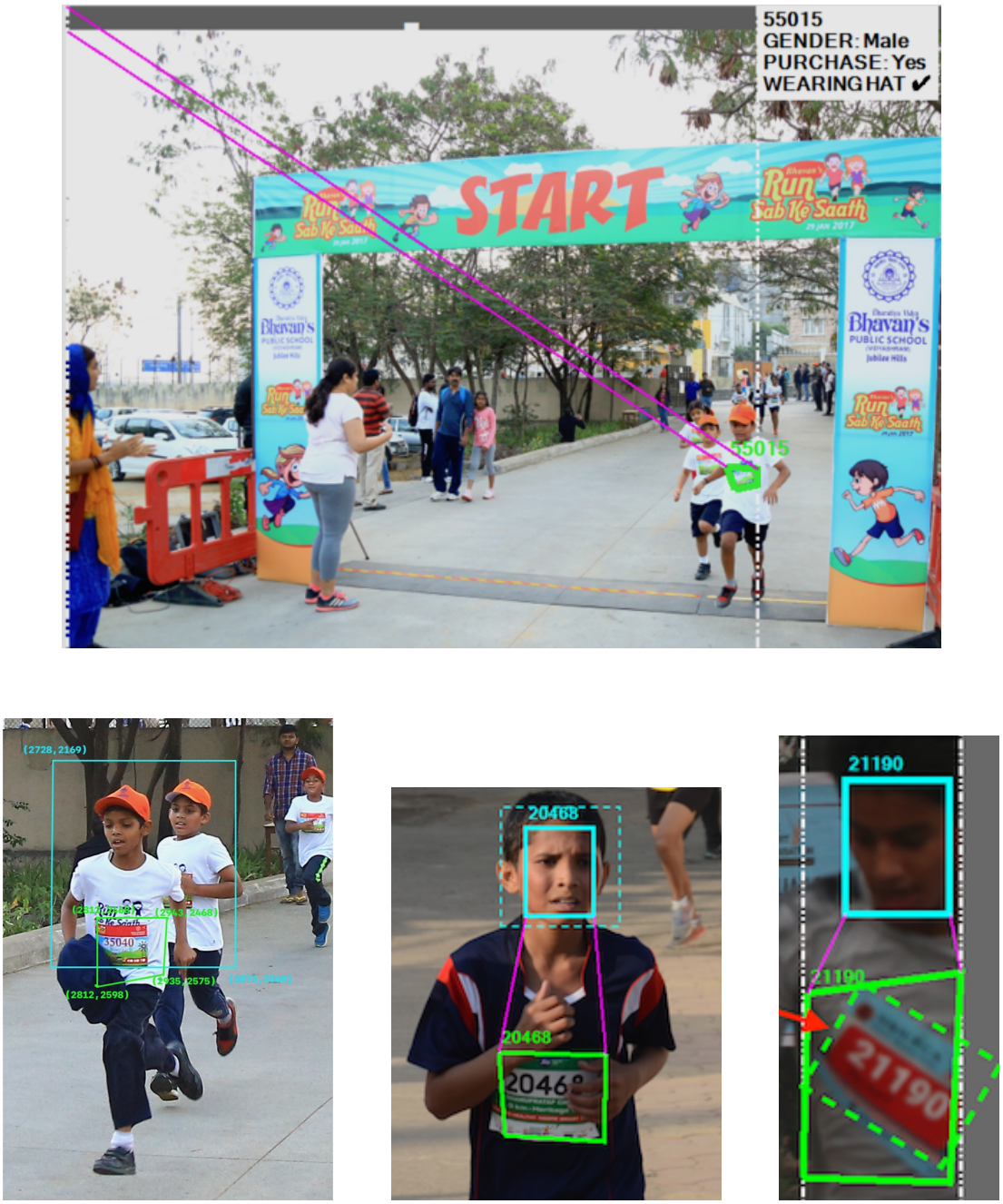


Figure 3.3: Issues identified from rounds of external testing to the tagging team. Cyan lines indicate face regions, lime lines indicate bib regions, magenta lines indicate bib-to-face guidelines. *Clockwise:* Face region is too far away from bib region; bib region is poorly marked (dotted lines expected); face region is too small (dotted lines expected); face region overlaps bib region.

3.1.2 What to Capture?

What features do we want to capture from images within our dataset? To answer this question, we need to consider what information we see as relevant in a standard marathon photo (such as Figure 1.1). There are two features to describe components of the entire photo that we call *Image-level* features:

1. whether or not the photo is considered as **crowded** (TRUE or FALSE), and
2. an *optional* collection of **runners** in the photo, given that the photo is not crowded.

(a) A crowded photo.

(b) A photo where all RBNs are cropped.

(c) A photo where there are no RBNs present.

Figure 3.4: Image-level features. In (a), the *PhotoCrowded* annotation would be marked as TRUE. Note the obstruction of all RBNs in this photo. In (b) and (c), *PhotoCrowded* is FALSE, but there are no *Runners* to tag.

We train a NN based on photos that are not crowded; photos that are considered crowded are typically not desirable as runners prefer photos where they are the key subject. We therefore discard these photos to remove any bias on the network. We can only tag runners on the condition that the photo is not marked as crowded (Figure 3.4a). Additionally, in some photos, all RBNs are missing within the photo (e.g., hidden behind other runners, cropped out of view) and some photos contain no runners at all (Figures 3.4b and 3.4c). We therefore identify this

as an *optional* collection as we must associate an RBN to a runner—the photo is not crowded, but there are no runners to tag, thereby making the annotation optional. Thus, an *implicit attribute* exists with such a collection: the *count* of the runners in a photo is something we can automatically count.

We identify the following two annotations of what we would label for a given runner's bib sheet. This is summarised in Figure 3.5. Within this feature we identify the following annotations for RBNs:

1. a polygon around the runner's **bib sheet** that contains the x and y coordinates, given that there are exactly four vertices of this polygon
2. a string with the runner's **RBN**, once the bib sheet is tagged (exists).

A further feature that is important is the runner's face region (Figure 3.6), which could compromise of five annotations:

1. a rectangle around the runner's **face** that contains the two x and y coordinates of the two opposite vertices, given that the $bottom$ of this rectangle are above the top of the bib sheet,

and once this face bounds has been tagged, more annotations can be extracted:

2. whether or not the runner is **wearing** a **hat** (TRUE or FALSE),
3. whether or not the runner is **wearing (sun)glasses** (TRUE or FALSE), and
4. the **gender** of the runner (MALE, FEMALE, or UNSURE).

The bib sheet polygon would contain four *explicit attributes* required as input by the tagger: the coordinates of its four vertices, $\{(x_1,y_1),(x_2,y_2),(x_3,y_3),(x_4,y_4)\}$. Similarly, there are two explicit attributes required for the runner's face rectangle: the coordinates of the top left and the bottom right of the rectangle (i.e., the two opposite vertices) of the runner's face bounds, $\{(x_1,y_1),(x_2,y_2)\}$. As both of these are shapes, we can automatically calculate the implicit attributes from these coordinates: $\{top, left, bottom, right, width, height\}$.

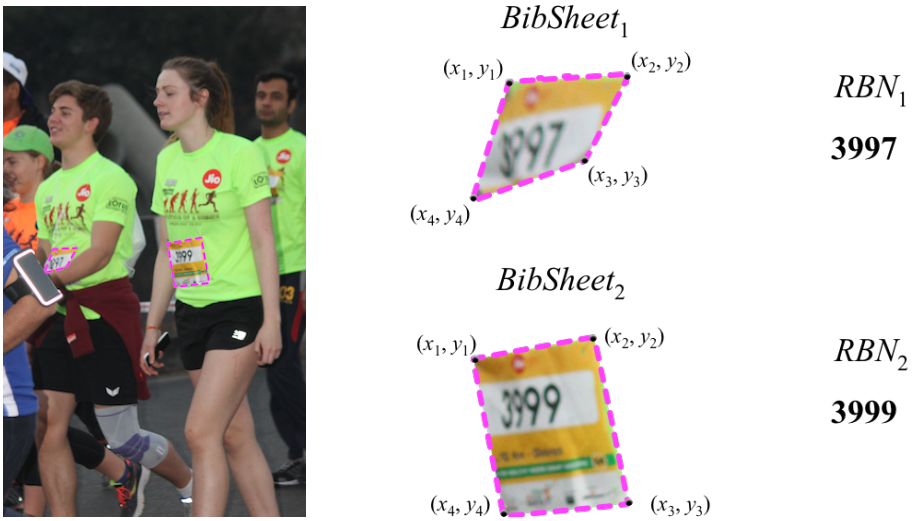


Figure 3.5: The bib segment-level feature. *From left to right*: The manually annotated bib sheets in the original photo (outlined in magenta); the relative *BibSheet* annotations with the four respective vertices; the detected *RBN* input strings for the corresponding *BibSheet*.

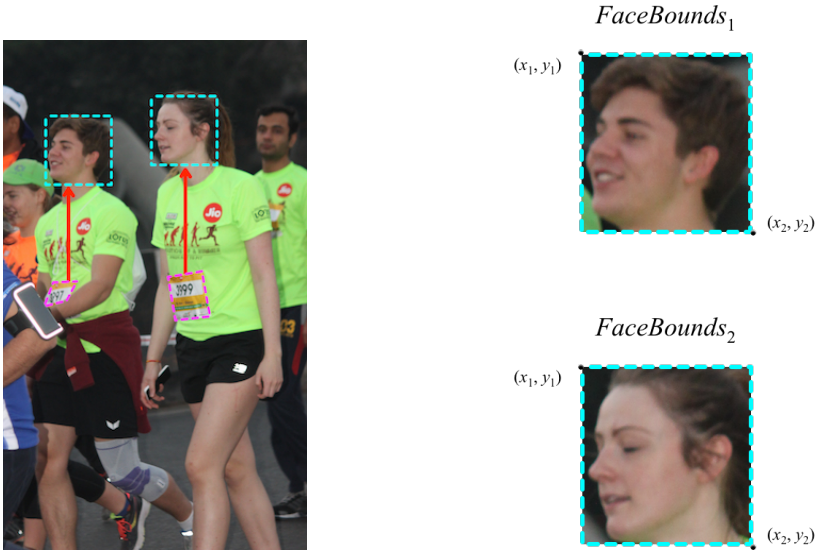


Figure 3.6: The face segment-level feature. *Left*: The manually tagged face regions (cyan) that comply with the conditions that the bottom must be above the top of the bib (red arrows). *Right*: the relative *FaceBounds* annotations with the two opposite vertices. Other annotations in this feature would be: $WearingHat1,2 = WearingGlasses1,2 = \text{FALSE}$, $Gender_1 = \text{MALE}$, $Gender_2 = \text{FEMALE}$.

We can now identify another feature, the prominence of the runner within the photo. These would consist of three annotations:

1. whether or not the runner's **face** is **visible** (TRUE or FALSE),
2. whether or not the runner is **blurry** (TRUE or FALSE),
3. the **LoP** that the runner buys the photo (NO, MAYBE, YES),

A runner's face is considered 'invisible' if it is obstructed by another object, cropped out of the image, or if the runner is looking down. See Figure 3.7. Runners are more likely purchase a photo if they are looking at the camera, and likewise if they are blurry in the image; these therefore have a significant impact on their prominence. We also define the Likelihood of Purchase (LoP) value as a qualitative metric. We ask the data tagger to 'picture' themselves as runner, and if so, we ask if they would purchase it. We then use this metric to train positive samples of good photos (LoP = YES) and samples of bad photos (LoP = NO). Where MAYBE is provided, the runner is ignored to prevent training the network with indeterminate samples. Refer to Figure 3.8 for examples of the varying LoP values.

Another feature we can capture is a way to identify runners by the colour of their clothing. (Within our dataset, various clothing items of distinct colours are given to runners for particular races.) These features comprise of four annotations:

1. an *optional* **colour** of the runner's **hat**, given that they were annotated as wearing a hat,
2. the **colour** of the runner's **shirt**,
3. an *optional* **colour** of the runner's **shorts**, and
4. an *optional* **colour** of the runner's **shoes**.

The colour of a runner's shirt is required, as we expect that a bib would be detected on the shirt of a runner (which would be tagged). The other colour annotations are all optional, as it is likely that some of these clothing items may be cropped out of the photo or is not visible. Furthermore, the hat colour can only be specified on the condition that the tagger has marked the runner has wearing a hat.

We can segment groups of the annotations we extract into features of two categories: annotations that feature at the *image*-level and those that do not, which we call *segment*-level features. The image-level features are those which apply to the entire image (i.e., *PhotoCrowded*,

(a) Face obstructed

(b) Face cropped

(c) Face looking down

Figure 3.7: The *FaceVisible* plays a significant impact on the prominence feature. Above are cases where the face would be marked as not visible.

(a) LoP = YES

(b) LoP = MAYBE

(c) LoP = NO

Figure 3.8: Various examples of the *LoP* values. In (a), the woman is clearly posing and is the primary subject of the photo. In (b), the runner is in focus in a pose, but is behind another runner and there are other runners behind him. In (c), the woman is out of focus, is blurry, and this photo has very poor lighting conditions.

Runners). Those that apply at the segment level can be grouped into what types of features we are extracting: the runner's bib, face, their prominence and their colour identification.

In summary, we have identified a total of 5 features of 15 annotations, which are summarised within Table 3.1. Each of these annotations can be classified with a name and type, conditions for the annotation to be valid, dependencies on other annotations to exist before the annotation can be made, explicit attributes which the tagger must specify, implicit attributes which we can automatically compute, possible values which limit the range of data for that annotation, and whether or not the annotation is optional.

Table 3.1: A summary of the annotations we wish to capture from our dataset. Image and segment-level features are separated using the double line.

Feature	Name	Type	Conditions	Dependencies	Explicit Attributes	Implicit Attributes	Possible Values	Optional
Image-Level	*PhotoCrowded*	Boolean	None	None	None	None	{TRUE, FALSE}	No
	Runners	Collection	$\{PhotoCrowded = \text{FALSE}\}$	None	None	$\{count\}$	N/A	Yes
Bib	*BibSheet*	Polygon	$\{vertices = 4\}$	None	$\{x_1 \ldots x_4, y_1 \ldots y_4\}$	$\{top, left, bottom, right, width, height\}$	N/A	No
	RBN	Label	None	*BibSheet*	None	None	N/A	No
Face	*FaceBounds*	Rectangle	$\{bottom > BibSheet_{top}\}$	*BibSheet*	$\{x_1, y_1, x_2, y_2\}$	$\{top, left, bottom, right, width, height\}$	N/A	No
	WearingHat	Boolean	None	*FaceBounds*	None	None	{TRUE, FALSE}	No
	WearingGlasses	Boolean	None	*FaceBounds*	None	None	{TRUE, FALSE}	No
	Gender	Category	None	*FaceBounds*	None	None	{MALE, FEMALE}	No
Prominence	*LoP*	Category	None	*FaceBounds*	None	None	{NO, MAYBE, YES}	No
	FaceVisible	Boolean	None	*FaceBounds*	None	None	{TRUE, FALSE}	No
	Blurry	Boolean	None	*FaceBounds*	None	None	{TRUE, FALSE}	No
Colours	*ShirtColor*	Color	None	*FaceBounds*	$\{red, green, blue\}$	None	N/A	No
	ShoeColor	Color	None	*FaceBounds*	$\{red, green, blue\}$	None	N/A	Yes
	ShortsColor	Color	None	*FaceBounds*	$\{red, green, blue\}$	None	N/A	Yes
	HatColor	Color	$\{WearingHat = \text{TRUE}\}$	*FaceBounds*	$\{red, green, blue\}$	None	N/A	Yes

3.1.3 Describing the Metamodel

In the example above, we describe a basic model which was derived using information discovered from Argus' incremental development. In this section, we generalise this information and develop a metamodel from that model. This metamodel is represented as a UML class diagram, shown in Figure 3.9. A more extensive overview of the metamodel is provided in Appendix B.

