Deep Learning for Text Spotting

D.Phil Thesis

Robotics Research Group
Department of Engineering Science
University of Oxford

Supervisors:
Professor Andrew Zisserman
Doctor Andrea Vedaldi

Max Jaderberg
Worcester College

Michaelmas Term, 2014

Max Jaderberg
Worcester College

Doctor of Philosophy
Michaelmas Term, 2014

Deep Learning for Text Spotting

Abstract

This thesis addresses the problem of text spotting – being able to automatically detect and recognise text in natural images. Developing text spotting systems, systems capable of reading and therefore better interpreting the visual world, is a challenging but wildly useful task to solve. We approach this problem by drawing on the successful developments in machine learning, in particular deep learning and neural networks, to present advancements using these data-driven methods.

Deep learning based models, consisting of millions of trainable parameters, require a lot of data to train effectively. To meet the requirements of these data hungry algorithms, we present two methods of automatically generating extra training data without any additional human interaction. The first crawls a photo sharing website and uses a weakly-supervised existing text spotting system to harvest new data. The second is a synthetic data generation engine, capable of generating unlimited amounts of realistic looking text images, that can be solely relied upon for training text recognition models. While we define these new datasets, all our methods are also evaluated on standard public benchmark datasets.

We develop two approaches to text spotting: character-centric and word-centric. In the character-centric approach, multiple character classifier models are developed, reinforcing each other through a feature sharing framework. These character models are used to generate text saliency maps to drive detection, and convolved with detection regions to enable text recognition, producing an end-to-end system with state-of-the-art performance. For the second, higher-level, word-centric approach to text spotting, weak detection models are constructed to find potential instances of words in images, which are subsequently refined and adjusted with a classifier and deep coordinate regressor. A whole word image recognition model recognises words from a huge dictionary of 90k words using classification, resulting in previously unattainable levels of accuracy. The resulting end-to-end text spotting pipeline advances the state of the art significantly and is applied to large scale video search.

While dictionary based text recognition is useful and powerful, the need for unconstrained text recognition still prevails. We develop a two-part model for text recognition, with the complementary parts combined in a graphical model and trained using a structured output learning framework adapted to deep learning. The trained recognition model is capable of accurately recognising unseen and completely random text.

Finally, we make a general contribution to improve the efficiency of convolutional neural networks. Our low-rank approximation schemes can be utilised to greatly reduce the number of computations required for inference. These are applied to various existing models, resulting in real-world speedups with negligible loss in predictive power.

This thesis is submitted to the Department of Engineering Science, University of Oxford, in fulfilment of the requirements for the degree of Doctor of Philosophy. This thesis is entirely my own work, and except where otherwise stated, describes my own research.

Max Jaderberg, Worcester College

Acknowledgements

I am exceptionally thankful for the help and support I have had from my supervisors, Andrew Zisserman and Andrea Vedaldi. Their initial faith towards me undertaking a DPhil is something I will always be grateful for, and their continual enthusiasm and optimism has been a big part of making this possible. I would also like to thank Karen Simonyan as a colleague and good friend for all his help, collaboration, and late night ideas.

The environment in the lab in Oxford has been fantastic due to the people in VGG. Thanks to everyone there, including but not limited to Amr, Relja, Ken, Meelis, Elliot (my engineering and VGG twin), Omkar, Tomas, Karel, Carlos, Chai, Yusuf, Mircea, and Ondrej. It also would never have been possible to have completed this work without the distractions from my amazing housemates, Mike, Flo, and Swetlana, as well as from the other beautiful people of Worcester College MCR.

Most importantly, I would like to thank my family for their years of support, motivation, and belief.

As a final note, thanks to Floating Points, Roche Musique, and Laid Back Radio for providing a lot of the soundtrack to my research and the energy to keep working.

Contents

Chapter 1

Introduction

1.1 Objective

This thesis addresses the problem of *text spotting*. At a high level, text spotting refers to being able to automatically detect and recognise text in images, an algorithm that is able to "read". We do not impose a constraint on the type of text, or the type of scene that should be able to be read – if a human can read it, a text spotting system should also aim to be able to read it. This warrants the development of general text spotting algorithms, which are not optimised for a specific domain, to be able to handle both an image of a newspaper front page as well as a busy high street scene, with text in the form of regular printed fonts, through to hand-written calligraphy. The goal of this thesis is to advance the state of the art of such text spotting algorithms.

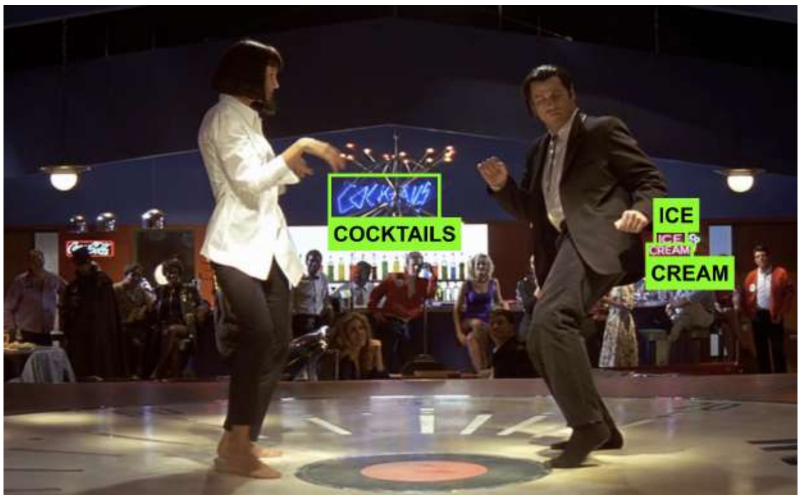
Extracting any sort of semantic information from the raw volume of pixel values of a digital image is not a trivial task, requiring multiple levels of feature extraction and inference. Throughout the last few decades, a multitude of feature extraction and inference tools have been proposed for a range of computer vision problems, including text spotting. In this work we focus on the use of deep learning based approaches for text spotting, reducing the reliance on hand-tuned image representations, showing superior performance in our application to text spotting.

The process of text spotting is mainly decomposed into two stages – text detection followed by text recognition (Figure 1.1). *Text detection* refers to finding and localising instances of text (for example words) from the image, correctly ignoring background clutter. *Text recognition* refers to the visual decoding of a localised, cropped instance of text (a word image) into the string of characters depicted. While these two stages are not necessarily wholly distinct (*i.e.* the result of the text recognition stage can be used to further improve the initial results of the text detection stage), in this work and when describing previous text spotting methods we generally follow this separation. This separation makes sense computationally, since the inference associated with detection is often very different and less computationally expensive than that associated with recognition.

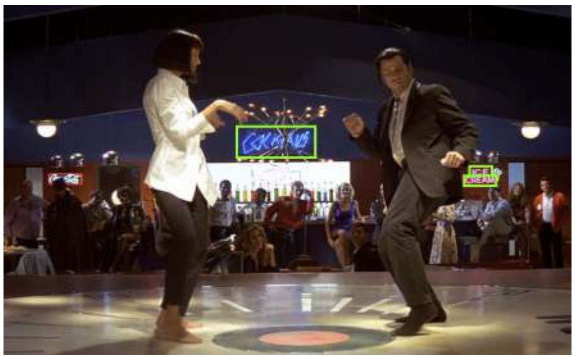
1.2 Motivation and Applications

The automatic detection and recognition of text in natural images, text spotting, is an important challenge for visual understanding.

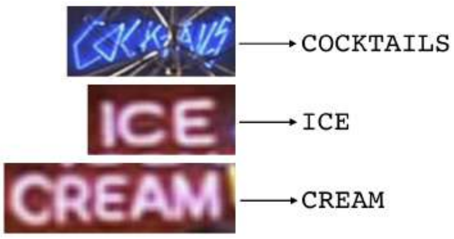
The human invention of language, and the cognitive ability to conceptualise, process, and communicate ideas through language, is one of the defining abilities of our species, and is associated with the outstanding intelligence of humans [Roth and Black, 2005]. Text, as the physical incarnation of language, is one of the basic tools for preserving and communicating information and ideas – indeed, text systems have been in existence for many millennia, since 6600 BC, when graphical symbols were carved in tortoise shells [Li and Wang, 2003]. Today, much of the modern world is designed to be interpreted through the use of labels and other textual cues, and so text finds itself scattered throughout many images and videos, and can often be the main semantic focus of an image. It is therefore imperative to create a visual



(a)



(b)



(c)

Figure 1.1: An example illustration of text spotting applied to an image (a). The text spotting process is decomposed in to text detection (b) which detects and localises words (word bounding boxes are shown in green), followed by text recognition (c) which recognises the string of characters depicted in each word image.

interface with text to allow rich machine interaction with the human world.