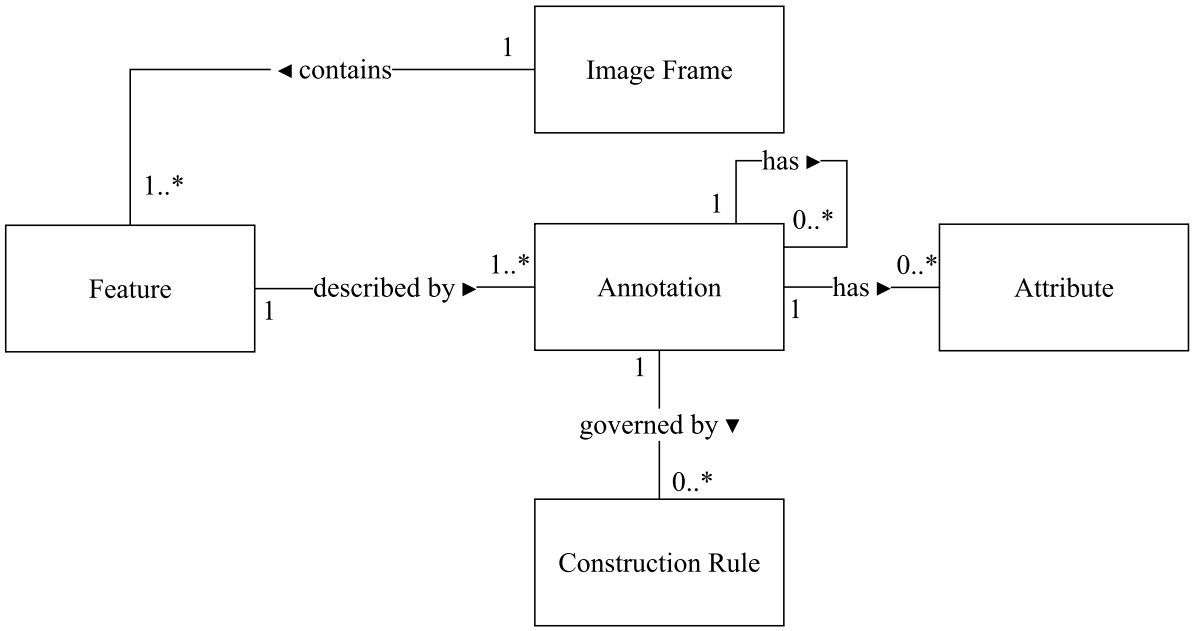


Figure 3.9: A class diagram of our proposed metamodel.

At the centre of this metamodel is an annotation, which (collectively) describe a feature within an image. Annotations may contain additional attributes to further extend what information they contain, and are governed by a set constrictions which we call 'Construction Rules'.

These form layers of data within images that we annotate, which can be represented in a hierarchical data format (e.g., a tree) that describes the layering of a fully annotated photo's features.

We describe each of these classes in the following sections.

Image Frame An image frame is any visual representation of a graphic. We extend the possibilities of our metamodel beyond just static images—these frames may be those found in videos also.

Feature Image frames contain a certain number of features. These features are the key 'concepts' within the image that we want to capture. There are two main features: (1) image-level

features, and (2) segment-level features. Image-level features are those that apply to the entire frame and do not exist in a broken down context—we can't break down these features into their own subjects. This contrasts to segment-level features, which are features that only apply to a segmented region of the image. We identified four segment-level features in our example (all of which relate to one runner): Bib, Face, Prominence and Colours.

Annotation Annotations are simply a set of feature descriptors that describes what the feature is compromised of. From Table 3.1, we describe a simple type system consisting of the following: *Boolean*s, a restricted form of the *Category* type (where possible values are $\{\text{TRUE}, \text{FALSE}\}$); *Polygon*s and *Rectangle*s, generalised into a high-level *Boundary* type; and *Colour*s, represented as an RGB represented hexadecimal *Label* (a generic labelling type). This only leaves our *Collection* of runners: a data type which encompasses multiple annotations. Attributes can therefore be described by a relatively simple type system that is generalisable from *Labels*, *Boundaries*, *Collections* and *Categories*. We further develop this into a more extensive type tree (Figure B.1).

Construction Rules Construction Rules govern how attributes can be made. We break these down into four types: Dependencies, Conditions, Optionality and Restrictions (Figrue B.4). Some annotations are dependent on others existing—these are rules that we name *Dependencies*. For instance, we cannot annotate an RBN if there is no *BibSheet* annotated. Likewise, a *FaceBounds* is dependent on where the *BibSheet* is, and so the *BibSheet* must be marked up first. These dependencies add order in which features are being extracted, which is important to the tagger marking up the photo. A *Condition* is a construction rule that disallows a data tagger to create an annotation until a condition is met by the annotator. For instance, we can never have a *BibSheet* above a *FaceBounds*, so when we can specify this as a condition. These are not bound to just geometric constraints; we also see that we cannot tag *Runners* if the *PhotoCrowded* annotation is marked as TRUE. Not all annotations are needed—in these cases the annotation is restricted by an *Optional* construction rule; by default, all annotations are required, but we can specify these to be optional using such a rule. Lastly, *Restrictions* exist to prevent invalid data from being tagged. In our example, we implemented two restrictions in Argus: (1) a face region must be tagged within an aspect ratio of $3 \times \mathit{BibSheet}_{width} : 5 \times \mathit{BibSheet}_{height}$, and (2) the opposite edges of a face region must be at least 15 pixels apart. This helps prevent issues shown in Figure 3.3.

Attributes Attributes exist to as a means capture further information about an annotation. We represent attributes in our metamodel as either implicit or explicit. As stated in Section 3.1.2, those attributes which are required for the tagger to manually markup are explicit (i.e., manually must be added) whereas those that can be computed by the system are implicit (i.e., inferred from explicit attributes). A useful example of implicit attributes may be the bounding box and area that inferred from a polygon, both of which are key values required in training NNs.

3.2 The Data-Capturing Process

Now that we have defined *what* to mark up, we now describe the process to define *how* we capture it. To do this, we need to first consider a way to encode both the *description* and *constraints* of the data we wish to capture from within our dataset. The following sections refer to *Argus* as a generic labelling tool, rather than its concrete implementation referred to in Section 3.4.

This can be done in a four-step process: (1) informing Argus about what it is that we want to capture; (2) extracting information from our dataset; (3) transforming the data into an encodable format; and, (4) loading the data into an AI model. Thus, the following sections describes the data capturing process as an informed Extract-Transform-Load (ETL) process.

3.2.1 Informing Argus

Let us consider that the data we capture from Section 3.1 can be placed into a structured hierarchical data format, represented using our metamodel. For convenience, we will refer to this hypothetical format as the ADF. Let's also assume that we can restrict the kind of data we want to capture using a set of constraint rules, which can be encoded into a hypothetical Argus Constraint Language (ACL). This, also, can be represented by our metamodel.

The first step is to feed in the constrains into a data capturer that restricts what kind of data we wish to read from our image dataset. This declarative format informs the capturer of the potential features, annotations and construction rules that is marked up by the annotator. For example, a concrete implementation of an ACL in an XML schema is given in Listing 3.1, where we define the features, annotations, conditions and dependencies of the image-level and *Bib* segment-level features, using ID referencing inspired by HTML.

Listing 3.1: A hypothetical Argus Constraint Language (ACL) file describing the image-level and *BibSheet* segment-level features as represented in an eXtensible Markup Language (XML) schema.

```
<?xml version="1.0" encoding="utf-8"?>
<capture>
  <!-- Define the Image-Level Features -->
  <feature level="image" id="image">
    <annotation id="photo-crowded" type="boolean">
    </annotation>
    <annotation id="runners" type="collection">
      <!-- Condition that photo crowded is false -->
      <condition context="photo-crowded" value="self == false" />
    </annotation>
  </feature>

  <!-- Define the Bib Segment-Level Feature -->
  <feature level="segment" id="bib" owner="runners">
    <annotation id="bib-sheet" type="polygon">
      <!-- Condition that vertices must be 4 -->
      <condition context="bib-sheet" value="self.vertices == 4" />
    </annotation>
    <annotation id="rbn" type="label">
      <!-- Dependent on bib sheet to exist -->
      <dependency on="bib-sheet" />
    </annotation>
  </feature>
</capture>
```

Argus is therefore told what it is that it needs to capture, and—by using this declarative language—a User Interface (UI) can automatically be generated to indicate to the marker the flow by which we must mark up information.

3.2.2 Extracting Features

Raw images are loaded into an ‘Annotate’ mode of Argus, whereby the user interface is generated using the constraints discussed in Section 3.2.1. Additionally, errors are now automatically shown when conditions or restrictions are not met. Each of the specific features can now be extracted using a number of the elements within the generated UI as per those shown in Table 3.2, whereby the annotator follows the required steps (i.e., in dependency order) to carry out annotations. This, therefore, becomes the workflow of extraction which the annotator runs through—a

Table 3.2: Sample UI elements that can be made to extract features.

Annotation Type	UI Element(s)	Usage
Boolean	Checkbox	Check to indicate TRUE value, FALSE otherwise.
Category	Dropdown-Box Radio Button Group	All variant Possible Values are shown as distinct options within a dropdown box's list items or as separate radio buttons.
Label	Text Box	Text box shown to enter in label value.
Colour	Eye-Dropper Colour Wheel Colour Palette	Selection of colour can be made from eye dropping a distinct colour in the image, from a selection of all colours in the colour wheel, or from a set of colours from a set colour palette.
Polygon	Clicking $n_{vertices}$ times	Boundary is selected by clicking the required number of vertices around the selection area.
Rectangle	Drag-And-Drop	A drag-and-drop around the required boundary is made.

concrete example of such a workflow for our case is shown in Figure 3.10, as represented as a UML activity diagram.

Additionally, the UI elements can provide guidance to the tagger in the form of an instruction. For instance, annotations of a boolean type can be asked as a question: "Is this photo crowded?". Polygons can be specifically guided: "Left click four times around the bib sheet region". Similarly, the same can be done with a colour selection using an eye-dropper: "Click the shorts of the runner to select their shorts colour". These labels and UI elements can be automatically generated based on the information supplied in the Argus Constraint Language (ACL) file.

Keyboard shortcuts can be automatically implemented to improve keyboard-driven navigation, reducing the need to use a mouse and automating responses (i.e., enter 'Y' for 'Yes' and 'N' for 'No')—ultimately to speed up the progression by which taggers extract these features.

Lastly, annotators can mark the image as 'complete', indicating they are finished with all possible annotations in the dataset. This then produces an Argus Data Format (ADF) based on the image (Section 3.2.3). Once all images in the dataset are fully 'complete', Argus informs that the extraction process on the entire dataset is now complete.

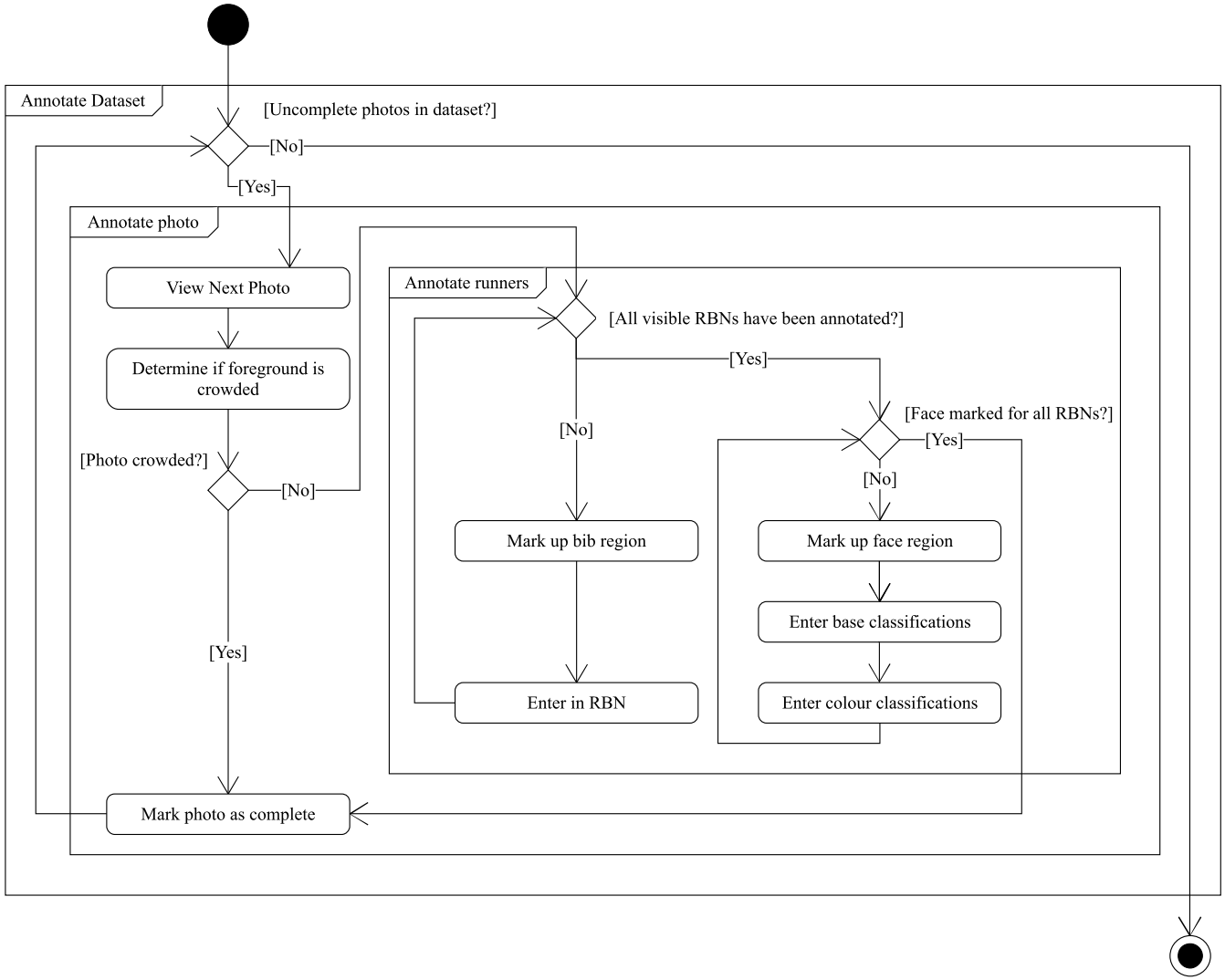


Figure 3.10: Concrete workflow of Argus in our bib annotation context. The ACL structures this process by defining dependency ordering.

3.2.3 Transformation

The features, once extracted, are automatically encoded into an Argus Data Format (ADF) upon marking the image as complete. To prevent data loss, files are encoded as the annotator progresses onto the next feature within the workflow of exatraction. All implicit attributes can be calculated given the explicit attributes manually entered by the runner at this stage.

A sample of an ADF in a language-agnostic tree format is shown in Figure 3.11 for a photo with one runner. This could be serialised into any human-readable hierarchical format, such as a YAML, JSON or XML file. Additionally, workflows from the ACL input file may be generated by a UML activity diagram (as we have done in Figure 3.10) to assist annotators in understanding how to mark up the photo. Thus, the transformation step *universalises* the annotated data into a format readable by any supported parser of the ADF or ACL files. We propose examples of how this data is loaded in the following section.

3.2.4 Load

Once the data is transformed into a consistent format (i.e., an ADF), we are able to supply it to a number of different tools. In our research, we load the ADF into four tools:

1. Argus is executed in 'Review' mode, whereby a second annotator reviews the parsed annotations by loading a directory containing multiple ADFs. There they can run through a number checks to ensure the annotated data is accurate when training the AI models.

2. Custom augmenters load the ADFs as well as the source images to produce a number of variations on the image, essentially turning one image and its respective ADF into many variant images and ADFs. This is imperative when training our NN, as explained in Section 3.3.2.

3. An adapter created for our NN is used to load in an ADF and convert it into a CSV of file paths to images, bounding boxes and labels. The NN reads this file and uses it to learn where bib locations are in an image. This is later discussed in Chapter 4.

4. We used Git[3] to track changes within our ADF. The benefit of serialising the data into a text file is that any version control management software can handle provenance for us (as suggested in Section 3.1), thereby sourcing changes in our AI models easier by sourcing how the training data (i.e., ADF file) has derived over time.

[3] `https://git-scm.com` last accessed 11 Aug 2017.

- **Image** `001`
  - PhotoCrowded : *Boolean*
    - FALSE
  - Runners : *Collection*
    - Runner `1`
      - **Bib**
        - Bounds : *Polygon*
          - $x_1 = 1164, y_1 = 2990$
          - $x_2 = 2147, y_2 = 2990$
          - $x_3 = 2229, y_3 = 3745$
          - $x_4 = 1215, y_4 = 3807$
        - RBN : *Label*
          - `21090`
      - **Face**
        - Bounds : *Rectangle*
          - $x_1 = 512, y_1 = 1034$
          - $x_2 = 1671, y_2 = 2162$
        - Gender : *Category*
          - MALE
        - WearingHat : *Boolean*
          - TRUE
        - WearingGlasses : *Boolean*
          - FALSE
      - **Prominence**
        - LoP : *Category*
          - NO
        - FaceVisible : *Boolean*
          - FALSE
        - Blurry : *Boolean*
          - FALSE
      - **Colours**
        - ShirtColor : *Color*
          - $r = 255, g = 248, b = 239$
        - ShoeColor : *Color*
          - N/A
        - ShortsColor : *Color*
          - $r = 66, g = 63, b = 58$
        - HatColor : *Color*
          - $r = 248, g = 255, b = 134$

Figure 3.11: Sample hierarchical data representation of an ADF, encoding the features (bolded) of the runner on the left. Implicit attributes are removed.

3.3 Data Postprocessing

Once our dataset is annotated, a number of actions can be applied–our images and annotations undergo a series of post-processing and eventually are used by the AI model to make predictions on new datasets. We can broadly categorise these into two sections: flagging the annotations/predictions and augmenting the data. The overall process is illustrated within Figure 3.12.

3.3.1 Flagging

Flagging helps indicate 'check'-states that the annotated or predicted ADF data has undergone. These checks can be considered as another way in which we maintain data provenance. We have considered four states of data: (1) ground truth annotations; (2) *reviewed* ground truth annotations; (3) predicted regions from an AI model; (4) *validated* predictions.

Ground truth annotations encompasses the general process by which data taggers annotate data. This process is largely the initial data capture that occurs within our dataset, described by the process outlined in Section 3.2. As mentioned in Section 3.2.4, once data is annotated, a second parse is done on the annotations by another reviewer to check for any mistakes. Some of our data is also annotated multiple times, and thus multiple ADFs are produced. A reviewer will check these files and use the best matches to merge the best data together—for instance, in a single photo, if three annotators mark a given runner's LoP as YES whereas one annotator marks it as MAYBE, the reviewer may choose to update the MAYBE to the more likely value.

Once ground truth annotations are inspected for a second time, the annotations in the ADF file are *flagged* as 'REVIEWED'. We only allow reviewed data to be augmented (Section 3.3.2) and thus trained in our AI model. This method allows us to maintain a high level of quality and integrity of input data.

The second flag is set on *predicted* ADF data generated by the AI. As the AI model produces ADF files (see Chapter 4), we are able to perform a second parse on the predicted data by loading the predicted ADFs into Argus' validation mode. Here, a human validates that the predictions made by the model are indeed correct, and if not, we are able to update any discrepancies where needed (via Argus modifications) and feed this corrected invalid data into the model again to inform it of any mistakes it is making. Here we can flag the predicted data as 'VALIDATED'. In turn, predications that are validated help ensure that mistakes made by the AI model are amended.

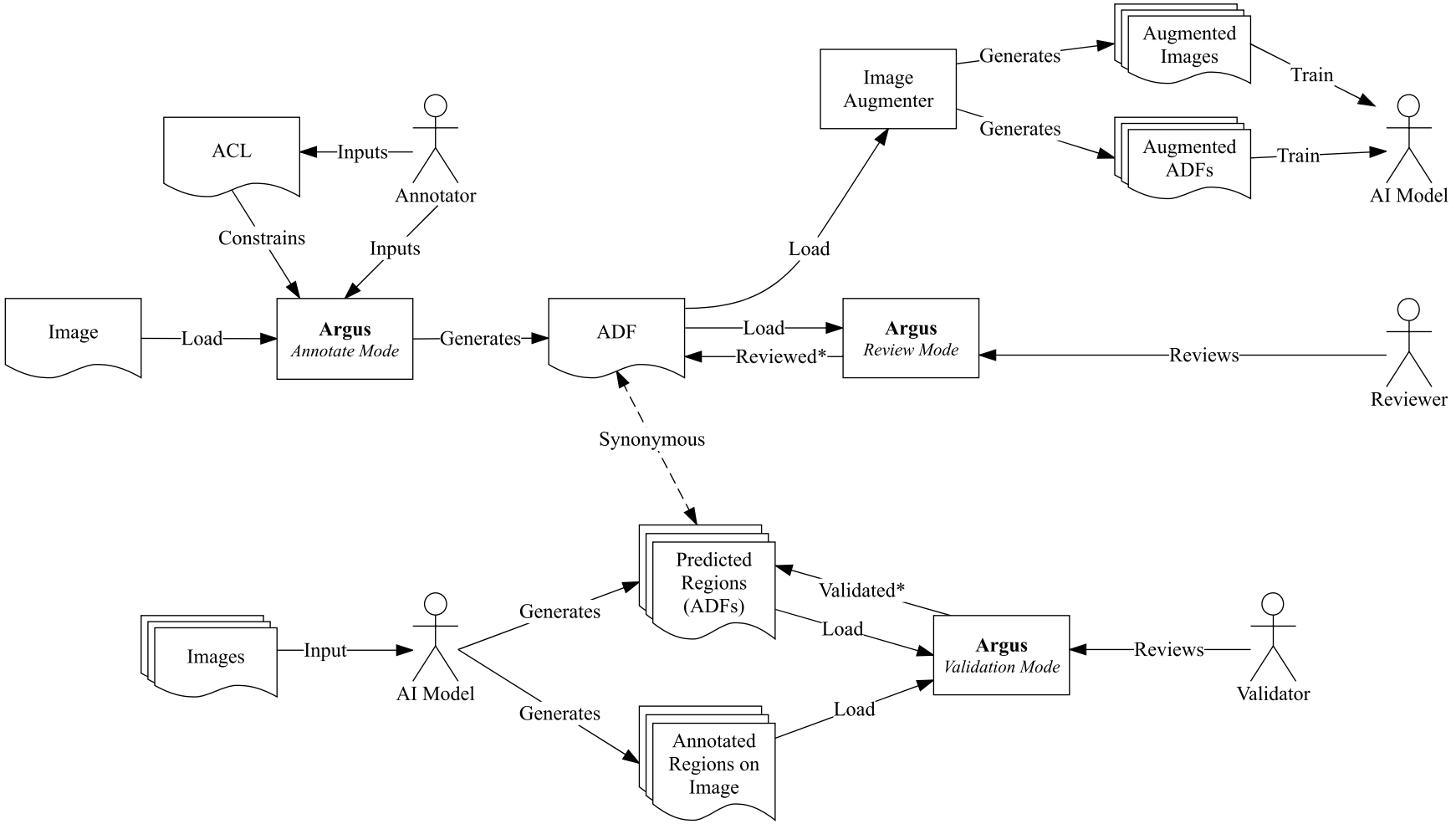


Figure 3.12: A general outline of the processing done on the dataset and all relevant actors and associated files. Asterisks indicate where flags have marked annotations as REVIEWED and predictions as VALIDATED.

Table 3.3: Breakdown of images and annotations (runners) used for training and validation.

Image Type	Training			Validation		
	Images	**Annotations**	**Annotations / Image**	**Images**	**Annotations**	**Annotations / Image**
Original	722	850	1.177	81	109	1.346
Augmented	36100	33098	0.917	4050	4266	1.053
Total	36822	33948	0.922	4131	4375	1.059

3.3.2 Data Augmentation

Previous works has shown that augmenting data makes a classifier more robust in its detection [3, 144, 146]. A solid augmentation strategy is applied to make the training data for our models more extensive and variable by generating deformations and defects and thereby producing synthetic data.

For our network, we manually tagged 803 images over a total of 14 different marathon events using our concrete Argus implementation (Section 3.4) within our dataset. We decided on an augmentation strategy using a 1:50 ratio (thereby to produce a further 40,150 augmented images from these original 803 images). We split the dataset into 90% for training and 10% for validation. The specific quantities of images and annotations chosen for training and validation are described in Table 3.3. An annotation to image ratio less than 1 indicates there were more crowded photos in the sample set.

Our augmentation strategy consisted of the following:

1. perform affine transformations every image,
2. distort colour channels by adding intensities of $\pm 45\%$ applied 50% of the time,
3. distort colour channels by multiplying intensities between a range of $[0.5, 1.5]$ applied 50% of the time,
4. apply one of a gaussian, average or median blur 30% of the time.

Our affine transformations consisted of translations in both the x and y axes shifted between $\pm 35\%$, rotate between $\pm 45^{\circ}$, and shear between $\pm 5\%$. These transformations were implemented using the *imgaug* library for Python[4]. Examples of our augmented images are shown in Figure 3.13.

[4] `https://github.com/aleju/imgaug` last accessed 11 August 2017.

As these transformations occur randomly, there are extreme cases where translation, shearing and rotation occur extremely (e.g., Figures 3.13e and f). This causes some bibs to be cropped off the image. Therefore, we remove these from the transformed ADF file as we cannot have a bounding box referencing outside the image.

(a) Original annotated image.

(b) Slight rotation with blur.

(c) Darkened channels

(d) Slight translation.

(e) Darkened channels with translation.

(f) Large translation with strong rotation.

Figure 3.13: Various instances of our augmented dataset. Here the ADF annotation files are drawn directly onto the source image, with the *BibSheet* polygon shown in green and its respective implicit bounding box attribute in red, dotted.

3.4 Argus

Within this section, we present Argus, a partial implementation of our metamodel. We discuss the overall design of Argus and how this fits into the workflow as presented in Section 3.2. Additionally, we briefly discuss productivity and performance statistics gathered during the data gathering process. Further information related to Argus can be found at `http://deakin.edu.au/~ca/argus`.

3.4.1 Design

Argus is designed as a full-screen application to utilise maximum screen space. The application is designed to be keyboard-driven, so shortcuts are displayed wherever possible. Additionally, the current instruction of the workflow is indicated on the top of the image in red to stand out to image taggers. Figure 3.14 shows the main UI. Further segment-level features are captured in dialogs or user interactiondirectly on the image, as shown in Figures 3.15 and 3.16.

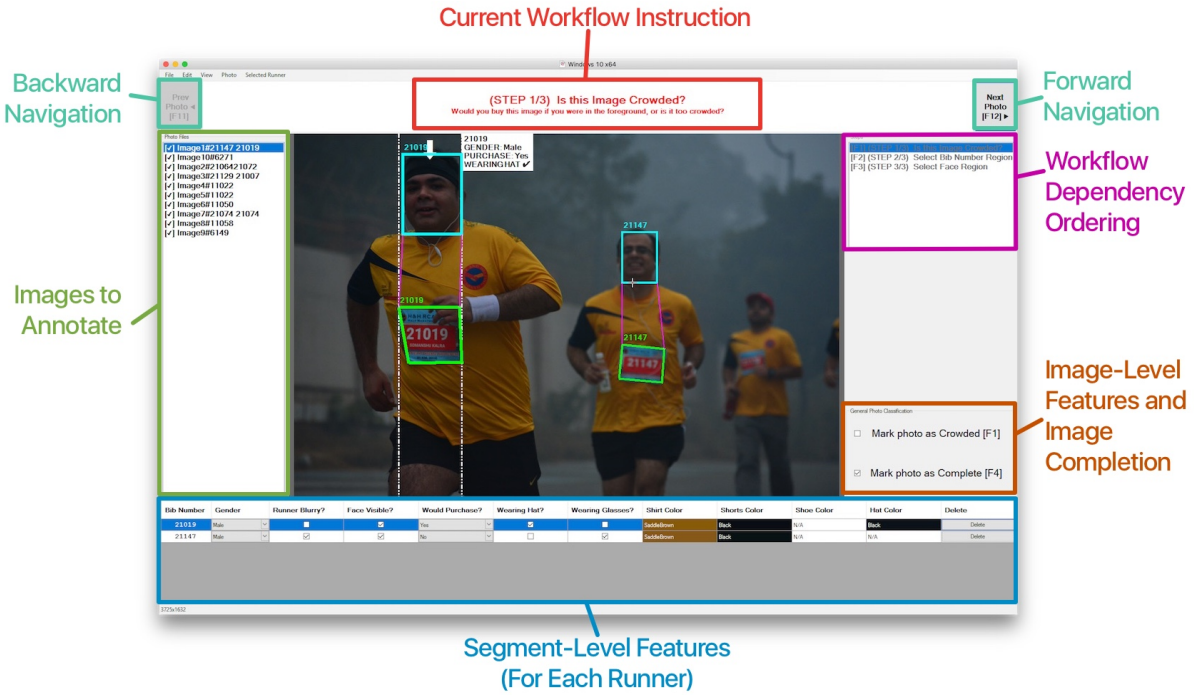


Figure 3.14: An overview of the Argus user interface.

We deployed Argus to data taggers remotely using the ClickOnce Deployment[5] strategy.

[5] `https://msdn.microsoft.com/en-us/library/t71a733d.aspx` last accessed 11 August 2017.

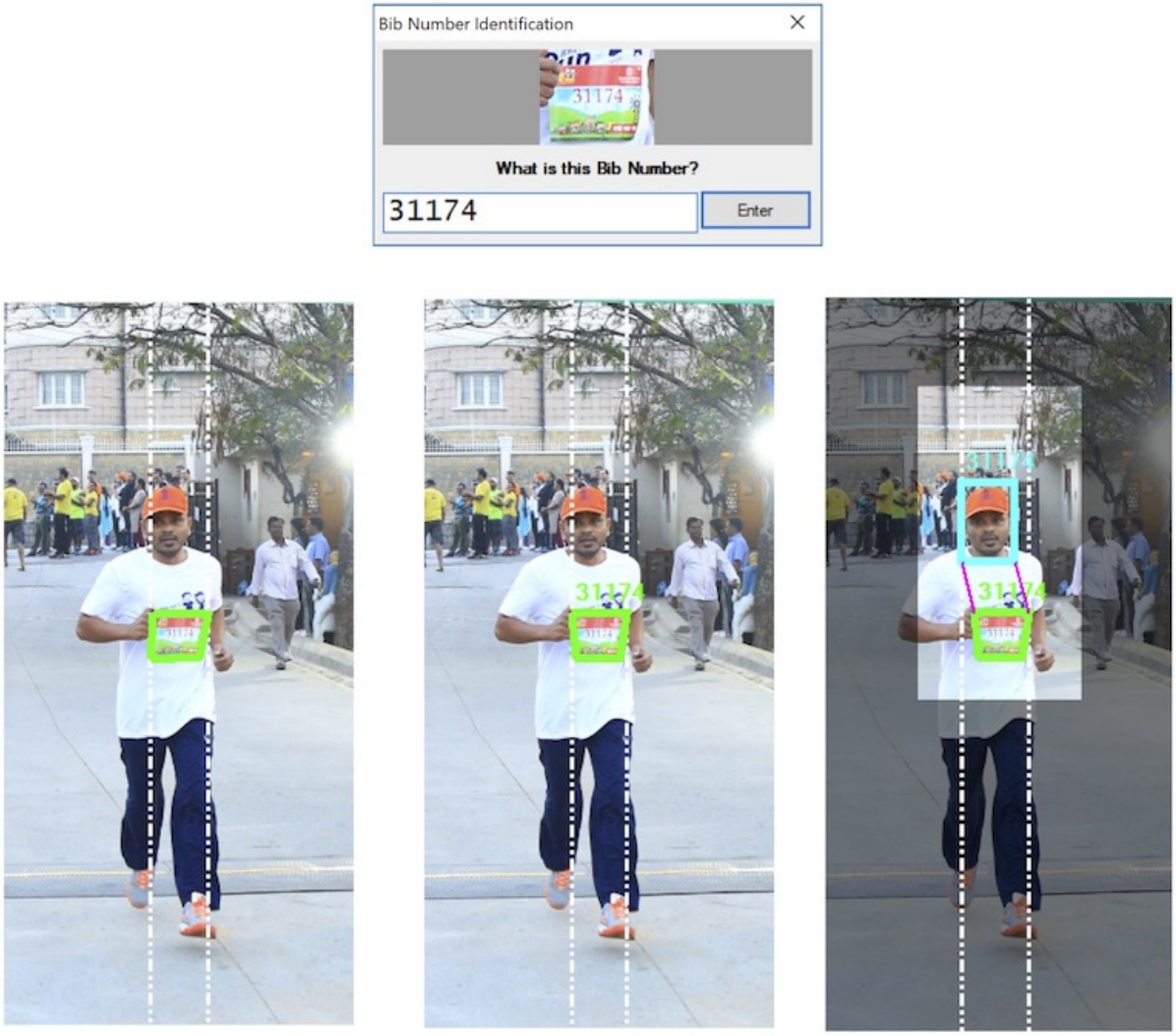


Figure 3.15: *Bib* and *Face* segment-level feature annotation. Users click four times around the bib to mark up the *BibSheet* region (left). A dialog asks the user to enter the *RBN* label (top). The RBN is annotated on the image (middle) and users can progress to drag-and-drop around the *Face* region within the restrictions set (see Section 3.1.3). Note the dependency ordering is present.