Through the use of text spotting, we can enable automated systems to decode the text within images and videos, allowing a greater understanding of media. This results in a number of direct applications, with some examples given below.

Automatic Annotation. A direct use of text spotting systems is for the automatic annotation of images and videos. Over the past decade, there has been an explosion in the quantity of videos and images created and shared in digital format. For example, every day 150k hours of video are uploaded to the popular video sharing site Youtube and 350 million photos are uploaded to the social network Facebook. However, most of the information contained in this data is not easily accessible. While occasionally images and videos have associated textual metadata, this is usually sparse or absent; as such, the vast majority of the semantic content of images and videos can be accessed only be decoding the information directly from the pixels. The latter is still the major bottleneck towards better and more efficient data indexing, cataloguing, and processing. Currently, it is largely the job of human workers to annotate the text in visual media by hand, however the use of text spotting systems can greatly speed up this human annotation process, and if accurate enough could even replace human workers altogether, greatly reducing the cost of annotation.

Visual Media Search. A further application of being able to automatically read the text within visual media is to allow the large scale search and retrieval of this media. Text spotting systems can be used to build semantic representations of images and videos, which can be indexed to allow very fast retrieval by query terms. As an example, Chapter 8 describes our own application to allow the search of text within large video archives.

The use of the information gleaned from text spotting systems could also be used for more general visual search, with the text contained within an image used in conjunction with the features of a more traditional instance search methods (*e.g.* Arandjelović and Zisserman, 2012), to allow searching the billions of consumer photos in existence for their visual content.

Robotics and Assistive Systems. Another area where text spotting is of great use is in robotics [Posner et al., 2010]. By being able to interpret signage with text spotting, robots may be able to understand the environment they are in, how they can interact with the environment, and even use the text for localisation. For example, the indoor environments of many buildings and offices are very bare and

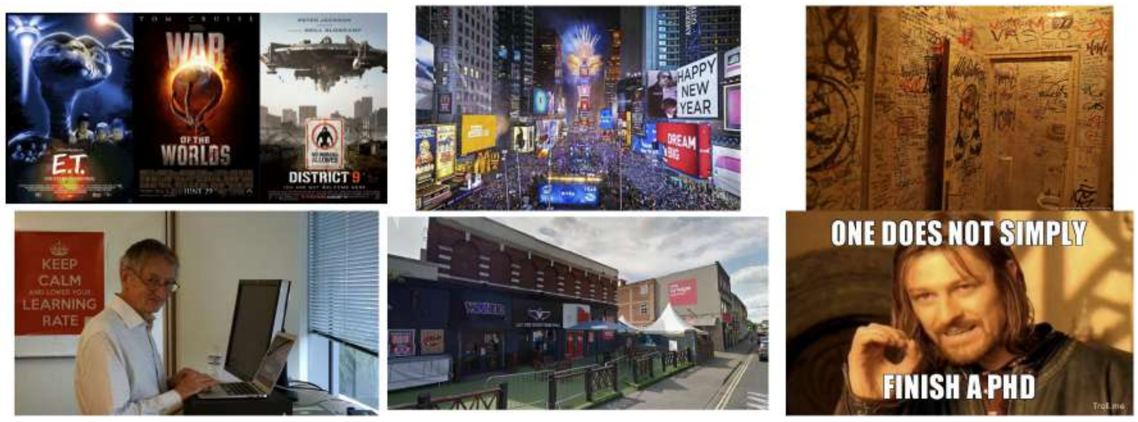


Figure 1.2: Text is found in a variety of different formats and scenarios in images and videos. The range of fonts, textures, orientations, noise, and background objects are some of the challenges associated with text spotting. However, this is a useful problem to solve, as often it is the text contained which captures the semantics of an image.

repetitive, where perhaps the main discriminative factor for localisation are signs containing different textual labels. Also when navigating the streets, a self driving car or robot may need to read temporarily placed signs to understand the changing rules of the road.

While text spotting can assists robots, it could also be used to assist humans. For visually impaired people and those who can't read a certain script (for example when reading another language), text spotting together with a text-to-speech system would allow a user to hear the text that appears in the world which they are unable to see [Markoff, 2013].

1.3 Challenges and Tasks

The definition of what the term "text" encompasses is somewhat ambiguous. For the purposes of this thesis, "text" will be defined as the visual occurrence of a collection of coincident Latin characters. While there are many cases of single character instances of text in the world, due to the simple shapes of many characters, it can be very difficult to differentiate between a purposeful rendering of a single character in the world, and a chance depiction of a character from other visual elements and structures – *e.g.* "l" can seem to appear in an image from an isolated vertical bar. For this reason, we focus on methods for text spotting text containing more than one character. Also, although in this thesis we limit ourselves to modelling Latin characters, the methods described should be able to be extended to any glyph-based text system (for example Chinese or Russian).

While the seemingly related problem of digitising text from planar, paper-based documents, often referred to as optical character recognition (OCR), is well studied and largely solved, general text spotting is still a challenging, unsolved problem. When applied to natural scene images, document OCR techniques fail as they are tuned to the largely black-and-white, line-based environment of printed documents. The text that occurs in natural scene images is hugely variable in appearance and layout, being drawn from a large number of fonts and styles, suffering from inconsistent lighting, occlusions, orientations, noise, and, in addition, the presence of background objects causes spurious false-positive detections. This places text spotting as a separate, far more challenging problem than document OCR (which is encompassed by text spotting itself). Figure 1.2 shows examples of the range of text that occurs in images that should be able to be read with a text spotting system. We now elaborate on the challenges associated with the two stages of text spotting.

Text Detection. Detecting text in natural scenes is a difficult task. Natural scenes contain lots of different types of objects, both text and non-text objects, which are composed of many small shapes and strokes. When looking at text and characters, the simple shapes that represent them are actually very common in natural scenes, and so it is very difficult to discriminate between text-like objects in images that are due to text and those that are products of background objects and imagery. This leads to the problem of false-positive text detection, and so a major challenge is to correctly disambiguate background noise from true text and create a system with *high precision.* Another aspect to the problem is to ensure that all instances of text are correctly found and localised rather than rejected as background noise, which is especially hard when characters are occluded or imaging noise, lighting, and texture distort their appearance. This dual problem – seeking a system which finds all instances of text – warrants a system with *high recall.* Achieving an ideal high precision, high recall text detection system is therefore a big challenge.

Text Recognition. To perform text recognition, a system must correctly identify the sequence of characters depicted in a word image. This is not a trivial task: text in scenes comes from a huge variety of different fonts or is even handwritten, so characters can look vastly different. In addition to the issue of different fonts and visual styles, the text is rendered on different surfaces, with different textures and colours, and can fall under varying lighting conditions across a single word or character, creating varying visual properties and distortions to the characters. The camera taking the image is usually not fronto-parallel with the surface the text is rendered on, meaning there are perspective distortions, as well as rotation or curvature to the baseline of the text, and the limit of the camera itself means text may be blurred, noisy, and low resolution. These issues pose a significant challenge to text recognition.

Since all characters of a word image must be correctly recognised, the error for recognising each character is compounded – the incorrect classification of a single character in a word will produce an incorrect recognition result. For this reason, the problem of text recognition in natural images is sometimes simplified by utilising a *fixed lexicon* or dictionary (the terms “lexicon” and “dictionary” are used interchangeably in this thesis). This language constrained recognition (which will simply be referred to as *constrained recognition*) constitutes an easier problem as the solution could be found by evaluating the suitability of each word in the lexicon exhaustively and choosing the most suitable lexicon word. Exhaustive search is not possible when performing text recognition without a lexicon (*unconstrained recognition*), as evaluating all possible combinations of characters and word lengths is a combinatorially hard problem.

1.4 Contributions

In this section, we list the main contributions made in this thesis.

1. **Automatically mining real-world data.** We introduce a method to automatically generate word and character level bounding box annotations from photos uploaded to a photo sharing website. This system can crawl the photos of the website, and searches using a weak baseline text spotting system to generate text annotations. These predicted annotations are compared to the user-given metadata about the image, with matching results assumed to be correct. This allows the generation of thousands of extra word and character data samples for training without any human interaction. Additionally, we also show that given only word level annotations for images, character level annotations can be automatically generated using a word fitting algorithm and a pre-trained character classifier. These methods are described in Section 3.2.