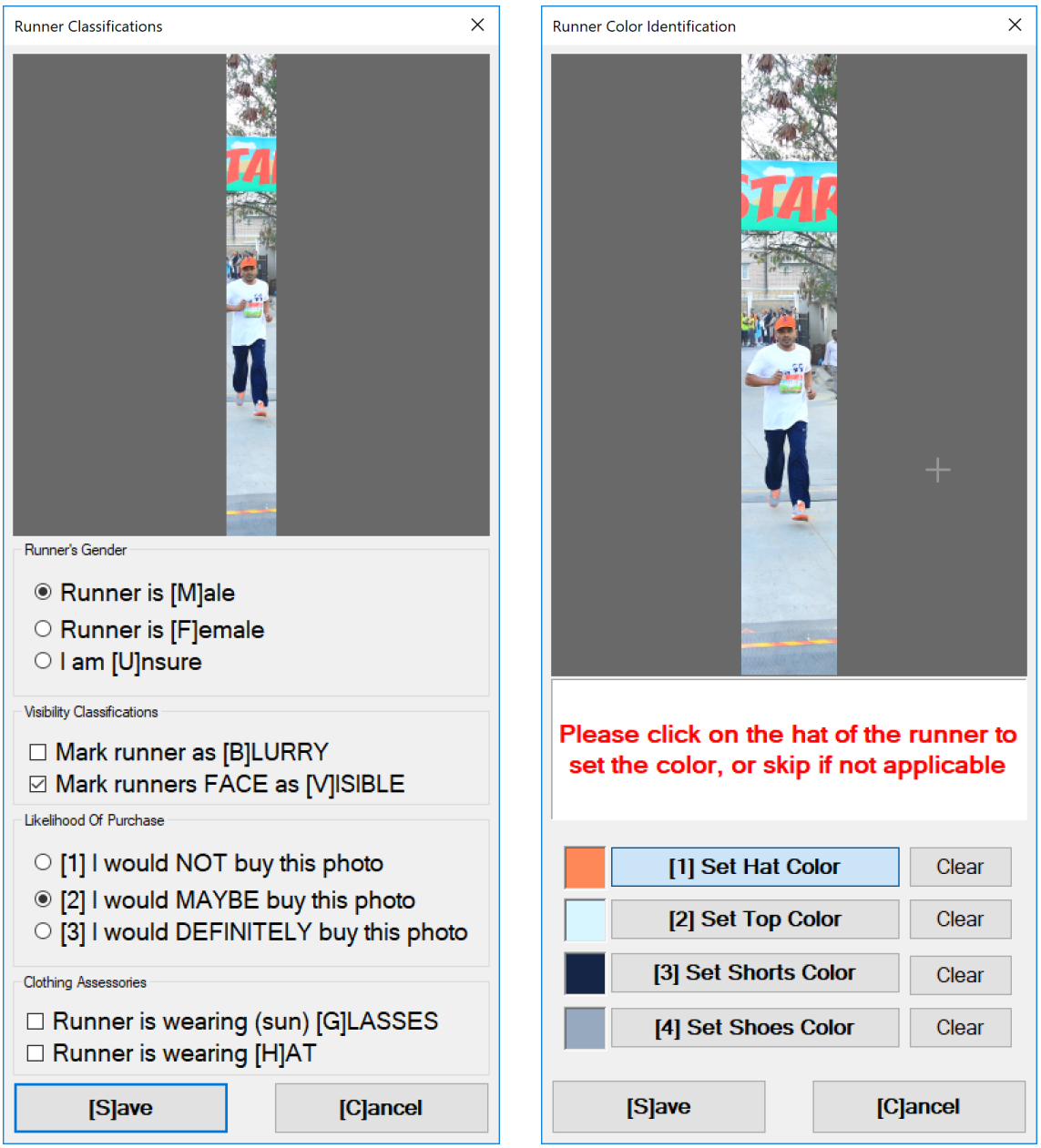
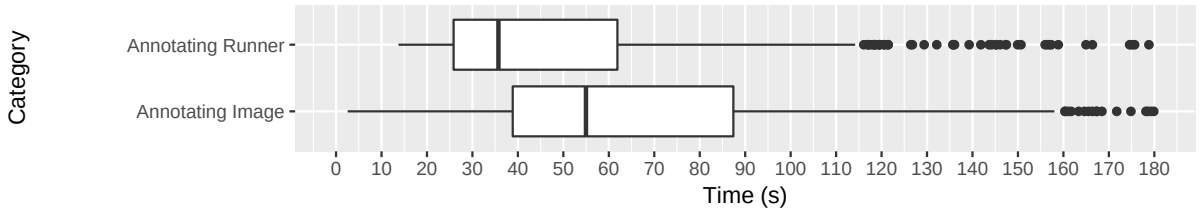


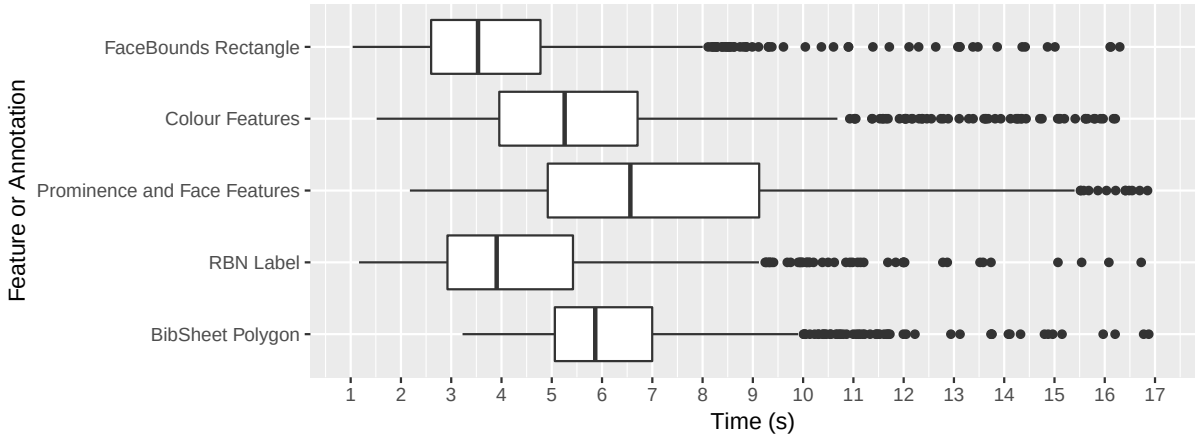
Figure 3.16: Annotation for the *Prominence* and *Colour* segment-level features.

3.4.2 Annotation Throughput

While Argus is running, we gathered a number of metrics to determine what throughput was achievable in our 803 images, presented in Figure 3.17. We measure throughput as either image-level of segment-level features. A single image usually takes less than a minute to markup, and the longest feature to markup is the prominence and face features, especially as we gather the most annotations here. These statistics may be useful for productivity analysis on how long a given dataset may take to markup using Argus, but we leave this area of investigation open for future works.



(a) Overall timing per image.



(b) Segment-level features and annotations marked up.

Figure 3.17: Throughput of 803 images using Argus.

Another metric gathered were the number of mistakes made during annotation. We define 'mistake' as a rule violation, UI violation, or poor data entry. As shown in Table 3.4, it is quite clear how useful adding construction rules to our metamodel is—upon trials made using Argus, we needed to add the *FaceBounds* restriction (Figure 3.3), which was then violated a further 130 times during our annotation. UI restrictions automatically built into Argus also proved useful. These restrictions prevented users from dragging-and-dropping from bottom-right to

top-left (inverse rectangle)[6] or checking if a polygon was being dragged-and-dropped instead of clicked (and vice versa for rectangles).

Table 3.4: Frequencies of mistakes made during the annotation process. These are separated into construction rule violations, UI violations, and poor data entry.

Statistic	**Frequency**
Marked as complete when *Face* region not tagged	20
Selected outside *FaceBounds* restriction	130
Selected *FaceBounds* below *BibSheet*	7
Drag-and-drop inversely for *FaceBounds*	4
Drag-and-drop made for *BibSheet* polygon	73
Clicked instead of drag-and-drop for *FaceBounds*	45
Undos made	70
Deleted runner annotation	46

3.4.3 Annotation LoP Bias

Upon inspection of our LoP metrics, we found that, of all runner's that were annotated (1,031), 77.1% were marked with a LoP of YES, 10.7% were marked as MAYBE and 12.2% were marked as NO. These values are important for prominence ranking training, as we we know there are far more prominent runners (with a higher LoP) to train an AI model. Furthermore, we want to reduce as many MAYBE LoP values as possible as these annotations are discarded in prominence training to reduce impartial bias when training a NN—and as shown our dataset was annotated with the MAYBE values at a minimum. Augmentation (Section 3.3.2) is required to increase the number of NO training samples.

3.4.4 Annotation Quality Evaluation

We took a subset of 260 randomly selected images (approximately one third of the entire dataset) tagged from the data tagging team and assessed the photos for quality assurance. Here, we inspected photos on three tiers: 'Good', 'Okay' and 'Bad' quality. We deemed photos as 'Good' if there were no flaws at all with the tagging, 'Okay' if there were minor errors made that we can compensate with, and 'Bad' if there are significant flaws with the tagging that would affect the AI model. Refer to Figure 3.18 for samples of these images.

[6] Although, it would have been possible to automatically swap the x_1 and y_1 with x_2 and y_2 within Argus.

(a) A 'Bad' annotation **(b)** A 'Bad' annotation **(c)** An 'Okay' annotation

Figure 3.18: 'Okay' and 'Bad' quality tagging tiers.

In Figure 3.18c, we see that the *FaceBounds* annotation is too small around the runner's head, and that the *BibSheet* polygon has been marked as a square, rather than directly around the four corners of the bib sheet. This is considered 'Okay' as we are able to compensate with the extra padding around the *BibSheet*, and the *FaceBounds* could have padding to extend the area. In Figures 3.18a and b, however, we are given completely incorrect information. In the former, the *BibSheet* polygon is not at all reflective of the actual sheet, including part of the runner's shirt in the annotation, and the *FaceRegion* is too small unless significant padding is added. In the latter, the *RBN* has been annotated with the value `100613`, confusing the first character as numeric when it is alphabetic (`I00613`). As stipulated, these instances show significant fallbacks to quality that may hinder training the NN.

In our sample set, we also evaluated whether the number of runner's in an overall photo has a negative impact on the annotation quality (Figure 3.20), as well as how long taggers spend on annotating photos (Figure 3.20). Upon analysis, we can see that most runners are actually well-annotated—only in images containing one runner where there are more 'Bad' annotations than 'Okay' ones, and generally 'Okay' and 'Good' outcomes are largely consistent regardless of runner count. Therefore, likelihood of time spent on annotations is 'Good', and 'Bad' marking distribution is more biased towards short evaluation times where there are few runners in the image. Figure 3.20 confirms our hypothesis where, we can see, the longer the annotator spend time tagging, the better the quality of our data is: a mean of 48 seconds is improved to 52 seconds between 'Bad' and 'Okay', and this jumps to a mean of 55 seconds

for ‘Good’. Therefore a difference in 7 seconds spend on tagging a photo, on average, shows a significant improvement on the quality of the tagging.

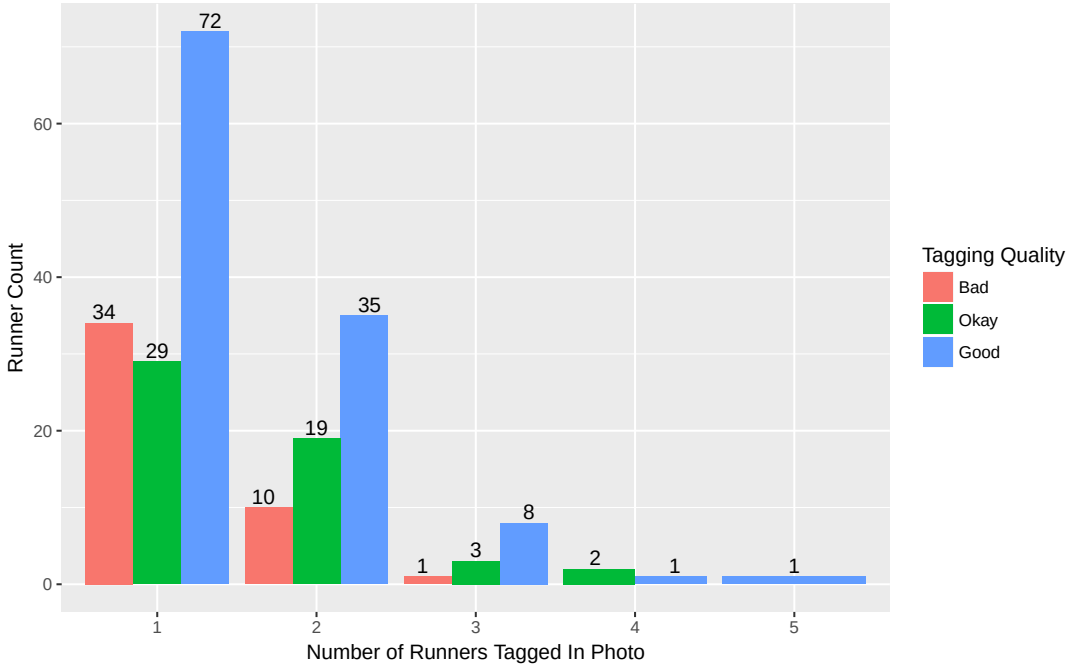


Figure 3.19: The quality of tagging against the number of runners per photo.

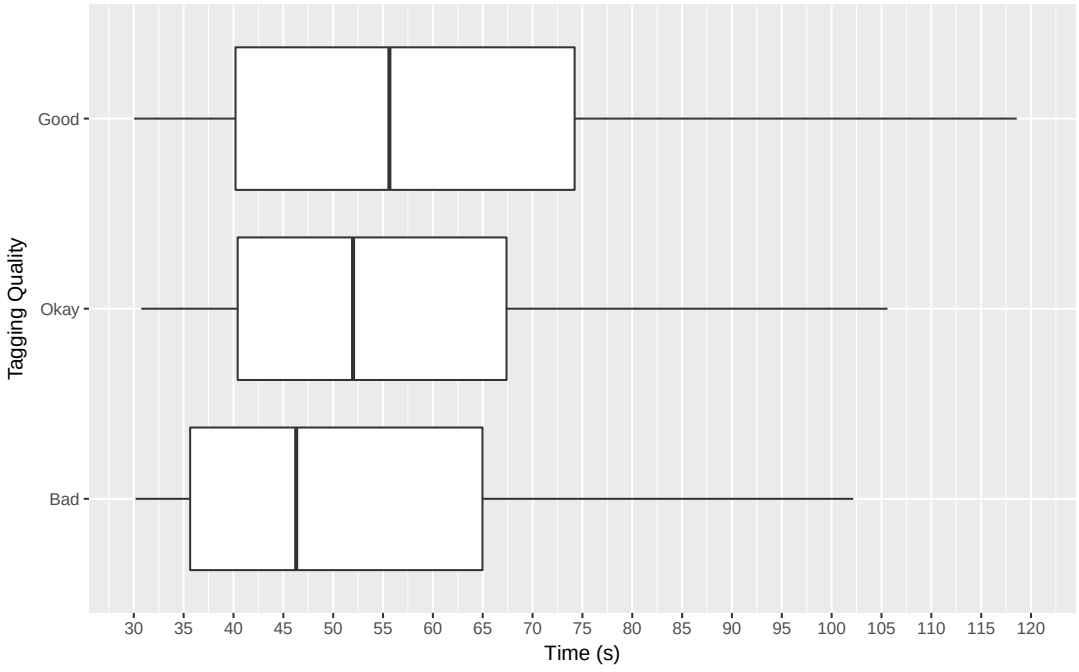


Figure 3.20: Quality of tagging against the time spent evaluating the photo.

3.5 Metamodel Evaluation

In this section, we evaluate our metamodel against other datasets with similar annotations, and heuristically assess it. Additionally, we propose areas where the metamodel could be improved, and highlight comparative systems to Argus.

3.5.1 Dataset Annotations

We analysed seven popular datasets in either text extraction or object recognition and assessed their annotation format to determine the heuristic overlap between the dataset's annotation model and our metamodel. An overview of the datasets, annotation formats and capturing software is shown in Table 3.5. We summarise our analysis of directly mapping components from these formats within the ADF in Table 3.6.

Table 3.5: Various datasets and their respective annotation formats which we have assessed for comparison with our metamodel. See Appendix C for serialisable annotation formats.