2. **Synthetic data generation engine.** In Section 3.3 we present our synthetic data generator. This method renders realistic looking word image samples of words from a dictionary or completely random strings. The process uses a multitude of pre-defined probability distributions which are sampled to govern the rendering of different fonts, character layouts, 3D projections, noise, and blending with natural images. This results in word image samples that are realistic enough to be used for training word recognition systems for real-world text recognition. This process is a major contribution in enabling the

training of the large text recognition models in Chapter 5 and Chapter 6, by generating huge synthetic training datasets of millions of word images.

3. **Feature learning for character-centric text spotting.** We show that convolutional neural networks (a machine learning framework introduced in Section 2.1) can be trained very effectively for character recognition and detection. We learn a set of visual features for text by sharing features between different classification tasks, allowing the data and labels for the different tasks to be combined in order to learn a more highly discriminative set of features. The resulting classifiers using these features achieve state-of-the-art results on standard character recognition benchmarks. This is described in Chapter 4, along with an end-to-end text spotting system based on these classifiers. This end-to-end system uses a binary text/no-text classifier to drive text detection, and the integration of multiple character classifiers and a bigram classifier to create a lexically constrained text recognition method.

4. **Word region proposal based text detection.** A new approach to text detection is proposed in Chapter 5. Fast region proposal methods are used to generate proposal word detections. This gives very high recall but also produces many false-positive detections. The false-positives are filtered with a subsequent classifier, and the remaining detections fine-tuned in a regression framework. The advantage of this method over previous detection methods is the ability to achieve much higher recall while maintaining high recall.

5. **Dictionary constrained text recognition.** Using the data generated by the synthetic text engine from Section 3.3, we propose a new method for text recognition in Section 5.4. This involves training a deep convolutional neural network for word classification, classifying the input, a whole word image, into a huge dictionary of 90k English words. The recognition accuracy of this

system is shown to be far higher than previous methods. It is combined with the new text detection method described in the previous paragraph to create an end-to-end text spotting system that performs exceptionally well. The complete system is given in Chapter 5.

6. **Unconstrained text recognition.** In Chapter 6 we seek to remove the constraint on text recognition of having a fixed dictionary of words that can be recognised. Two convolutional neural network models are defined, both taking the whole word image as input, but one predicting the sequence of characters in the image directly, and the other predicting the presence of character N-grams in the image. We show that while these can be used alone for text recognition, combining them and jointly training this formulation results in more accurate text recognition, and does not rely on a dictionary, lexicon, or language model for accurate prediction.

7. **Speeding up convolutional neural networks.** We also depart from the text spotting problem and explore a method for speeding up convolutional neural networks, a large source of computational complexity within our text spotting algorithms. We show that we can use low rank approximations together with a mathematical trick to efficiently compute convolutions with separable kernels in order to gain large speedups in processing with minimal losses in accuracy. We describe this contribution in Chapter 7 and show that this general method can be applied to the character classifiers of Chapter 4 very effectively.

8. **Searching for text in video.** Chapter 8 gives an example application of text spotting for visual media search. Thousands of hours of video footage is processed by the text spotting system introduced in Chapter 5 and indexed, allowing the millions of frames of video to be instantly searched for a user-given

query. The resulting application displays the accurate and scalable nature of our text spotting algorithm, as well as its usefulness for video retrieval.

1.5 Publications

This section gives a list of the publications that contain the content of this thesis.

Published

- **ECCV 2014, Jaderberg et al., 2014a** – This paper covers the end-to-end text spotting system described in Chapter 4, as well as the data mining methods from Section 3.2.
- **BMVC 2014, Jaderberg et al., 2014c** – This paper contains the speedup framework that is presented in Chapter 7.
- **NIPS Deep Learning Workshop 2014, Jaderberg et al., 2014b** – This paper first introduces our synthetic data engine (Section 3.3) as well as the dictionary constrained text recognition method from Section 5.4, the character sequence model (Section 6.1.1), and the N-gram detection model (Section 6.1.2).
- **IJCV May 2015: Jaderberg et al., 2015a** – This paper describes the full word-centric text spotting system of Chapter 5.
- **ICLR 2015: Jaderberg et al., 2015b** – This paper contains the work from Chapter 6 on unconstrained text recognition with structured output learning.

Chapter 2

Literature Review

In this chapter we review the development in methods relating to this thesis. We start with a review of deep learning in Section 2.1 and a number of works making use of deep learning. In particular we focus on convolutional neural networks, their use for a range of vision tasks, and the work on speeding up networks.

In Section 2.2 we provide an overview of this thesis' target problem domain – text spotting. We review a range of different methods for text spotting, and its constituent sub-problems of text detection and text recognition.

2.1 Deep Learning

In this section we discuss *deep learning*. Deep learning can be seen as a form of hierarchical learning, where algorithms make use of multiple layers of representations to gradually transform data into high level concepts. A "deep" architecture is one which is composed of more than one layer of transformations from input to output. Moving from the input, through each layer of transformations, higher level features are derived from the lower level features, leading to a hierarchical representation.

Such a design choice – systems built from multiple levels of representation – is highly motivated from nature. It has been well observed that the mammalian visual

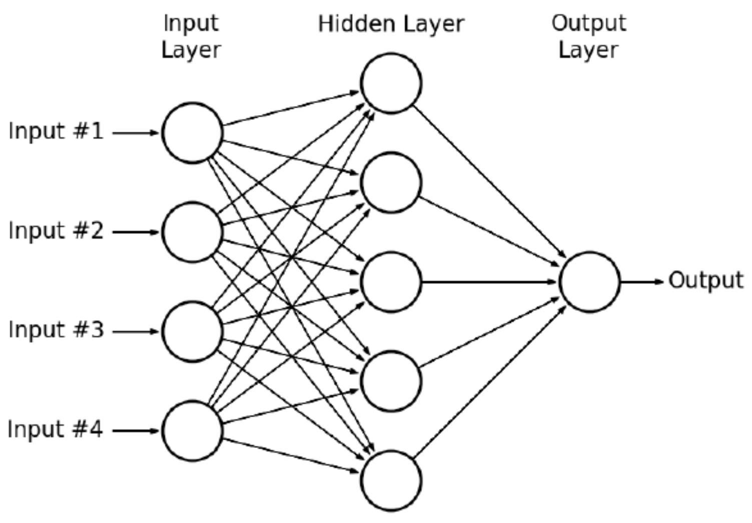


Figure 2.1: A neural network. Each circle corresponds to a neuron, taking a number of inputs and producing a single output. Connections between the neurons form a network, and can take any arbitrary network structure. By arranging neurons in a hierarchical manner, layers of representation can be built. This neural network shown has three layers of neurons, so is referred to as a three layer network. Intermediate layers between the input and output are referred to as *hidden layers*, as their states are not directly observed.

cortex exhibits a layered structure [Hubel and Wiesel, 1962], with layers comprising of groups of neuron cells in the brain. This has led to a number of computational architectures aimed at emulating the brain, with arguably the most successful of those formulations being the introduction of *neural networks* [Rosenblatt, 1958].

Neural networks allow multiple layers of "neurons", where a neuron is a functional unit taking multiple inputs and producing a single scalar output. We will use the terms "neurons" and "units" interchangeably. A layer is comprised of multiple neurons, whose inputs are taken from a previous layer of neurons, and whose outputs are fed to a subsequent layer of neurons. Figure 2.1 illustrates a simple three layer neural network.

While a neural network can take any arbitrary network structure, in this thesis we consider only feed-forward neural networks, whose graph structure takes the form of a directed acyclic graph. However, it is important to note that other network structures are possible, such as those that incorporate cyclic or feedback connections, giving rise to recurrent neural networks (*e.g.* Hochreiter and Schmidhuber, 1997) that lend themselves to modeling sequential data, for example for speech [Graves and Schmidhuber, 2005]. While feed-forward and recurrent neural networks are directed graphs, it is also possible to define more generic graphical models with undirected connections between units, such as Deep Boltzman Machines [Salakhutdinov and Hinton, 2009] – though computationally very expensive, Deep Boltzman Machines explicitly model data distributions with stochastic units, and are used for generative models.

For computer vision, one of the earliest neural networks was the Neocognitron [Fukushima, 1988], comprised of alternating layers of filters (where a neuron's response in a spatial position is the response of a filter at that spatial location) and pooling layers. Each filter creates a *feature map* – the response of the filter across spatial locations of the input. The pooling layers aggregate local responses and subsample the feature maps, allowing some translational invariance to the filter responses. This framework was able to learn to perform simple pattern recognition.