Dataset	Ref	Challenge	Format	Listing	Annotation Tooling
MS COCO	[21, 85]	Object Recognition	MS COCO JSON	C.1	Custom Tool via AMT
COCO-Text	[135]	Text Reading	MS COCO JSON	C.2	Tool from Matera et al. [96] via AMT
ICDAR 03–11	[19, 91, 92]	Text Reading	XML	C.4	Custom Tool via Java Applet
ICDAR 11–15	[65, 67, 116]	Text Reading	XML	C.3	CVC APEP [66]
ImageNet	[29]	Object Recognition	PASCAL VOC	C.5	Custom Tool via AMT
SUN	[145]	Object Recognition	PASCAL VOC	C.5	LabelMe [112]
PASCAL-Context	[102]	Object Recognition	PASCAL VOC (2012)	C.6	Custom Tool similar to LabelMe [112]
Synth90K	[47]	Text Reading	MATLAB Binary	N/A	SynthText[7]
SVHN	[104]	Text Reading	MATLAB Binary	N/A	Custom Tool via AMT

For each of these datasets, the annotation capturing strategies differed and, similarly, the annotation encoding was either encoded as a binary MATLAB[8] file or serialised in plain text (either as XML or JSON). In the case where annotations were serialised in XML, a popular format was the PASCAL[9]-Visual Object Classes (VOC) [33] format. This format was made popular with the various PASCAL VOC Challenges [32–35], though custom annotation formats in XML are also present (e.g., ICDAR).

[7] `https://github.com/ankush-me/SynthText` last accessed 15 August 2017.

[8] MATLAB is a registered trademark of The MathWorks, Inc.

[9] Pattern Analysis, Statistical Modelling and Computational Learning (PASCAL)

Table 3.6: Presence of components of popular annotation formats within the ADF metamodel. See Appendix D for detailed mapping.

ADF		COCO JSON	PASCAL VOC XML	ICDAR XML	MATLAB Binaries
Feature	Image-Level	•			
	Segment-Level	•	•	•	•
Annotation	Label	•	•	•	•
	Category	•	•	•	
	Collection		•	•	•
	Boundary	•	•	•	•
Attribute	Implicit	•	•	•	•
	Explicit				•

MS COCO and COCO-Text JSON Format The Microsoft Common Objects in COntext (COCO) [85] dataset and its image captions extension [21] require all annotations to be labelled to one of 91 specified categories. This is done using a three-step annotation pipeline consisting of: (1) labelling all categories of objects in the image, (2) spotting all instances of these characters, and (3) segmenting the instances using a polygon. For our analysis, we focus only on the Object Keypoint and Image Caption Annotation challenges.

We mapped the annotation concepts from both MS COCO (and its text-reading equivalent COCO-Text) and mapped components to the ADF. As shown in Figure D.1, these high-level components are largely complete when expressed in ADF, and the mapping is powerful enough to compute required components of the COCO-based annotation formats to implicit calculations within ADF.

Our metamodel did not consider using Run Length Encoding (RLE) to encode multiple boundaries (polygons), as is used when *Is Crowd* is set to 1 to indicate multiple instances in the image. However, we still capture this under the a form of a *Boundary*. Additionally, we found that the *Is Crowd* is the only mapping of all the formats that relates to an ADF *Image-Level Feature*.

PASCAL VOC XML The PASCAL VOC XML format showed the most consistency with ADF's variant *Annotation* sub-types (Figure B.2), along with the ICDAR XML format. The PASCAL VOC format shows direct mappings to our components, such as *Bounding Box* to an *Annotation*'s equivalent, *Boundary*. Additionally, a collection of *Action*s (as given in the 2012 format to extend the *Pose* of a person feature) can be used as a list of variant *Category* annotations. Further mappings can be seen in Figure D.3.

ICDAR XML Format The two ICDAR datasets (including the 2011 update) can also be captured within an ADF. We present this mapping in Figure D.2. Most notably, we map the *Text Line*, *Word*, *Atom* and *Text Parts* as ADF *Collection*s. The *Don't Care* component is a flag to notify if a line is illegible, which we can indicate using a *Boolean* annotation (a *Category*). The explicit *Color* type in the ICDAR 2011–2015 format can also be mapped to a *Label* (more specifically, the *Colour* label).

MATLAB Binaries To investigate the mapping of variant MATLAB binaries, we inspected the relevant guides on how to read the cell-arrays for both the SynthText 90k[10] and SVHN[11] datasets. Of the formats investigated, these were the simplest, and were the only ones to directly map an *Explicit Attribute* (for Synth90k). These are shown in further detail within Figure D.4.

We have shown that our metamodel is largely consistent with four popular existing data formats, and therefore is largely complete to what previous annotation systems require. We propose that adapters can be used to map such formats, thereby unifying all datasets into one readable format which can be used for AI training and validation. We leave this open for future work.

3.5.2 Comparative Annotation Tools

In this section, we compare Argus to comparative annotation tools and services used to annotate datasets.

LabelMe LabelMe[12] (Figure 3.22), developed by Russell et al. [112], is a web-based application that requires annotators to set up on their own system. It has been successfully used to markup the large Scene UNderstanding (SUN) dataset by novice users, though the developers did note a number of reflective fallbacks and potential difficulties using the tool [4]. LabelMe does not contain a specific instructional workflow, unlike Argus, and therefore annotators are allowed to selectively choose whichever objects to annotate[13]. This makes the resulting dataset largely incomplete as (potentially) not all objects in each image are fully annotated. Additionally, there is no restriction to what types of objects are annotated in the image, as any text can be used to describe an object (rather from a hierarchical category, such as in [85]).

[10] `http://www.robots.ox.ac.uk/~vgg/data/scenetext/readme.txt` last accessed 17 August 2017.
[11] `http://ufldl.stanford.edu/housenumbers/#downloads` last accessed 17 August 2017.
[12] `http://labelme.csail.mit.edu/` last accessed 17 August 2017.
[13] Instructions are potentially vague: "Use your mouse to click around the boundary of *some* objects".

Figure 3.21: The LabelMe user interface [112].

APEP Karatzas et al. [66] introduced the CVC Annotation and Performance Evaluation Platform (APEP)[14], made popular for use using the ICDAR 2011–2015 Robust Reading Competitions. Unlike our system, users can edit the per-pixel boundaries using a 'flood-fill' or 'magic-select' markup tool (with an adjustable tolerance), followed by a skeletonisation of the boundaries filled. This speeds up selection of specific text areas quite useful (see Figure 3.22). While we did not incorporate a feature into Argus, a potential future version would benefit using such an annotation tool for a polygon, rather than clicking $n_{vertices}$ times if the vertices count is unknown. This could also be encoded using RLE as suggested from the COCO format in Section 3.5.1. Additionally, users are able to zoom into the photo to get finer accuracies at the pixel level.

AMT Amazon Mechanical Turk (AMT)[15] provides Software as a Service (SaaS) that does not require downloading or setting up (typically laborious for annotators). Annotation is achieved in the form of customised Human Intelligent Tasks (HITs), using a custom interface. Recently, this is typically the most popular choice of annotation outsourcing [21, 29, 85, 104, 135] due

[14] `http://labelme.csail.mit.edu/` last accessed 17 August 2017.
[15] `https://www.mturk.com/mturk/welcome` last accessed 11 August 2017.

Figure 3.22: The APEP ground truthing tool, showing a hierachy of textual content and defined text parts (flood-filled areas and skeletons) [66].

to flexibility in developing customised interfaces for the task at hand, though the benefits of ensuring a user-friendly crowdsourcing annotation system are presented by Matera et al. [96]. Sorokin and Forsyth [125] discuss the utility of using AMT for annotation.

VATIC VATIC (the Video Annotation Tool from Irvine, California) is a a free web-based product developed by Vondrick, Patterson, and Ramanan [137], running on AMT. We note this tool specifically for the use of annotating video stills within an image: its intended purpose is for the sole use of object tracking in videos. There are only three annotations per object: the object name, the object is out of view, and the object is obstructed. It is useful in that not all frames have to be individually annotated, as the tool uses object tracking to estimate the movement between every 2-3 seconds. This may be useful to incorporate into future works of Argus.

ScaleAPI ScaleAPI[16] is a recent SaaS that provides a web-based Application Programming Interface (API) to return human-marked annotation in realtime. Various boundary annotations can be made on an image. Listings 3.2 and 3.3 show the request and response for a simple instruction to draw bounding boxes around all pedestrians and cars in the image[17]. Additionally, object segmentation (on a per-pixel basis). While there is no need for workflows in a single request (as multiple requests can be made for multiple workflow steps), instructions to annotators are provided on a per-request basis. Furthermore, ScaleAPI allows for categorisation and

[16] `http://www.scaleapi.com` last accessed 11 August 2017.

[17] `https://docs.scaleapi.com/?shell#bounding-box-annotation` last accessed 17 August 2017.

comparison tasks, data collection, OCR and audio transcription.

Listing 3.2: A sample ScaleAPI HTTP request made using cURL[18].

```
curl "https://api.scaleapi.com/v1/task/annotation" \
  -u "SCALE_API_KEY:" \
  -d callback_url="http://www.example.com/callback" \
  -d instruction="Draw a box around each **car** and **pedestrian**." \
  -d attachment_type=image \
  -d attachment="http://i.imgur.com/XOJbalC.jpg" \
  -d objects_to_annotate="car" \
  -d objects_to_annotate="pedestrian" \
  -d with_labels=true \
  -d min_width="30" \
  -d min_height="30"
```

Listing 3.3: Sample JSON response from the request made in Listing 3.2.

```
{
  "task_id": "5774cc78b01249ab09f089dd",
  "created_at": "2016-9-03T07:38:32.368Z",
  "callback_url": "http://www.example.com/callback",
  "type": "annotation",
  "status": "pending",
  "instruction": "Draw a box around each **car** and **pedestrian**",
  "urgency": "day",
  "params": {
    "with_labels": true,
    "min_width": 30,
    "min_height": 30,
    "objects_to_annotate": [
      "car",
      "pedestrian"
    ],
    "attachment_type": "image",
    "attachment": "http://i.imgur.com/XOJbalC.jpg"
  },
  "metadata": {}
}
```

[18] `https://curl.haxx.se/` last accessed 17 August 2017.

Figure 3.23: The VATIC web-based user interface. Sourced from `https://github.com/cvondrick/vatic`. (Last viewed 11 August, 2017.)

3.6 Conclusions

This chapter describes an inductive approach used for developing a generalisable metamodel from empirical iterations of systems development to gap the missing epistemology in the area of data capturing architectures. By developing tagging system, Argus, and theorising and reflecting upon the implementation, we were able to refine and discover a concrete metamodel. This metamodel alleviates the difficulties in maintaining data provenance when training AI models by making the serialisable, the therefore trackable with any version control system.

How may this metamodel be used elsewhere? It seems easily adaptable to similar areas in the field of computer vision processing, such as LPR or TSR. However, could we move this to beyond static images? Future works may extend our metamodel to multiple image frames, and therefore annotating videos frame-by-frame is within reach. Some current limitations of the metamodel (in its current form) is the difficulty to apply it to non-vision-related topics, such as Natural Language Processing (NLP), audio processing or sensor data processing. Some key concepts may be missing from our metamodel in these areas, and we speculate this for work that researchers with domain knowledge in relevant areas could discover. This said, the fundamental features of a *feature* that an AI is to learn from this data still remains, so perhaps it is plausible.

Is the methodological approach we used extendable? Is it wise to develop concrete systems, and then develop back toward a metamodel? These are questions that we leave for future research in metamodel development. The area of scalable and crowdsourced dataset annotation is becoming evermore increasing, with services such as AMT and ScaleAPI rising in popularity.

The primary contributions of this chapter are:

- a metamodel that describes a schema to annotate large image datasets, and
- an exploratory methodological approach in designing a metamodel from a concrete system.

Minor contributions in this chapter include:

- an overview of metrics gathered to determine the throughput of annotating a large dataset of images,
- an overall pipeline in how to handle and deal with data provenance in AI models, and
- a basic methodology and metrics to assess the quality of annotations.

Chapter 4

Processing Pipeline

Chapter 5

Evaluation Strategies

Develop a number of different evaluation strategies for the bib and number detection pipeline. First pass (ideal case) of this calculation would be 100 images that contain 1 bib only. Accuracy would be number of detected bibs / total bibs. Second pass (realistic case) would be to select random number of bibs per photo in a sample size of 100 photos, aggregate the number of bibs we have, then run our pipeline to see how many bibs were detected. Evaluation strategy will also consist of the following factors:

Run detection with and without segment the runner figure (i.e., with and without person detection filter).

Run with 1 training image augmented n times; 100 training images (10 from 10 races) augmented n times; 500 training images (50 from 10 races) augmented n times; all training images from all races augmented n times.

This would therefore produce 2 * 4 = 8 models to validate with and calculate accuracies (thereby seeing how accuracies change given the different models).

Passing cropped bib regions into OpenCV's Neural Network image detection or Tesseract to see how well the recognition works.

5.1 Open Source Tools

5.2 Existing Pipelines From Literature

5.3 Hermes Approach

Chapter 6

Findings

Chapter 7

Discussion

Chapter 8

Conclusions and Future Work

References

[1] Anagnostopoulos, C.-N., I. Anagnostopoulos, V. Loumos, and E. Kayafas (2006). A License Plate-Recognition Algorithm for Intelligent Transportation System Applications. *IEEE Trans. Intelligent Transportation Systems*.

[2] Anagnostopoulos, C.-N., I. Anagnostopoulos, I. D. Psoroulas, V. Loumos, and E. Kayafas (2008). License Plate Recognition From Still Images and Video Sequences - A Survey. *IEEE Trans. Intelligent Transportation Systems*.

[3] Baird, H. S. (1992). Document Image Defect Models. In *Erratum*, pp. 546–556. Berlin, Heidelberg: Springer Berlin Heidelberg.

[4] Barriuso, A. and A. Torralba (2012). Notes on image annotation. *CoRR abs/1210.3448*.

[5] Bay, H., A. Ess, T. Tuytelaars, and L. J. Van Gool (2008). Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*.

[6] Belongie, S. J., J. Malik, and J. Puzicha (2001). Matching Shapes. *ICCV*.

[7] Ben-ami, I., T. Basha, and S. Avidan (2012). Racing Bib Numbers Recognition. In *British Machine Vision Conference 2012*, pp. 19.1–19.10. British Machine Vision Association.

[8] Bengio, Y., P. Lamblin, D. Popovici, and H. Larochelle (2006). Greedy Layer-Wise Training of Deep Networks. *NIPS*.

[9] Bézivin, J. (2006). Model Driven Engineering: An Emerging Technical Space. In *Computer Vision – ACCV 2010*, pp. 36–64. Berlin, Heidelberg: Springer Berlin Heidelberg.

[10] Bissacco, A., M. Cummins, Y. Netzer, and H. Neven (2013). PhotoOCR - Reading Text in Uncontrolled Conditions. *ICCV*.

[11] Buneman, P., S. Khanna, and W.-C. Tan (2000, November). Data Provenance: Some Basic Issues. In *Computer Vision – ACCV 2010*, pp. 87–93. Berlin, Heidelberg: Springer Berlin Heidelberg.

[12] Burges, C. J. C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Min. Knowl. Discov. 2*(2), 121–167.

[13] Canny, J. F. (1986). A Computational Approach to Edge Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*.

[14] Cano-Perez, J. and J. C. Pérez-Cortes (2003). Vehicle License Plate Segmentation in Natural Images. *IbPRIA 2652*(Chapter 17), 142–149.

[15] Chen, D. and J. Luettin (2000). A survey of text detection and recognition in images and videos.

[16] Chen, D., J.-M. Odobez, and H. Bourlard (2004a). Text detection, recognition in images and video frames. *Pattern Recognition*.

[17] Chen, D., J.-M. Odobez, and H. Bourlard (2004b). Text detection, recognition in images and video frames. *Pattern Recognition*.

[18] Chen, D., J.-M. Odobez, and J.-P. Thiran (2004). A localization/verification scheme for finding text in images and video frames based on contrast independent features and machine learning methods. *Sig. Proc. - Image Comm.*.

[19] Chen, H., S. S. Tsai, G. Schroth, D. M. Chen, R. Grzeszczuk, and B. Girod (2011). Robust text detection in natural images with edge-enhanced Maximally Stable Extremal Regions. *ICIP*.

[20] Chen, T., M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang (2015, December). MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *arXiv.org*.

[21] Chen, X., H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollár, and C. L. Zitnick (2015, April). Microsoft COCO Captions: Data Collection and Evaluation Server.

[22] Chen, X. and A. L. Yuille (2004a). Detecting and reading text in natural scenes. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004*, pp. 366–373. IEEE.

[23] Chen, X. and A. L. Yuille (2004b). Detecting and Reading Text in Natural Scenes. *CVPR*.

[24] Chen, X. and A. L. Yuille (2005). A Time-Efficient Cascade for Real-Time Object Detection - With applications for the visually impaired. *CVPR Workshops*.

[25] Cleverdon, C., J. Mills, and M. Keen (1966). Factors Determining the Performance of Indexing Systems. Technical report, ASLIB Cranfield Research Project, Cranfield.

[26] Cortes, C. and V. Vapnik (1995). Support-Vector Networks. *Machine Learning*.

[27] Cui, Y. and J. Widom (2003, May). Lineage tracing for general data warehouse transformations. *The VLDB Journal The International Journal on Very Large Data Bases 12*(1), 41–58.

[28] Dalal, N. and B. Triggs (2005). Histograms of Oriented Gradients for Human Detection. *CVPR 1*, 886–893.

[29] Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255. IEEE.

[30] Eichner, M. L. and T. P. Breckon (2008). Integrated speed limit detection and recognition from real-time video. In *2008 IEEE Intelligent Vehicles Symposium (IV)*, pp. 626–631. IEEE.

[31] Epshtein, B., E. Ofek, and Y. Wexler (2010). Detecting text in natural scenes with stroke width transform. *CVPR*.

[32] Everingham, M., S. M. A. Eslami, L. J. Van Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman (2015). The Pascal Visual Object Classes Challenge - A Retrospective. *International Journal of Computer Vision*.

[33] Everingham, M., L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman (2009, September). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision 88*(2), 303–338.

[34] Everingham, M., L. J. Van Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman (2010). The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*.

[35] Everingham, M., A. Zisserman, C. K. I. Williams, L. J. Van Gool, M. Allan, C. M. Bishop, O. Chapelle, N. Dalal, T. Deselaers, G. Dorkó, S. Duffner, J. Eichhorn, J. D. R. Farquhar, M. Fritz, C. Garcia, T. L. Griffiths, F. Jurie, D. Keysers, M. Koskela, J. Laaksonen, D. Larlus, B. Leibe, H. Meng, H. Ney, B. Schiele, C. Schmid, E. Seemann, J. Shawe-Taylor, A. J. Storkey, S. Szedmák, B. Triggs, I. Ulusoy, V. Viitaniemi, and J. Zhang (2005). The 2005 PASCAL Visual Object Classes Challenge. *MLCW*.

[36] Faloutsos, C., R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz (1994, July). Efficient and effective Querying by Image Content. *Journal of Intelligent Information Systems 3*(3-4), 231–262.

[37] Freeman, W. T. and M. Roth (1995). Orientation histograms for hand gesture recognition. *International workshop on automatic ...*.

[38] Freund, Y. and R. E. Schapire (1996). Experiments with a New Boosting Algorithm. *ICML*.

[39] Friedman, J., R. Tibshirani, and T. Hastie (2000, April). Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). *The annals of statistics 28*(2), 337–407.

[40] Fu, C., C.-W. Cheng, W.-H. Shen, Y.-L. Wei, and H.-M. Tsai (2015). LightBib: Marathoner Recognition System with Visible Light Communications. In *2015 IEEE International Conference on Data Science and Data Intensive Systems (DSDIS)*, pp. 572–578. IEEE.

[41] Gatos, B., I. Pratikakis, and K. Kepene (2005). Text detection in indoor/outdoor scene images. *Proc First Workshop of ...*.

[42] Girod, B., V. Chandrasekhar, D. Chen, N.-M. Cheung, R. Grzeszczuk, Y. Reznik, G. Takacs, S. Tsai, and R. Vedantham (2011). Mobile Visual Search. *IEEE Signal Processing Magazine 28*(4), 61–76.

[43] Girshick, R., J. Donahue, T. Darrell, and J. Malik (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 580–587. IEEE.

[44] Girshick, R. B. (2015). Fast R-CNN. *ICCV*.

[45] Gllavata, J., R. Ewerth, and B. Freisleben (2004). Text Detection in Images Based on Unsupervised Classification of High-Frequency Wavelet Coefficients. *ICPR*.

[46] Gonzalez, Á., L. M. Bergasa, J. J. Y. Torres, and S. Bronte (2012). Text location in complex images. *ICPR*.

[47] Gupta, A., A. Vedaldi, and A. Zisserman (2016, April). Synthetic Data for Text Localisation in Natural Images. *arXiv.org*.

[48] Hanif, S. M. and L. Prevost (2009). Text Detection and Localization in Complex Scene Images using Constrained AdaBoost Algorithm. *ICDAR*.

[49] Hanif, S. M., L. Prevost, and P. Negri (2008). A cascade detector for text detection in natural scene images. *ICPR*.

[50] He, K., G. Gkioxari, P. Dollár, and R. B. Girshick (2017). Mask R-CNN. *CoRR*.

[51] Heisele, B., T. Serre, S. Mukherjee, and T. A. Poggio (2001). Feature Reduction and Hierarchy of Classifiers for Fast Object Detection in Video Images. *CVPR*, 8.

[52] Horn, B. (1986, January). *Robot Vision*. MIT Press.

[53] Hua, X.-S., L. Wenyin, and H. Zhang (2001). Automatic Performance Evaluation for Video Text Detection. *ICDAR*.

[54] Hua, X.-S., L. Wenyin, and H. Zhang (2004). An automatic performance evaluation protocol for video text detection algorithms. *IEEE Trans. Circuits Syst. Video Techn.*.

[55] Huang, X., T. Shen, R. Wang, and C. Gao (2015). Text detection and recognition in natural scene images. In *2015 International Conference on Estimation, Detection and Information Fusion (ICEDIF)*, pp. 44–49. IEEE.

[56] Ikeda, R. and J. Widom (2009). Data lineage: A survey.

[57] Ivanov, I., J. Bzivin, and M. Aksit (2002, 10). Technological spaces: An initial appraisal. pp. 1–6. <http://www.cs.rmit.edu.au/fedconf/2002/program.html>.

[58] Jaderberg, M., K. Simonyan, A. Vedaldi, and A. Zisserman (2016). Reading Text in the Wild with Convolutional Neural Networks. *International Journal of Computer Vision*.

[59] Jain, A. K. and B. Y. 0002 (1998). Automatic text location in images and video frames. *ICPR*.

[60] Jin, C. M., Z. Omar, and M. H. Jaward (2016). A mobile application of American sign language translation via image processing algorithms. In *2016 IEEE Region 10 Symposium (TENSYMP)*, pp. 104–109. IEEE.

[61] Jin, J., K. Fu, and C. Zhang (2014, September). Traffic Sign Recognition With Hinge Loss Trained Convolutional Neural Networks. *IEEE Transactions on Intelligent Transportation Systems 15*(5), 1991–2000.

[62] Jung, C., Q. Liu, and J. Kim (2009, January). A stroke filter and its application to text localization. *Pattern Recognition Letters 30*(2), 114–122.

[63] Jung, K., K. In Kim, and A. K Jain (2004, May). Text information extraction in images and video: a survey. *Pattern Recognition 37*(5), 977–997.

[64] Jung, K., K. I. Kim, and A. K. Jain (2004). Text information extraction in images and video - a survey. *Pattern Recognition*.

[65] Karatzas, D., L. Gomez-Bigorda, A. Nicolaou, S. K. Ghosh, A. D. Bagdanov, M. Iwamura, J. Matas, L. Neumann, V. R. Chandrasekhar, S. Lu, F. Shafait, S. Uchida, and E. Valveny (2015). ICDAR 2015 competition on Robust Reading. *ICDAR*.

[66] Karatzas, D., S. Robles, and L. Gomez (2014). An On-line Platform for Ground Truthing and Performance Evaluation of Text Extraction Systems. In *2014 11th IAPR International Workshop on Document Analysis Systems*, pp. 242–246. IEEE.

[67] Karatzas, D., F. Shafait, S. Uchida, M. Iwamura, L. G. i. Bigorda, S. R. Mestre, J. Mas, D. F. Mota, J. A. Almazan, and L. P. de las Heras (2013). ICDAR 2013 Robust Reading Competition. In *2013 12th International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1484–1493. IEEE.

[68] Kim, H.-K. (1996). Efficient Automatic Text Location Method and Content-Based Indexing and Structuring of Video Database. *J. Visual Communication and Image Representation*.

[69] Kim, K. I., K. Jung, and J. H. Kim (2003). Texture-Based Approach for Text Detection in Images Using Support Vector Machines and Continuously Adaptive Mean Shift Algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*.

[70] Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). ImageNet Classification with Deep Convolutional Neural Networks. *NIPS*.

[71] Kumar, D., M. N. A. Prasad, and A. G. Ramakrishnan (2012, December). *MAPS: midline analysis and propagation of segmentation*. midline analysis and propagation of segmentation. New York, New York, USA: ACM.

[72] Kundu, S. K. and P. Mackens (2015). Speed Limit Sign Recognition Using MSER and Artificial Neural Networks. *ITSC*.

[73] LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2324.

[74] Lee, C. W., K. Jung, and H. J. Kim (2003, November). Automatic text detection and removal in video sequences. *Pattern Recognition Letters 24*(15), 2607–2623.

[75] Lee, C.-Y. and S. Osindero (2016, March). Recursive Recurrent Nets with Attention Modeling for OCR in the Wild. pp. 1–10.

[76] Lee, E. R., P. K. Kim, and H. J. Kim (1994). Automatic Recognition of a Car License Plate using Color Image Processing. *ICIP 2*, 301–305.

[77] Lee, S., M. S. Cho, K. Jung, and J. H. Kim (2010). Scene Text Extraction with Edge Constraint and Text Collinearity. *ICPR*.

[78] Li, H., D. S. Doermann, and O. E. Kia (2000). Automatic text detection and tracking in digital video. *IEEE Trans. Image Processing*.

[79] Li, Y. and H. Lu (2012). Scene text detection via stroke width. *ICPR*.

[80] Li, Y., H. Qi, J. Dai, X. Ji, and Y. Wei (2016). Fully convolutional instance-aware semantic segmentation. *arXiv.org*.

[81] Lian, Z., X. Jing, S. Sun, and H. Huang (2016). Frequency Selective Convolutional Neural Networks for Traffic Sign Recognition. In *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring*, pp. 1–5. IEEE.

[82] Liang, J., D. S. Doermann, and H. Li (2005). Camera-based analysis of text and documents - a survey. *IJDAR*.

[83] Lienhart, R. and J. Maydt (2002). An extended set of Haar-like features for rapid object detection. *ICIP*.

[84] Lienhart, R. and A. Wernicke (2002). Localizing and segmenting text in images and videos. *IEEE Trans. Circuits Syst. Video Techn.*.

[85] Lin, T.-Y., M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár (2014, May). Microsoft COCO: Common Objects in Context. *arXiv.org*.

[86] Liu, C. L., M. Koga, and H. Fujisawa (2005). Gabor feature extraction for character recognition: comparison with gradient feature. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pp. 121–125 Vol. 1. IEEE.

[87] Liu, H. and X. Ding (2005). Handwritten Character Recognition Using Gradient Feature and Quadratic Classifier with Multiple Discrimination Schemes. *ICDAR*.

[88] Liu, Y., S. Goto, and T. Ikenaga (2006). A Contour-Based Robust Algorithm for Text Detection in Color Images. *IEICE Transactions*.

[89] Liu, Z. and S. Sarkar (2008). Robust outdoor text detection using text intensity and shape features. *ICPR*.

[90] Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision 60*(2), 91–110.

[91] Lucas, S. M. (2005). ICDAR 2005 text locating competition results. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pp. 80–84 Vol. 1. IEEE.

[92] Lucas, S. M., A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young (2003). ICDAR 2003 robust reading competitions. In *Seventh International Conference on Document Analysis and Recognition*, pp. 682–687. IEEE Comput. Soc.

[93] Lucas, S. M., A. Panaretos, L. Sosa, A. Tang, S. Wong, R. Young, K. Ashida, H. Nagai, M. Okamoto, H. Yamamoto, H. Miyao, J. Zhu, W. Ou, C. Wolf, J.-M. Jolion, L. Todoran, M. Worring, and X. Lin (2005, July). ICDAR 2003 robust reading competitions: entries, results, and future directions. *IJDAR 7*(2-3), 105–122.

[94] Mairal, J., F. R. Bach, J. Ponce, G. Sapiro, and A. Zisserman (2008). Discriminative learned dictionaries for local image analysis. *CVPR*, 10.

[95] Matas, J., O. Chum, M. Urban, and T. Pajdla (2002). Robust Wide Baseline Stereo from Maximally Stable Extremal Regions. *BMVC*.

[96] Matera, T., J. Jakes, and M. Cheng (2014). A user friendly crowdsourcing task manager. *. . . on Computer Vision . . . .*

[97] McConnell, R. (1986, January 28). Method of and apparatus for pattern recognition. US Patent 4,567,610.

[98] Mikolajczyk, K., T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. J. Van Gool (2005). A Comparison of Affine Region Detectors. *International Journal of Computer Vision*.

[99] Minetto, R., N. Thome, M. Cord, J. Fabrizio, and B. Marcotegui (2010). Snoopertext - A multiresolution system for text detection in complex visual scenes. *ICIP*.

[100] Mohan, A., C. Papageorgiou, and T. A. Poggio (2001). Example-Based Object Detection in Images by Components. *IEEE Trans. Pattern Anal. Mach. Intell.*.

[101] Moody, D. L. (2009). The "Physics" of Notations - Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Trans. Software Eng.*.

[102] Mottaghi, R., X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille (2014). The Role of Context for Object Detection and Semantic Segmentation in the Wild. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004*, pp. 891–898. IEEE.

[103] Mutch, J. and D. G. Lowe (2006). Multiclass Object Recognition with Sparse, Localized Features. *CVPR*.

[104] Netzer, Y., T. Wang, and A. Coates (2011). Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.

[105] Nistér, D. and H. Stewénius (2008). Linear Time Maximally Stable Extremal Regions. *ECCV*.

[106] Ojala, T., M. Pietikainen, and D. Harwood (1994). Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In *12th International Conference on Pattern Recognition*, pp. 582–585. IEEE Comput. Soc. Press.

[107] Oxford English Dictionary (2015). 5th Edition. Retrieved 13 July 2017, <http://www.oed.com/view/Entry/196665>.

[108] Pan, Y.-F., C.-L. Liu, and X. Hou (2010). Fast scene text localization by learning-based filtering and verification. In *2010 17th IEEE International Conference on Image Processing (ICIP 2010)*, pp. 2269–2272. IEEE.

[109] Phan, T. Q., P. Shivakumara, and C. L. Tan (2009). A Laplacian Method for Video Text Detection. In *2009 10th International Conference on Document Analysis and Recognition*, pp. 66–70. IEEE.

[110] Ren, S., K. He, R. B. Girshick, and J. S. 0001 (2017). Faster R-CNN - Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Trans. Pattern Anal. Mach. Intell.*.

[111] Rijsbergen, C. J. V. (1979, January). *Information Retrieval*. Butterworth-Heinemann.

[112] Russell, B. C., A. T. 0001, K. P. Murphy, and W. T. Freeman (2008). LabelMe - A Database and Web-Based Tool for Image Annotation. *International Journal of Computer Vision*.

[113] Sarkar, S. and K. L. Boyer (1996). Quantitative Measures of Change based on Feature Organization - Eigenvalues and Eigenvectors. *CVPR*, 478–483.

[114] Seo, Y.-W., J. Lee, W. Zhang, and D. Wettergreen (2015). Recognition of Highway Workzones for Reliable Autonomous Driving. *IEEE Transactions on Intelligent Transportation Systems 16*(2), 1–11.

[115] Sermanet, P. and Y. LeCun (2011). Traffic sign recognition with multi-scale Convolutional Networks. *IJCNN*.

[116] Shahab, A., F. Shafait, and A. Dengel (2011). ICDAR 2011 Robust Reading Competition Challenge 2: Reading Text in Scene Images. In *2011 International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1491–1496. IEEE.

[117] Shivakumara, P., W. Huang, T. Q. Phan, and C. L. Tan (2010). Accurate video text detection through classification of low and high contrast images. *Pattern Recognition*.

[118] Shivakumara, P., T. Q. Phan, and C. L. Tan (2011). A Laplacian Approach to Multi-Oriented Text Detection in Video. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33*(2), 412–419.

[119] Sivic, J. and A. Zisserman (2003). Video Google - A Text Retrieval Approach to Object Matching in Videos. *ICCV*.