A similar representation, convolutional neural networks, was also developed for visual recognition [LeCun et al., 1989; LeCun et al., 1998]. These convolutional models have proved to be highly successful in computer vision. The next section introduces convolutional neural networks. We then review some of the applications of this learning framework to computer vision problems in Section 2.1.2, as well as ways of approximating, reducing the size, and speeding up these models in Section 2.1.3.

2.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [LeCun et al., 1989; LeCun et al., 1998] are obtained by stacking multiple layers of convolutional filter banks on top of each other, followed by a non-linear response function. Each filter bank or *convolutional layer* takes an input which is a feature map $\mathtt{z}_i \in \mathbb{R}^{H \times W \times C}$, with width W and height H. $\mathtt{z}_i(u, v) \in \mathbb{R}^C$ contains C scalar features or *channels* $z_i^c(u, v)$. The output of a convolutional layer is a new feature map $\mathtt{z}_{i+1} \in \mathbb{R}^{H' \times W' \times N}$ such that

$$\mathtt{z}_{i+1}^n = h_i(\mathtt{w}_{in} * \mathtt{z}_i + \mathbf{b}_{in}) \; \forall n \in [1 \dots N], \tag{2.1}$$

where $\mathtt{w}_{in}$ and $\mathbf{b}_{in}$ denote the n-th filter kernel and bias respectively, $\mathtt{z}_{i+1}^n$ is the n-th channel of the output tensor $\mathtt{z}_{i+1}$, $*$ denotes the convolution operator, and h_i is an element-wise non-linear activation function. This non-linear activation function can take any form, but is most often a sigmoid such as $h_i(\mathtt{z}) = \tanh(\mathtt{z})$, or a rectified linear unit (ReLU) $h_i(\mathtt{z}) = \max\{0, \mathtt{z}\}$ where all negative elements of $\mathtt{z}$ are clamped to zero.

Convolutional layers can be intertwined with pooling layers with subsampling. A *pooling layer* takes a local region of a feature map and outputs a scalar response using a pooling function – commonly this pooling function is average pooling, which outputs the average value of the pooling region, or max pooling, which outputs the maximum value of the pooling region. This pooling operation can be evaluated with a pooling region centred on every pixel of the feature map, or can be evaluated on a subset of pixels leading to a subsampled output feature map. For example, a 2×2 max pooling layer with a stride of 2 takes the maximum value of a 2×2 pooling region in a feature map centred at every other pixel, leading to a $2\times$ subsampled output feature map. Pooling layers help build translation invariance in local neighbourhoods while also downsampling the feature maps to reduce dimensionality for the rest of the model.

The forward evaluation process of a CNN starts with $\mathtt{z}_1 = x$, where $\mathtt{x}$ is the input image, and ends by, for example, connecting the last feature map to a logistic regressor in the case of classification. Often the last few layers of a network are *fully connected layers*, where each unit is connected to every element of the input feature maps.

CNNs are a sub-type of neural network, which exploit weight sharing by defining filters whose weights are the same but applied to each spatial location of the input feature map. Besides helping to reduce the number of parameters required by the network, weight sharing with convolutional layers means that a single model of a particular visual object can be learnt which responds to that object regardless of its spatial location in the input image. The response of the visual object is simply translated in the output feature map corresponding to the object's translation in the input image.

Training. Part of the power of convolutional neural networks is in the ability for all the parameters of the network, $\psi = \{\mathtt{w}_{in}, \mathbf{b}_{in}\} \; \forall \; n, i$ to be jointly optimised in the case of supervised learning. This is done by defining a differentiable loss function and using back-propagation [Rumelhart et al., 1986]. For example, the supervised learning error is

$$E(\mathcal{X}, \psi) = \frac{1}{|\mathcal{X}|} \sum_{k=1}^{|\mathcal{X}|} L(f(\mathtt{x}_k; \psi), y_k) \tag{2.2}$$

with a loss function L between the groundtruth label y_k and the output of the CNN $f(\mathtt{x}_k; \psi)$ on the k-th training sample $\mathtt{x}_k$ from the training set $\mathcal{X}$. This can be minimised, $\min_\psi E(\mathcal{X}, \psi)$, by computing the gradients of the loss with respect to the parameters of the network.

The parameters of the model can be jointly optimised to minimise the loss over the training set using Stochastic Gradient Descent (SGD). In a classification task, this means that the classifier (loosely corresponding to the final layer of the CNN) is jointly trained with the feature extractors (the lower layers of the network) without any human engineering of features, however, the distinction between the classifier

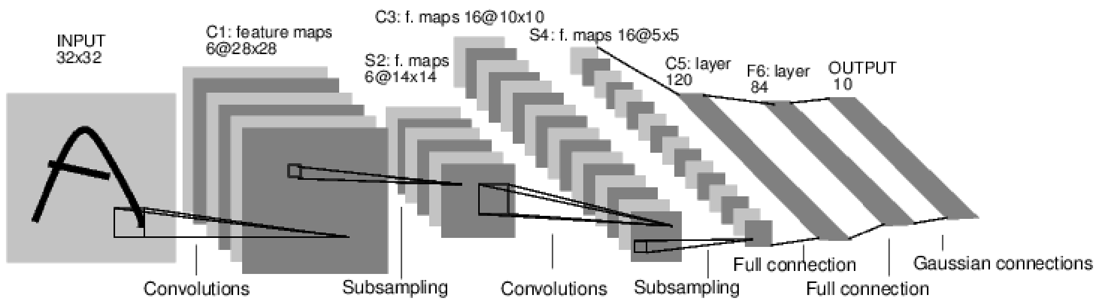


Figure 2.2: The classic LeNet-5 convolutional neural network, taken from LeCun et al., 1998. Alternating convolutional layers (C) and subsampling layers (S) transform the input image, followed finally by a series of fully connected layers (F) to classify the handwritten character image inputs.

and feature extractors are blurred.

Outside of a supervised learning paradigm, CNNs can be also trained in an unsupervised manner, such as by using an autoencoder [Bengio, 2009], which learns layers by minimising the error between the original input and its reconstruction, which is the input after it has been transformed by a number of layers. This is done in a greedy manner, layer by layer, to learn feature extractors based on the data. Unsupervised learning is one way to initialise CNNs for supervised learning, rather than, for example, randomly initialising the weights of a CNN with a Gaussian distribution.

2.1.2 CNNs for Vision

The general structure of convolutional networks provides a powerful framework for building computer vision systems. In fact, one of the earliest successful CNN models was used for character recognition, something useful in traditional text spotting pipelines. The LeNet-5 convolutional neural network (LeCun et al., 1998, shown in Figure 2.2) performs digit recognition, defining the classic, successful architecture of alternating convolutional layers with subsampling layers for feature extraction, and using fully connected layers for the final classification layers. By using subsampling layers, the spatial dimensionality of feature maps are reduced, reducing the computation required by the remainder of the network. This was shown to achieve very good performance on the MNIST digit recognition dataset [LeCun and Cortes, 1998].

However, due to the complexity of natural images, it wasn't until recent years that CNNs have proven to be competitive in natural image vision problems. The advent of large training datasets, such as ImageNet [Deng et al., 2009], and massively-parallel GPU implementations, has made it possible to train much larger networks quickly, and with enough data to avoid overfitting. This was exploited by Krizhevsky et al., 2012 and Ciresan et al., 2012, showing large improvements in visual classification tasks.

Krizhevsky et al., 2012 use an eight layer CNN (the number of layers refers to only the weight layers), incorporating ReLU non-linearities, max-pooling, and contrastive normalisation. This was trained on ImageNet to perform 1000-way object classification. Since the network contains 60 million parameters, the authors use dropout [Hinton et al., 2012] and input image jittering to ensure that overfitting is mitigated.

Since then, a number of other works have used CNNs to advance object classification further [Szegedy et al., 2014; Simonyan and Zisserman, 2015; He et al., 2014]. Simonyan and Zisserman, 2015 use a very deep, 19 layer CNN to perform object classification, and additionally perform localisation – predicting the bounding box of the object within the image – by training a CNN to regress the coordinates of the object's bounding box directly.

There have also been advancements in CNN based object detection (localising and recognising *all instances* of objects within an image) for example with the OverFeat system from Sermanet et al., 2013 and the R-CNN system from Girshick et al., 2014. To perform object detection in Girshick et al., 2014, region proposals are extracted with Selective Search [Uijlings et al., 2013] which are subsequently classified by a CNN. This general object detection framework inspired our own text spotting pipeline in Chapter 5.