[120] Smeulders, A. W. M., M. Worring, S. Santini, A. Gupta, and R. C. Jain (2000). Content-Based Image Retrieval at the End of the Early Years. *IEEE Trans. Pattern Anal. Mach. Intell.*.

[121] Smith, R. (2007). An Overview of the Tesseract OCR Engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2*, pp. 629–633. IEEE.

[122] Smith, R. W. (1987). *The Extraction and Recognition of Text from Multimedia Document Images*. Ph. D. thesis, University of Bristol.

[123] Sobottka, K., H. Bunke, and H. Kronenberg (1999). Identification of Text on Colored Book and Journal Covers. *ICDAR*.

[124] Sochman, J. and J. Matas (2005). WaldBoost ? Learning for Time Constrained Sequential Detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pp. 150–156. IEEE.

[125] Sorokin, A. and D. A. Forsyth (2008). Utility data annotation with Amazon Mechanical Turk. *CVPR Workshops*.

[126] Srivastav, A. and J. Kumar (2008). Text detection in scene images using stroke width and nearest-neighbor constraints. In *TENCON 2008 - 2008 IEEE Region 10 Conference (TENCON)*, pp. 1–5. IEEE.

[127] Subramanian, K., P. Natarajan, M. Decerbo, and D. A. Castañón (2007). Character-Stroke Detection for Text-Localization and Extraction. *ICDAR*.

[128] Sun, Q., Y. Lu, and S. Sun (2010). A Visual Attention Based Approach to Text Extraction. *ICPR*.

[129] Sung, K. K. and T. A. Poggio (1998). Example-Based Learning for View-Based Human Face Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*.

[130] Takacs, G., V. Chandrasekhar, N. Gelfand, Y. Xiong, W.-C. Chen, T. Bismpigiannis, R. Grzeszczuk, K. Pulli, and B. Girod (2008). Outdoors augmented reality on mobile phone using loxel-based visual feature organization. *Multimedia Information Retrieval*.

[131] Torresen, J., J. W. Bakke, and L. Sekanina (2004). Efficient recognition of speed limit signs. In *The 7th International IEEE Conference on Intelligent Transportation Systems*, pp. 652–656. IEEE.

[132] Tsai, S. S., D. M. Chen, V. Chandrasekhar, G. Takacs, N.-M. Cheung, R. Vedantham, R. Grzeszczuk, and B. Girod (2010). Mobile product recognition. *ACM Multimedia*.

[133] Tu, Z., X. Chen, A. L. Yuille, and S. C. Zhu (2003). Image Parsing - Unifying Segmentation, Detection, and Recognition. *ICCV*.

[134] Vapnik, V. (1999, November). *The Nature of Statistical Learning Theory*. Springer Science & Business Media.

[135] Veit, A., T. Matera, L. Neumann, J. Matas, and S. J. Belongie (2016). COCO-Text - Dataset and Benchmark for Text Detection and Recognition in Natural Images. *CoRR*.

[136] Viola, P. A., M. J. Jones, and D. Snow (2003). Detecting Pedestrians Using Patterns of Motion and Appearance. *ICCV*.

[137] Vondrick, C., D. Patterson, and D. Ramanan. Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, 1–21. 10.1007/s11263-012-0564-1.

[138] Wang, K., B. Babenko, and S. J. Belongie (2011). End-to-end scene text recognition. *ICCV*.

[139] Wang, X., T. X. Han, and S. Yan (2009). An HOG-LBP human detector with partial occlusion handling. *ICCV*.

[140] Wang, X., L. Huang, and C. Liu (2009). A New Block Partitioned Text Feature for Text Verification. In *2009 10th International Conference on Document Analysis and Recognition*, pp. 366–370. IEEE.

[141] Wickham, H. (2007). Reshaping data with the reshape package. *Journal of Statistical Software*.

[142] Wickham, H. (2010, January). A Layered Grammar of Graphics. *Journal of Computational and Graphical Statistics 19*(1), 3–28.

[143] Wolf, C. and J.-M. Jolion (2006, April). Object count/area graphs for the evaluation of object detection and segmentation algorithms. *IJDAR 8*(4), 280–296.

[144] Wong, S. C., A. Gatt, V. Stamatescu, and M. D. McDonnell (2016). Understanding data augmentation for classification - when to warp? *CoRR cs.CV*, arXiv:1609.08764.

[145] Xiao, J., J. Hays, K. A. Ehinger, A. Oliva, and A. T. 0001 (2010). SUN database - Large-scale scene recognition from abbey to zoo. *CVPR*.

[146] Yaeger, L. S., R. F. Lyon, and B. J. Webb (1996). Effective Training of a Neural Network Character Classifier for Word Recognition. *NIPS*.

[147] Ye, Q. and D. Doermann (2014, December). Text Detection and Recognition in Imagery: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence 37*(7), 1480–1500.

[148] Ye, Q., Q. Huang, W. G. 0001, and D. Zhao (2005). Fast and robust text detection in images and video frames. *Image Vision Comput.*.

[149] Zagoruyko, S. and N. Komodakis (2015). Learning to compare image patches via convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4353–4361. IEEE.

[150] Zhang, J. and R. Kasturi (2008). Extraction of Text Objects in Video Documents: Recent Progress. In *2008 The Eighth IAPR International Workshop on Document Analysis Systems (DAS)*, pp. 5–17. IEEE.

[151] Zhang, J. and R. Kasturi (2010). Text Detection Using Edge Gradient and Graph Spectrum. *ICPR*.

[152] Zhang, J. and R. Kasturi (2011). Character Energy and Link Energy-Based Text Extraction in Scene Images. In *Computer Vision – ACCV 2010*, pp. 308–320. Berlin, Heidelberg: Springer Berlin Heidelberg.

[153] Zhu, L., C.-S. Yang, and J.-S. Pan (2016). Detection and Recognition of Speed Limit Sign from Video. *ACIIDS*.

Appendix A

Ethics Clearance

Professor Rajesh Vasa
Associate Professor Andrew Cain
Mr Alex Cummaudo
School of Information Technology
Burwood Campus

9 August 2017

Dear Rajesh, Andrew and Alex

STEC-49-2017-CUMMAUDO titled *"Recognition and Prominence Ranking of Alphanumeric Number Sequences in Images."*

Thank you for submitting the above project for consideration by the Faculty Human Ethics Advisory Group (HEAG). The HEAG recognised that the project complies with the National Statement on Ethical Conduct in Human Research (2007) and has approved it. You may commence the project upon receipt of this communication.

The approval period is for three years. It is your responsibility to contact the Faculty HEAG immediately should any of the following occur:

- Serious or unexpected adverse effects on the participants
- Any proposed changes in the protocol, including extensions of time
- Any changes to the research team or changes to contact details
- Any events which might affect the continuing ethical acceptability of the project
- The project is discontinued before the expected date of completion.

You will be required to submit an annual report giving details of the progress of your research. Please forward your first annual report on 9/8/18 Failure to do so may result in the termination of the project. Once the project is completed, you will be required to submit a final report informing the HEAG of its completion.

Please ensure that the Deakin logo is on the Plain Language Statement and Consent Forms. You should also ensure that the project ID is inserted in the complaints clause on the Plain Language Statement, and be reminded that the project number must always be quoted in any communication with the HEAG to avoid delays. All communication should be directed to sciethic@deakin.edu.au

The Faculty HEAG and/or Deakin University Human Research Ethics Committee (HREC) may need to audit this project as part of the requirements for monitoring set out in the National Statement on Ethical Conduct in Human Research (2007).

If you have any queries in the future, please do not hesitate to contact me.

We wish you well with your research.

Kind regards

Teresa Treffry

Teresa Treffry
Secretary, Human Ethics Advisory Group (HEAG)
Faculty of Science Engineering & Built Environment

Appendix B

Metamodel Class Diagrams

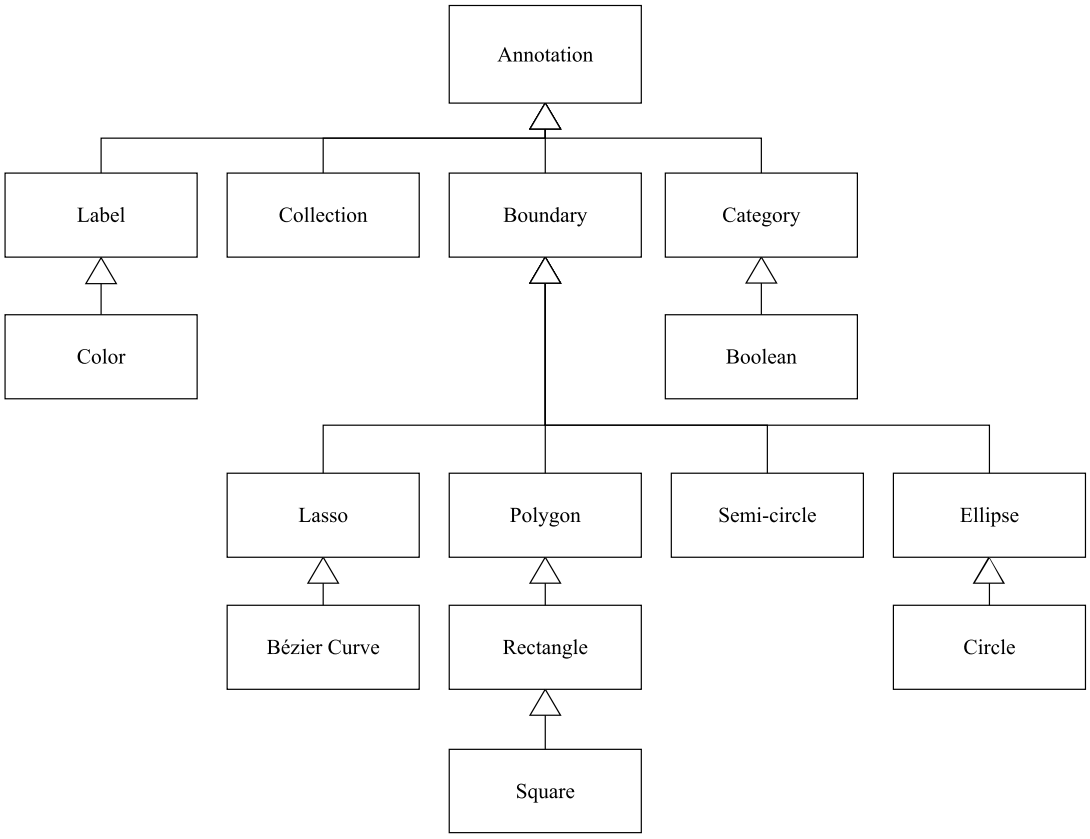


Figure B.1: Type class hierarchy of annotations

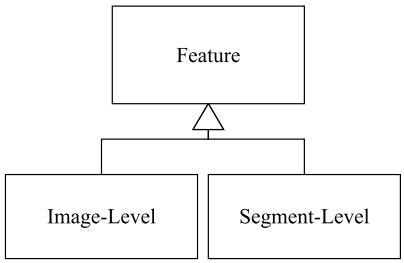


Figure B.2: Type class hierarchy of features

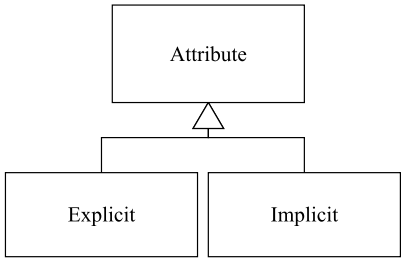


Figure B.3: Type class hierarchy of attributes

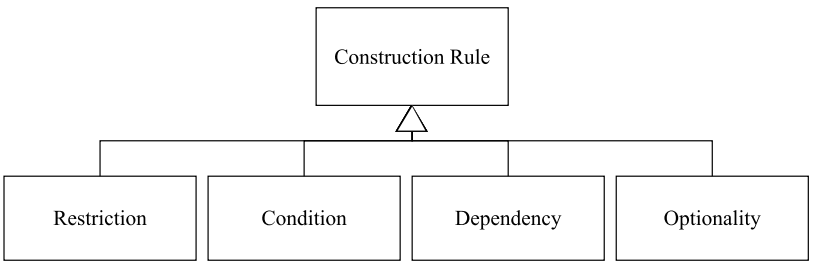


Figure B.4: Type class hierarchy of construction rules

Appendix C

Dataset Schemas

Listing C.1: MS COCO JavaScript Object Notation (JSON) Schema, sourced from `http://mscoco.org/dataset/#download`.

```
{
  "info" : info,
  "images" : [image],
  "annotations" : [annotation],
  "licenses" : [license],
}

info {
  "year" : int,
  "version" : str,
  "description" : str,
  "contributor" : str,
  "url" : str,
  "date_created" : datetime,
}

image {
  "id" : int,
  "width" : int,
  "height" : int,
  "file_name" : str,
  "license" : int,
  "flickr_url" : str,
  "coco_url" : str,
  "date_captured" : datetime,
}

license {
  "id" : int,
  "name" : str,
  "url" : str,
}

// Object Instance Annotations

annotation {
  "id" : int,
  "image_id" : int,
  "category_id" : int,
  "segmentation" : RLE or [polygon],
```

```
  "area" : float,
  "bbox" : [x,y,width,height],
  "iscrowd" : 0 or 1,
}

categories [{
  "id" : int,
  "name" : str,
  "supercategory" : str,
}]

// Object Keypoint Annotations

annotation {
  "keypoints" : [x1,y1,v1,...],
  "num_keypoints" : int,
  "[cloned]" : ...,
}

categories [{
  "keypoints" : [str],
  "skeleton" : [edge],
  "[cloned]" : ...,
}]

// Image Caption Annotations

annotation {
  "id" : int,
  "image_id" : int,
  "caption" : str,
}
```

Listing C.2: COCO-Text JSON Schema, sourced from `https://vision.cornell.edu/se3/coco-text-2/`.

```
{
  "info" : info,
  "imgs" : [image],
  "anns" : [annotation]
}

info {
  "version" : str,
  "description" : str,
  "author" : str,
  "url" : str,
  "date_created" : datetime
}

image {
  "id" : int,
  "file_name" : str,
  "width" : int,
  "height" : int,
  "set" : str // train or val
}

annotation {
  "id" : int,
  "image_id" : int,
  "class" : str, // machine printed or handwritten or others
  "legibility" : str, // legible or illegible
  "language" : str, // english or not english or na
  "area" : float,
  "bbox" : [x,y,width,height],
  "utf8_string" : str,
  "polygon" : []
}
```

Listing C.3: XML Schema Definition (XSD) of the ICDAR 2011–2015 annotation schema, sourced within `http://www.cvc.uab.es/apep/downloads.php`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="textGt">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="textLine" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="width" type="xs:integer" use="required" />
      <xs:attribute name="height" type="xs:integer" use="required" />
      <xs:attribute name="imageFileName" type="xs:string" use="required" />
      <xs:attribute name="areaImageFilename" type="xs:string" use="optional
        ↪ " />
      <xs:attribute name="skeletonImageFilename" type="xs:string" use="
        ↪ optional" />
      <xs:attribute name="docVersion" type="xs:integer" use="required"
        ↪ fixed="3" />
      <xs:attribute name="author" type="xs:string" use="required" />
      <xs:attribute name="comments" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>
  <xs:element name="textLine">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="word" minOccurs="0" maxOccurs="unbounded" />
        <xs:element ref="atom" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="id" type="xs:integer" use="required" />
      <xs:attribute name="transcription" type="xs:string" use="required" />
      <xs:attribute name="xmin" type="xs:integer" use="required" />
      <xs:attribute name="xmax" type="xs:integer" use="required" />
      <xs:attribute name="ymin" type="xs:integer" use="required" />
      <xs:attribute name="ymax" type="xs:integer" use="required" />
      <xs:attribute name="dontCare" type="xs:boolean" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="word">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="atom" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
```

```
38      <xs:attribute name="id" type="xs:integer" use="required" />
39      <xs:attribute name="transcription" type="xs:string" use="required" />
40      <xs:attribute name="xmin" type="xs:integer" use="required" />
41      <xs:attribute name="xmax" type="xs:integer" use="required" />
42      <xs:attribute name="ymin" type="xs:integer" use="required" />
43      <xs:attribute name="ymax" type="xs:integer" use="required" />
44      <xs:attribute name="dontCare" type="xs:boolean" use="required" />
45    </xs:complexType>
46  </xs:element>
47  <xs:simpleType name="colorHex">
48    <xs:restriction base="xs:string">
49      <xs:pattern value="#[0-9a-fA-F]{6}"/>
50    </xs:restriction>
51  </xs:simpleType>
52  <xs:element name="atom">
53    <xs:complexType>
54      <xs:attribute name="id" type="xs:integer" use="required" />
55      <xs:attribute name="transcription" type="xs:string" use="required" />
56      <xs:attribute name="splitByDesign" type="xs:boolean" use="required" /
            ↪ >
57      <xs:attribute name="mergedByDesign" type="xs:boolean" use="required"
            ↪ />
58      <xs:attribute name="color" type="colorHex" use="required" />
59      <xs:attribute name="dontCare" type="xs:boolean" use="required" />
60      <xs:attribute name="textParts" type="xs:integer" use="required" />
61      <xs:attribute name="xmin" type="xs:integer" use="required" />
62      <xs:attribute name="xmax" type="xs:integer" use="required" />
63      <xs:attribute name="ymin" type="xs:integer" use="required" />
64      <xs:attribute name="ymax" type="xs:integer" use="required" />
65    </xs:complexType>
66  </xs:element>
67 </xs:schema>
```

Listing C.4: Sample annotation file of the ICDAR 2003–2011 annotation schema, from [93].

```
<tagset>
  <image>
    <imageName>scene/ComputerScienceSmall.jpg</imageName>
    <resolution x="338" y="255" />
    <taggedRectangles>
      <taggedRectangle x="99" y="94" width="128" height="20" offset="0"
          ↪ rotation="0">
      <tag>Department</tag>
      <segmentation>
        <xOff>16</xOff>
        <xOff>29</xOff>
        <xOff>43</xOff>
        <xOff>54</xOff>
        <xOff>64</xOff>
        <xOff>74</xOff>
        <xOff>93</xOff>
        <xOff>106</xOff>
        <xOff>117</xOff>
      </segmentation>
    </taggedRectangle>
    ...
  </image>
...
</tagset>
```

Listing C.5: Sample annotation file of the PASCAL VOC 2007 annotation schema, from `http://host.robots.ox.ac.uk/pascal/VOC/voc2007`.

```
<annotation>
  <folder>VOC2012</folder>
  <filename>2007_000032.jpg</filename>
  <source>
    <database>The VOC2007 Database</database>
    <annotation>PASCAL VOC2007</annotation>
    <image>flickr</image>
  </source>
  <size>
    <width>500</width>
    <height>281</height>
    <depth>3</depth>
  </size>
  <segmented>1</segmented>
  <object>
    <name>aeroplane</name>
    <pose>Frontal</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>104</xmin>
      <ymin>78</ymin>
      <xmax>375</xmax>
      <ymax>183</ymax>
    </bndbox>
  </object>
  <object>
    <name>aeroplane</name>
    <pose>Left</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>133</xmin>
      <ymin>88</ymin>
      <xmax>197</xmax>
      <ymax>123</ymax>
    </bndbox>
  </object>
  <object>
    <name>person</name>
```

```
    <pose>Rear</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>195</xmin>
      <ymin>180</ymin>
      <xmax>213</xmax>
      <ymax>229</ymax>
    </bndbox>
  </object>
  <object>
    <name>person</name>
    <pose>Rear</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>26</xmin>
      <ymin>189</ymin>
      <xmax>44</xmax>
      <ymax>238</ymax>
    </bndbox>
  </object>
</annotation>
```

Listing C.6: Sample annotation file of the PASCAL VOC 2012 annotation schema (extended from 2007), from `http://host.robots.ox.ac.uk/pascal/VOC/voc2012`.

```
<annotation>
  <filename>2012_004317.jpg</filename>
  <folder>VOC2012</folder>
  <object>
    <name>person</name>
    <actions>
      <jumping>0</jumping>
      <other>0</other>
      <phoning>0</phoning>
      <playinginstrument>0</playinginstrument>
      <reading>0</reading>
      <ridingbike>0</ridingbike>
      <ridinghorse>0</ridinghorse>
      <running>0</running>
      <takingphoto>0</takingphoto>
      <usingcomputer>1</usingcomputer>
      <walking>0</walking>
    </actions>
    <bndbox>
      <xmax>170</xmax>
      <xmin>64</xmin>
      <ymax>310</ymax>
      <ymin>131</ymin>
    </bndbox>
    <difficult>0</difficult>
    <pose>Unspecified</pose>
    <point>
      <x>109</x>
      <y>193</y>
    </point>
  </object>
  <segmented>0</segmented>
  <size>
    <depth>3</depth>
    <height>500</height>
    <width>375</width>
  </size>
  <source>
    <annotation>PASCAL VOC2012</annotation>
    <database>The VOC2012 Database</database>
```

```
41      <image>flickr</image>
42    </source>
43  </annotation>
```

Appendix D

Dataset Mappings

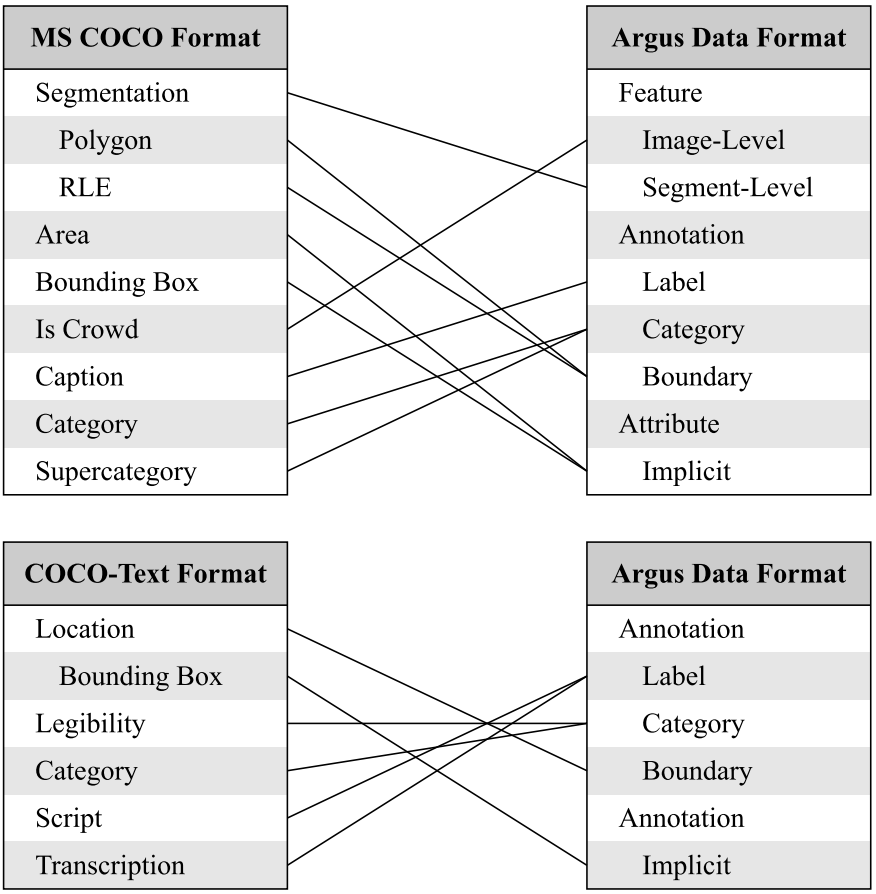


Figure D.1: Annotation components of both MS-COCO and COCO-text (left) and ADF (right).

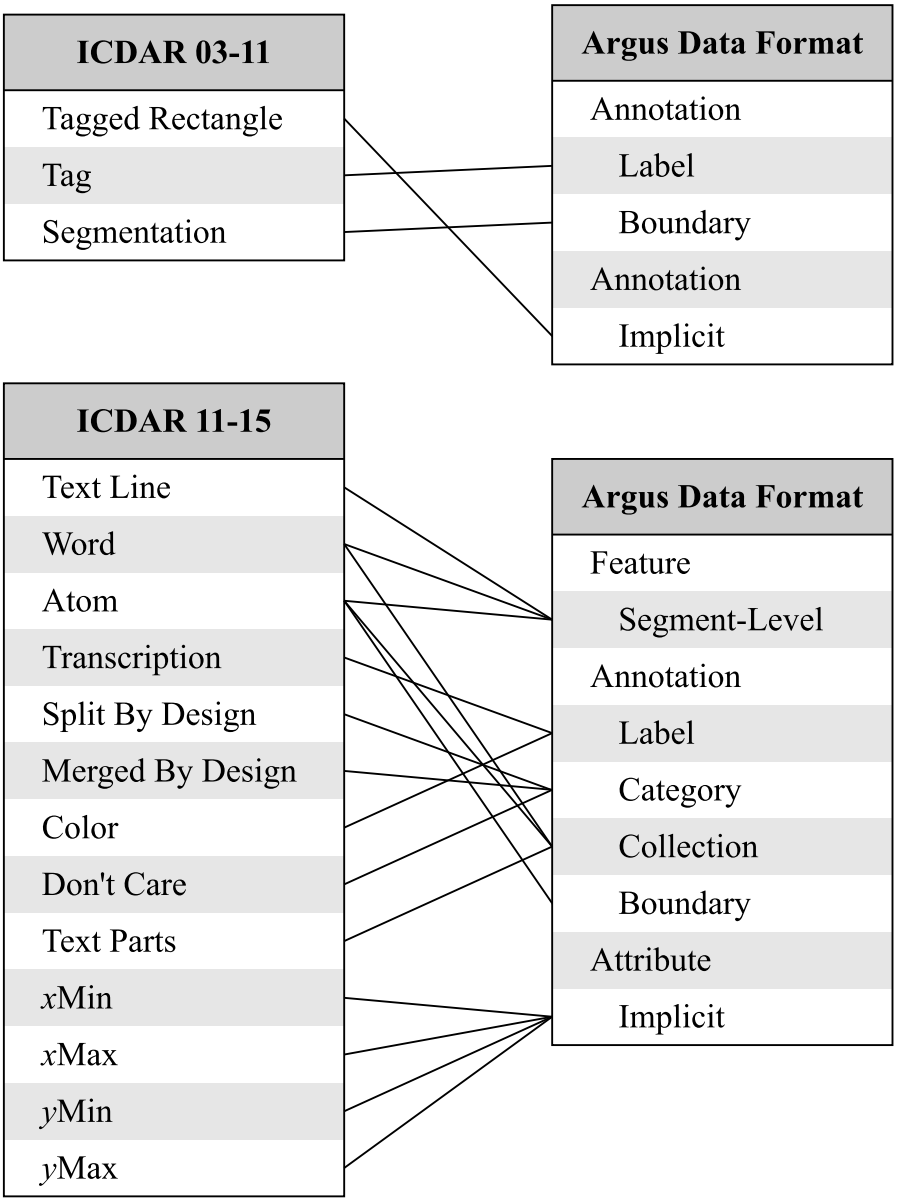


Figure D.2: Annotation components of the ICDAR 03–11 and 11–15 formats (left) with ADF (right).

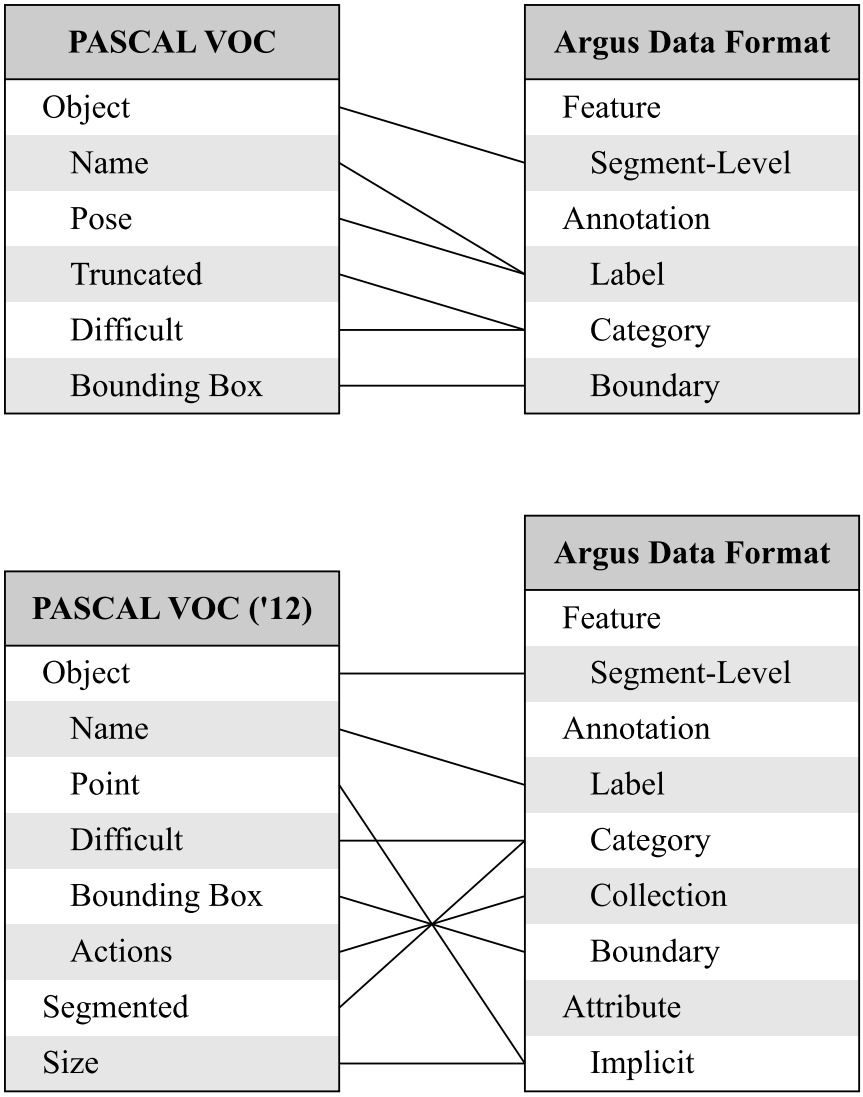


Figure D.3: Annotation components of the PASCAL VOC formats (left) with ADF (right).

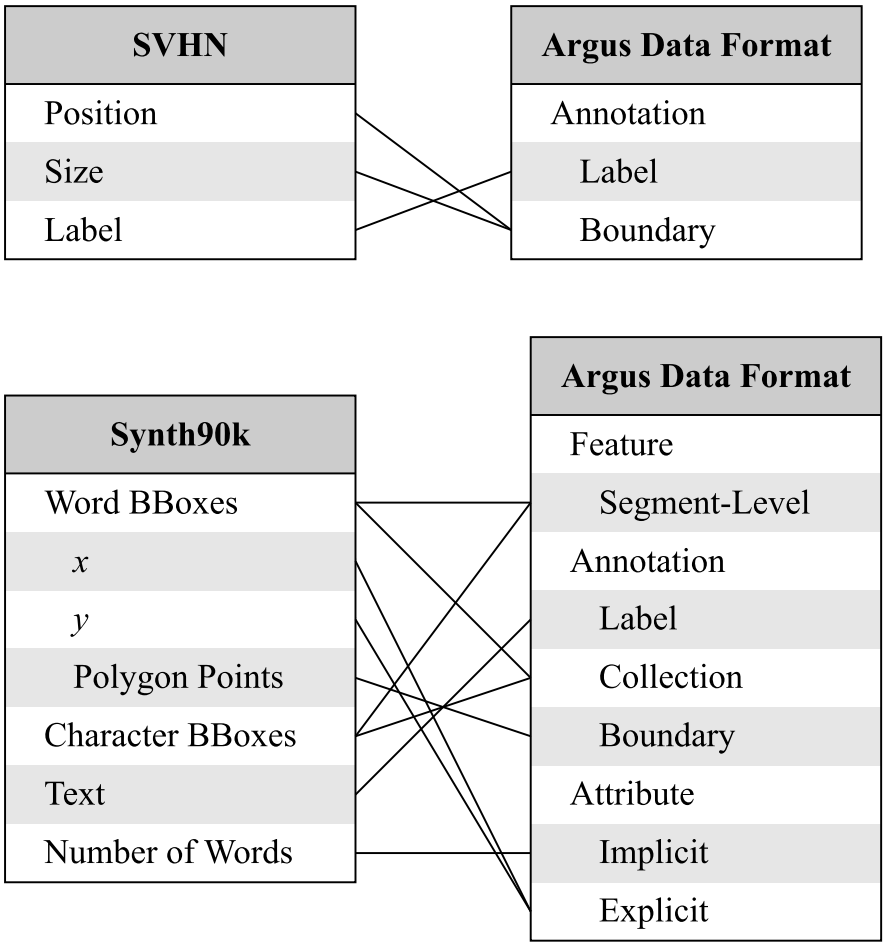


Figure D.4: Annotation components of binary-encoded MATLAB formats of the SVHN and Synth90k datasets (left) with ADF (right).