Outside of general object recognition, the work by Farabet et al., 2012 has shown the effectiveness of a multi-scale CNN architecture for semantic scene segmentation, directly predicting the class label of each pixel in the input image. Taigman et al., 2014 shows that with a huge amount of training data, CNNs can be used to obtain human level face recognition accuracy. Toshev and Szegedy, 2013; Pfister et al., 2014; Tompson et al., 2014 use CNNs effectively for human pose estimation, and Simonyan and Zisserman, 2014 enable action recognition in videos with a two-stream convolutional neural network. One stream of the CNN acts on the raw image frames, while the other on motion data computed with optical flow, with the output descriptors of each stream undergoing late fusion for classification.

Similar to our own work in Chapter 6, there have been previous works which extend CNNs to incorporate graphical models, such as the early text recognition work of LeCun et al., 1998 to combine character classifier results on image segmentations. Another example is the recent work of Tompson et al., 2014 for human pose estimation, where an MRF-like model over the distribution of spatial locations for each body part is constructed, incorporating a single round of message-passing.

In addition, Section 2.2 includes the review of some further works using CNNs specifically for text spotting.

2.1.3 Fast and Efficient CNNs

As is clear from the wealth of state-of-the-art results in computer vision reported in the works from the previous section, CNNs are a compelling tool for any deployed or commercial vision system. However, the massive number of parameters causes the memory and storage requirements to be large, and the need to perform large numbers of convolutions on multiple feature maps causes the computational complexity and hence the time for inference to be large. This section reviews work related to speeding up and reducing the memory footprint of convolutional neural networks. Some of these methods can be applied at training and test time, while others can only be applied at test time after the model has been fully trained.

2.1.3.1 Hardware Acceleration

To speed up the training and execution of CNNs, one can focus on hardware-specific optimisations. In recent years, a successful approach to this has been to utilise the massively parallel nature of GPUs. GPUs are general purpose computation units which contain thousands of processing cores. Though these are generally clocked slower than CPU cores, they can be very efficiently programmed for parallel execution, allowing large speedups on parallelisable problems. The convolution of multiple filters across feature maps – the main computational component of a convolutional layer in a CNN – is an example of a parallelisable computation problem. This is taken advantage of in `cuda-convnet` [Krizhevsky et al., 2012], where Krizhevsky parallelises the application of convolutions in spatial locations, across filters, and also across data samples. GPU execution of CNNs is also exploited in the `Caffe` framework Jia, 2013 and the `MatConvNet` framework Vedaldi and Lenc, 2014. In these two frameworks, the 3D convolution operation across multiple channels and multiple data samples is converted to matrix-matrix multiplication. While this conversion incurs some computational overhead, as well as increasing the amount of memory required for the routine, the matrix-matrix multiplication can take advantage of low level optimised linear algebra routines, leading to large overall speedups. The work of Chetlur et al., 2014 also addresses GPU based parallelisation with low level optimisations for GPUs. In general, by taking advantage of a parallel GPU based CNN framework, speedups of around 40$\times$ can be gained over CPU execution.

Another promising approach to speeding up convolutional layer execution in CNNs is to compute the convolution in the Fourier domain [Mathieu et al., 2013]. The convolution between two kernels in the real domain becomes the point-wise multiplication in the Fourier domain, *i.e.*

$$\mathtt{w}_{in} * \mathtt{z}_i = \mathcal{F}^{-1}(\mathcal{F}(\mathtt{w}_{in}) \cdot \mathcal{F}(\mathtt{z}_i)) \tag{2.3}$$

where $\mathcal{F}$ denotes the Fourier transform and $\mathcal{F}^{-1}$ the inverse Fourier transform. When the kernel $\mathtt{w}_{in}$ is large and the Fourier transform can be computed quickly, for example using the Fast Fourier Transform algorithm (FFT), speedups over regular convolution can be achieved. In particular, by sharing the FFT computation of the filters between data samples within the same SGD batch, the transformation costs can be minimised. Mathieu et al., 2013 show that this can be implemented efficiently on a GPU for speed gains.

Other hardware specific speedup techniques for CNNs are presented in the work of Vanhoucke et al., 2011. It is shown that specific CPU architectures can be taken advantage of by using SSSE3 and SSE4 fixed-point instructions and intelligently aligning data in memory. Finally, Farabet et al., 2011 engineer a bespoke FPGA implementation of CNNs, allowing real-time inference of a semantic scene segmentation system.

While not specific to hardware, one can take advantage of the parallelisation schemes of convolution frameworks to achieve speedups during inference. As an example, to speed up test-time in a sliding window context for a CNN, Iandola et al., 2014 shows that multi-scale features can be computed efficiently by simply convolving the CNN across a flattened multi-scale pyramid. This allows sharing convolutions, something we also exploit in Chapter 4.

2.1.3.2 Approximating for Speed

Another, often orthogonal, approach to speed up the execution of CNNs is to look at approximating the computations of a CNN. These can be seen as more general speedup methods for CNNs, but can have an associated impact on the accuracy of the CNN due to the approximations made.

Denil et al., 2013 make in important contribution by showing that many of the parameters of a neural network are redundant, and can in fact be predicted by a few base parameters. In particular, the smoothness of the Gabor-like filters produced in the first layer of a CNN can be modelled, and in the best case the full set of parameters can be predicted from just 5% of the parameters, with no loss in accuracy of the full CNN. This is achieved by selecting an appropriate basis set of features and regressing the full set of parameters from the basis. Taking advantage of this redundancy allows memory savings in their method, but this work paves the way for methods that can exploit these redundancies for computational speedups as well.

A simple manner to force approximations of CNNs for computational speedup from Vanhoucke et al., 2011 is to use 8-bit quantisation of the weight parameters, instead of single or double precision floating point representations. This reduces the memory use of the CNN by up to $4\times$, and also allows the use of fast SSSE3 and SSE4 instructions as described in the previous section. This quantisation is applied to all but the first layer of the network, with negligible impact on accuracy.

Taking into account the structure of CNNs, in particular convolutional layers, it is possible to perform low-rank approximations of the filters in the layer.

As opposed to performing full convolution, with full-rank filters, Mamalet and Garcia, 2012 enforce that the filters are rank-1, so the convolution can be simplified into separable convolution, which is less computationally demanding. In addition to the separability of filters, Mamalet and Garcia, 2012 aim at combining the dense convolution layer followed by pooling into a single convolutional layer. This can be achieved by ignoring the non-linear component of the network after convolution is performed. In that case, convolution followed by average pooling with $2\times$ down-sampling can by simplified to a single convolutional layer with a stride of 2. Separable convolution and sub-sampling layers are fused together in the network design, and used for training and testing with some success. However, this technique imposes constraints on the designs of the network.

Denton et al., 2014 also use low rank approximations and clustering of filters to find approximate the filters in a CNN and achieve speedups. The 4D filter tensors (width $\times$ height $\times$ channels $\times$ number of filters) can be reduced in rank using a CP-decomposition based scheme. Denton et al., 2014 perform biclustering, splitting the two non-spatial dimensions of the filter kernel into subgroups, and reduces the effective ranks in the CP-decomposition. Each component is computed in a greedy manner, and found by minimising the reconstruction error with least squares. It should be noted that this process is not applied to the first layer of the network. Similar to our observations in Chapter 7, the filters in this exhibit different properties which do not lend themselves to low rank decomposition. Using these techniques Denton et al., 2014 achieve $1.6\times$ speedup of single convolutional layers (not of the whole network) with a 1% drop in classification accuracy on a CNN trained for ImageNet classification.

An alternative method for computing low rank decompositions of filter banks, not specific to CNNs, is the work of Rigamonti et al., 2013. Rigamonti *et al.* show that multiple image filters can be approximated by a shared set of separable (rank-1) filters, allowing large speedups with minimal loss in accuracy. We build on this approach in our own work in Chapter 7, with the method of Rigamonti et al., 2013 discussed fully there. The recently released work of Lebedev et al., 2014 also builds on this idea and our own work, performing a full CP-decomposition on the 4D filter tensors, showing impressive results.

2.2 Text Spotting

This section now focusses on a review of the previous work related to text spotting. In particular, this literature review will focus on the recognition of text from natural images, as opposed to recognising text from scanned (or fronto-parallel photographed) documents – a much less complex problem.

Since text spotting is usually decomposed into two salient part, detection and recognition, with previous work sometimes constrained to an individual sub-task or the end-to-end text spotting problem as a whole, we focus on reviewing each sub-problem separately in Section 2.2.1 and Section 2.2.2.

2.2.1 Text Detection Methods

Text detection methods tackle the first task of the standard text spotting pipeline [Chen and Yuille, 2004]: producing segmentations or bounding boxes of words in natural scene images. Detecting instances of words in noisy and cluttered images is a highly non-trivial task, and the methods developed to solve this are based on either character regions [Chen et al., 2011; Epshtein et al., 2010; Yi and Tian, 2011; Yin et al., 2013; Neumann and Matas, 2010; Neumann and Matas, 2011; Neumann and Matas, 2012; Neumann and Matas, 2013; Huang et al., 2014] or sliding windows [Quack, 2009a; Anthimopoulos et al., 2011; Posner et al., 2010; Wang et al., 2011; Wang et al., 2012; Chen and Yuille, 2004].

2.2.1.1 Character Region Methods

Character region methods aim at finding connected component regions in the image which are characters, and then grouping character regions into words. The advantage of character region methods for detecting words is that the segmenta-

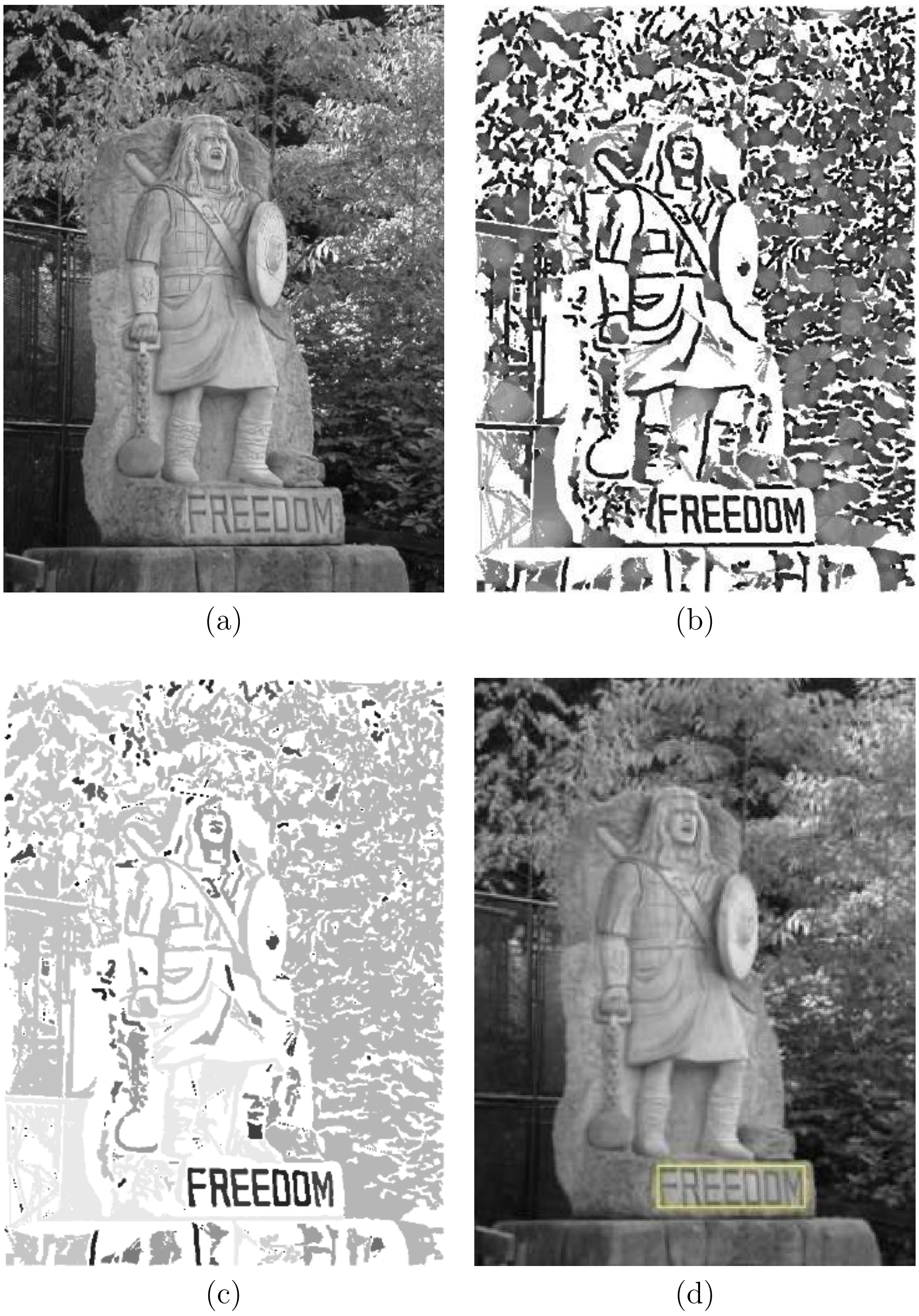


Figure 2.3: An example of the Stroke Width Transform (SWT) [Epshtein et al., 2010] used for text detection of an image (a). The SWT assigns every pixel the value of the width of the stroke it belongs to (b). Text and characters tend to be regions of uniform stroke width, so the variance of each connected component can be computed (c). Regions of low variance are likely to be characters, and can be grouped for word detection (d). Images from Epshtein et al., 2010

tion of characters is predicted as part of the process which can aid the subsequent recognition process.

Epshtein et al., 2010 use the notion of *stroke width* to assign pixels to characters. The Stroke Width Transform (SWT) produces a per-pixel output transformation of the original image, where each pixel is assigned the value of the width of the stroke it is estimated to belong to. Regions of similar stroke width are grouped together, and the variance of each connected component computed. Intuitively, it is observed that characters are composed from regions of constant stroke width, so low variance connected components are taken as character candidates. Nearby character candidates with similar stroke width, sizes, and other heuristic properties, are grouped together to form words. This process is illustrated in Figure 2.3. In more recent work, Neumann and Matas, 2013 revisit the notion of characters represented as strokes, but rather than using the SWT, use gradient filters to detect oriented strokes.

Another method of finding character regions in images is to use Extremal Regions Matas et al., 2002. An Extremal Region (ERs) is a connected component region of a grayscale image with similar intensity values. These regions are found by incrementally thresholding the image which gives rise to a tree of ERs, with the root node corresponding to an ER consisting of the whole image. Maximally Stable Extremal Regions (MSERs) are a subset of ERs which exhibit stability in shape while varying the threshold intensities. ERs and MSERs have the advantage of being fast to compute. The work by Neumann and Matas [Neumann and Matas, 2011; Neumann and Matas, 2010; Neumann and Matas, 2012] exploit this method of generating connected components for character and word detection. In Neumann and Matas, 2010, MSERs are extracted and each one is classified by a binary character/no-character support vector machine (SVM) classifier [Cristianini and Shawe-Taylor, 2000] acting on shape and colour features (Figure 2.4). Text lines are then created by connecting

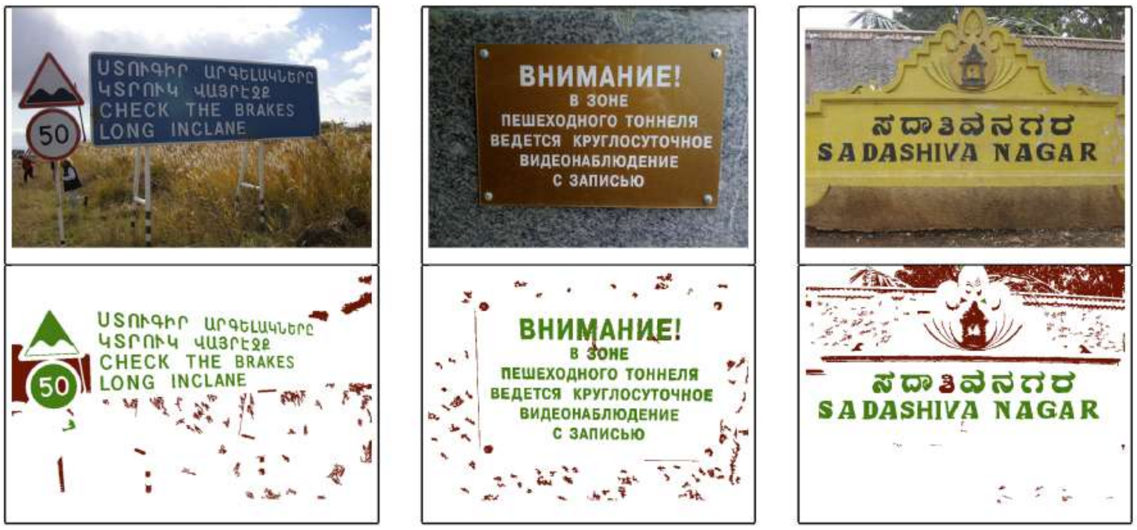


Figure 2.4: Three examples of images containing text of different scripts. Below each image, the MSERs extracted are shown, with green regions having been classified as characters regions, and red regions classified as non-character regions. MSER classification is used as the basis for a number of previous text spotting methods, including that of Neumann and Matas, 2010, where this figure is taken from.

each character MSER to other character MSERs in the graph of nearby detections when it is predicted the two character MSERs belong together, with the prediction driven by another SVM. In Neumann and Matas, 2012, a cascade of classifiers filter out ERs which do not resemble characters, and through pruning, an exhaustive search of the graph connecting all character candidates is performed, grouping characters into text lines (presented in Neumann and Matas, 2012). This method is particularly impressive in being able to achieve real time speeds for localisation and recognition (the recognition stage is described in Section 2.2.2). An open source implementation of this detection algorithm is available[1] which includes an improved grouping method from Gomez and Karatzas, 2013.

MSERs are also used for text detection in the work of Yin et al., 2013; Chen et al., 2011; Huang et al., 2014. Chen et al., 2011 build on vanilla MSERs by using edge-enhanced MSERs which are more robust to blur, and use geometric properties

[1]`http://docs.opencv.org/trunk/modules/objdetect/doc/erfilter.html`

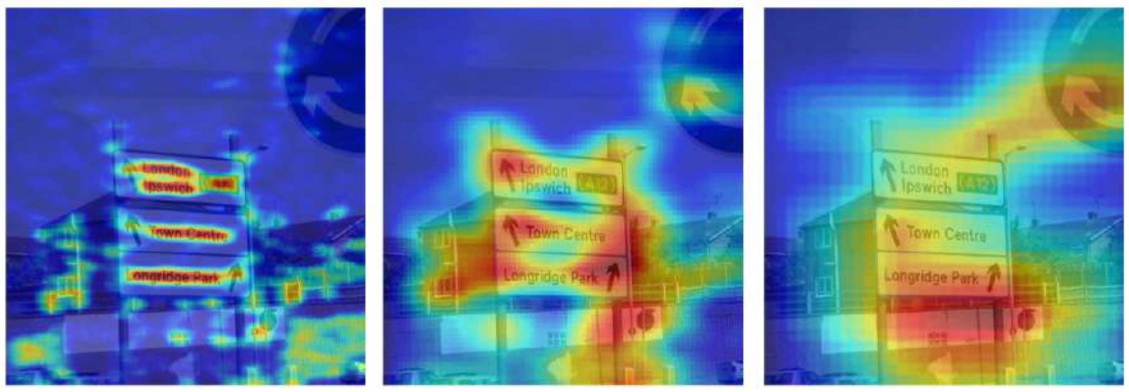


Figure 2.5: The text saliency maps (red indicates a high probability of text) computed with a sliding window random forest classifier acting on local edge features from Anthimopoulos et al., 2011. Each response map, from left to right, is computed at a different scale, with a target text height of 10 pixels, 50 pixels, and 90 pixels respectively. The original image has a resolution of 480 $\times$ 503 pixels. Note how the "correct" text size of 10 pixels allows fairly accurate text detection.

combined with the stroke width computed with the SWT [Epshtein et al., 2010] as an additional cue for filtering out false positive character candidate detections. More recently, Huang et al., 2014 expand on the use of Maximally Stable Extremal Regions by incorporating a strong CNN classifier to efficiently prune the trees of Extremal Regions, also leading to less false-positive detections.

2.2.1.2 Sliding Window Methods

As an alternative strategy for text detection, many previous work uses sliding window methods to find text in an image. Sliding window based methods operate in a manner similar to generic object detection – that is taking a small window of interest within the full image, and classifying whether or not text is contained within that window. This window is then translated across the entire image, and at different scales, to detect text at all possible positions and scales. The output is a pixel wise text/no-text classifier, creating a text saliency map. The text saliency map is then generally thresholded to generate bounding box predictions for text localisation.

Early work by Chen and Yuille, 2004, Quack, 2009a, and Posner et al., 2010 all make use of a similar sliding window strategy for text detection based on boosted classifiers and Haar-like features. Posner et al., 2010 uses a cascade of boosted classifiers to generate a saliency map for each scale, then thresholding this map to obtain bounding boxes of text. Till Quack's text detector [Quack, 2009a] is based on the Viola-Jones method of face detection [Viola and Jones, 2001] – simple rectangular block based features are combined as weak classifiers in an Adaboost framework [Freund and Schapire, 1997] – with the positive sliding window detections being clustered into complete bounding box estimates for text localisation.

Anthimopoulos et al., 2011 create hand-crafted features specifically for text detection. These are dynamically-normalised edge features which generate local binary patterns within the sliding window image patch. The local binary patterns are used for a Random Forest classifier [Breiman, 2001] evaluated in a sliding window manner to produce a text saliency map. Figure 2.5 shows some example text saliency maps generated using our own implementation of this method.

Borrowing a popular detection framework from the general object detection literature, Wang et al., 2011 use a random ferns classifier [Ozuysal et al., 2007] trained on HOG features [Felzenszwalb et al., 2010] in a sliding window scenario to find characters in an image. These are grouped into words using a pictorial structures framework [Felzenszwalb and Huttenlocher, 2005], with the grouping constrained by a small fixed lexicon. Interestingly, Wang et al., 2011 demonstrate the ability to train on synthetically generated character data, a theme we also explore in Chapter 5 and Chapter 6.

Wang et al., 2012 was the first work to make use of the discriminative power of CNNs for text detection. The authors define a text/no-text classification CNN, which is applied in a sliding window manner convolutionally to generate a text saliency map. The CNNs are trained on a synthetic character dataset (described in Section 3.1.6), but only after being pre-trained in an unsupervised manner [Coates et al., 2011]. Word detections are generated by first finding text lines (where a text line could contain multiple words) through non-maximal suppression across rows in the text saliency map, and then splitting the text lines into words by performing non-maximal suppression along the columns of each text line individually. The work by Wang et al., 2012 was a key inspiration for the text detection phase in Chapter 4.

All text detection methods reviewed employ the notion of detecting text by first detecting characters, and subsequently grouping these into words. In our own text detection work in Chapter 5, we depart from this notion and model words explicitly, removing some of the ambiguity faced trying to detect single characters, and avoiding the inaccuracy that is packaged with having to perform a character detection grouping technique.

2.2.2 Text Recognition Methods

Once text lines or words have been detected, the next stage for a text spotting system is to recognise the text. The methods reviewed in this section are focussed purely on this text recognition stage. Since the input to the text recognition stage is generally a cropped word image, all references to images in this section refer to word images.

While it should be noted that there are many previous works focussing on handwriting or historical document recognition [Graves et al., 2009; Fischer et al., 2010; Rath and Manmatha, 2007; Frinken et al., 2012; Manmatha et al., 1996], these methods don't generalise in function to generic scene text due to the highly variable foreground and background textures that are not present with documents.

For scene text recognition, methods can be split into two groups – character based recognition [Yao et al., 2014; Bissacco et al., 2013a; Alsharif and Pineau, 2014; Wang et al., 2011; Wang et al., 2012; Quack, 2009b; Posner et al., 2010; Weinman et al., 2014] and whole word based recognition [Goel et al., 2013; Rodriguez-Serrano et al., 2013; Novikova et al., 2012; Mishra et al., 2012; Almazán et al., 2014; Gordo, 2014].

2.2.2.1 Character Based Recognition

Character based recognition relies on an individual character classifier to perform per-character recognition which is integrated across the word image to generate the full word recognition. This therefore requires some level of character detection or segmentation within the word image, which is often provided by the detection stage of a text spotting pipeline.

As discussed in the previous section, Neumann and Matas, 2011; Neumann and Matas, 2010; Neumann and Matas, 2012 construct a graph of character MSERs to represent a word. To recognise the characters in this graph, these methods use the gradients of the size-normalised character segmentations (the MSER associated with a character) as features to an SVM classifier to classify character regions. This is followed by a typographical model for disambiguating uppercase and lowercase characters, and a pairwise letter-based language model to choose the most probable recognised word estimate. In their more recent work, Neumann and Matas, 2013 use nearest neighbour search in feature space to a database of synthetic characters to perform character recognition, but crucially keep a shortlist of multiple possible recognition results for each character to mitigate uncertainty when performing word recognition. The final word is then the maximum scoring sequence of characters from the shortlists given a 2-gram language model and a graphical model.

The method in Wang et al., 2011 also uses a graphical model, with unary terms from character classifiers acting on HOG features and pairwise terms incorporating spatial layout and scale similarity between neighbouring characters. This is done in the context of a lexicon, so each lexicon word is scored for each pictorial structure, with the position of the characters optimised efficiently with dynamic programming. The highest scoring lexicon word is then taken as the final recognition result.

The work by Yao et al., 2014 also uses HOG features for character recognition, but additionally define a set of mid-level features, strokelets, which are learnt by clustering sub-patches of characters. Characters are detected with Hough voting, with the characters classified by a random forest classifier acting on strokelet and HOG features combined.

The works of Wang et al., 2012; Bissacco et al., 2013a; Alsharif and Pineau, 2014 all use CNNs as character classifiers. Wang et al., 2012 use unsupervised pre-training (as with their text/no-text classifier described in Section 2.2.1) to create a multi-class CNN classifier to perform case-sensitive character recognition. This is then slid across a word image, producing a $62 \times W$ response, where W is the width of the word image and 62 is the number of characters (0-9, a-z, A-Z). Non maximal-suppression finds the columns of the response most likely to contain a character. As in Wang et al., 2011, recognition is performed in the context of a fixed lexicon by scoring each lexicon word, optimally fitting each word to the non-maximally suppressed columns of the response map.

To remove the need to perform expensive sliding window evaluation of character classifiers across the whole word image, Bissacco et al., 2013a and Alsharif and Pineau, 2014 over-segment the word image into potential character regions, either through unsupervised binarization techniques or with a fast supervised classifier trained to detect gaps between characters. Over-segmentation produces a set of potential character segments which are individually classified, and the final word is recognised by selecting a path through the over-segmentation graph to maximise a score (see Figure 2.6). Alsharif and Pineau, 2014 use a complicated combination of segmentation-correction and character recognition CNNs together with an HMM, constrained by a fixed lexicon, to generate these scores. The PhotoOCR system [Bissacco et al., 2013a] uses a neural network classifier acting on the HOG features of the segments as scores to find the best combination of segments using beam

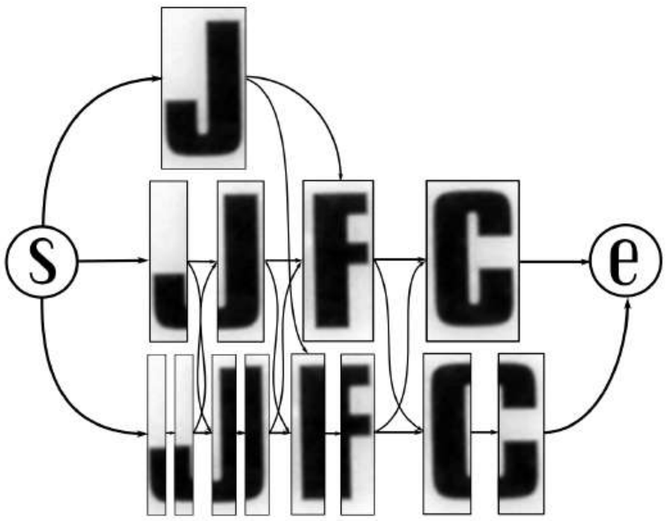


Figure 2.6: An example of the graph of potential character detections produced by an over-segmentation method, such as in Bissacco et al., 2013a; Alsharif and Pineau, 2014. Each segmentation undergoes character classification, and the full word is found by finding the path from start (S) to end (e) which maximises some score. Image taken from Alsharif and Pineau, 2014.

search. The beam search incorporates a strong static N-gram language model, and the final beam search proposals are re-ranked with a further language model and shape model.

We make our own contribution to character based recognition with the recognition system in Chapter 4.

2.2.2.2 Word Based Recognition

As an alternative approach to text recognition, other methods use whole word based recognition methods. Rather than requiring character detection and classification, features from across the entire word image are integrated or pooled to perform word classification.

The works of Mishra et al., 2012 and Novikova et al., 2012 still rely on explicitly trained character classifiers, but pool evidence together from across the word. Novikova et al., 2012 combines both visual and lexicon consistency into a single probabilistic model. The solution is obtained by estimating the maximum a posteriori solution of the joint posterior of the model using weighted finite-state transducers, resulting in accurate recognition of scene text using a full English language lexicon.

Goel et al., 2013 take a whole word image approach by comparing the word image to many simple black-and-white synthetic font-renderings of lexicon words. Histogram of gradient features are computed from vertical strips of the image to be recognised at different horizontal locations, and a weighted Dynamic Time Warping algorithm is used to match these features to the database of synthetic word images.

A popular way to exploit a full word image approach to text recognition is to define a joint embedding space between images and dictionary/lexicon words [Rodriguez-Serrano et al., 2013; Almazán et al., 2014; Gordo, 2014]. A mapping from words (labels) to a common embedding space is learnt together with a mapping from word images to the same space. Recognition can be then posed as a nearest neighbour search by projecting the test word image into the embedding space and finding the nearest word label in that space.

Rodriguez-Serrano et al., 2013 use aggregated Fisher Vector [Perronnin et al., 2010] encodings of patches from the image as features, and a spatial pyramid of characters representation for the word labels. A linear projection between these two encodings is learnt in a Structured SVM framework, ensuring groundtruth pairs of images and word labels are closer than the images with incorrect labels. This creates an effective joint embedding space to allow word recognition.

Almazán et al., 2014 further explore the notion of word embeddings. Differently to Rodriguez-Serrano et al., 2013, word labels are encoded with a pyramidal histogram of characters representation and projected into the common embedding space, and although images are also encoded with Fisher Vectors, an attribute model is learnt to represent the Fisher Vector encoded images before then being transformed into the common embedding space with word labels. This framework is extended in Gordo, 2014 where Gordo makes explicit use of character level training data to learn mid-level features to represent word images. This results in performance on par with Bissacco et al., 2013a but using only a small fraction of the amount of training data.

While not performing full scene text recognition, Goodfellow et al., 2013b had great success using a CNN with multiple position-sensitive character classifier outputs to perform street number recognition. This model was extended to CAPTCHA sequences up to 8 characters long where they demonstrated impressive performance using synthetic training data for a synthetic problem (where the generative model is known). In contrast, we show that synthetic training data can be used for a real-world data problem (where the generative model is unknown) in the work of this thesis.

Our methods for text recognition in Chapter 5 and Chapter 6 also follow a whole word image approach. Similarly to Goodfellow et al., 2013b, we take the word image as input to a deep CNN, however we employ a dictionary classification model in Chapter 5, as well as combining a character prediction and an N-gram prediction model with a conditional random field in Chapter 6.

Chapter 3

Data

In this chapter we describe the datasets used throughout this thesis. Since the focus of this thesis is solely on text spotting, the same datasets for training and evaluation are used in multiple chapters, and so we introduce and describe them here.

There are a number of pre-existing publicly available datasets for text detection and text recognition, which we describe in Section 3.1.

However, we also make our own contributions to the data in this field. The text spotting methods in our work are largely based on supervised machine learning, requiring labelled training datasets. Since the quality of models learnt generally improves as the volume of training data is increased, we collect new datasets for training and testing. To avoid an expensive human labelling process, we present two methods for *automatically generating labelled data* with minimal human interaction.

Section 3.2 describes two methods for mining existing data repositories with weak supervision to generate word and character level annotations. In Section 3.3 we show that we can generate highly realistic *synthetic* data, which can be used in place of real-world data for training and testing word recognition models.

Word Recognition Datasets			
Label	Description	Lexicon	# images
Synth90k	Our synthetic test dataset with words from 90kDict.	90k	900k
SynthRand	Our synthetic test dataset with random text.	-	900k
IC03	ICDAR 2003 [Lucas et al., 2003] test dataset.	-	860
IC03-50	ICDAR 2003 [Lucas et al., 2003] test dataset with fixed lexicon.	50	860
IC03-Full	ICDAR 2003 [Lucas et al., 2003] test dataset with fixed lexicon.	563	860
SVT	SVT [Wang et al., 2010] test dataset.	-	647
SVT-50	SVT [Wang et al., 2010] test dataset with fixed lexicon.	50	647
IC13	ICDAR 2013 [Karatzas et al., 2013] test dataset.	-	1015
IIIT5k-50	IIIT5k [Mishra et al., 2012] test dataset with fixed lexicon.	50	3000
IIIT5k-1k	IIIT5k [Mishra et al., 2012] test dataset with fixed lexicon.	1000	3000

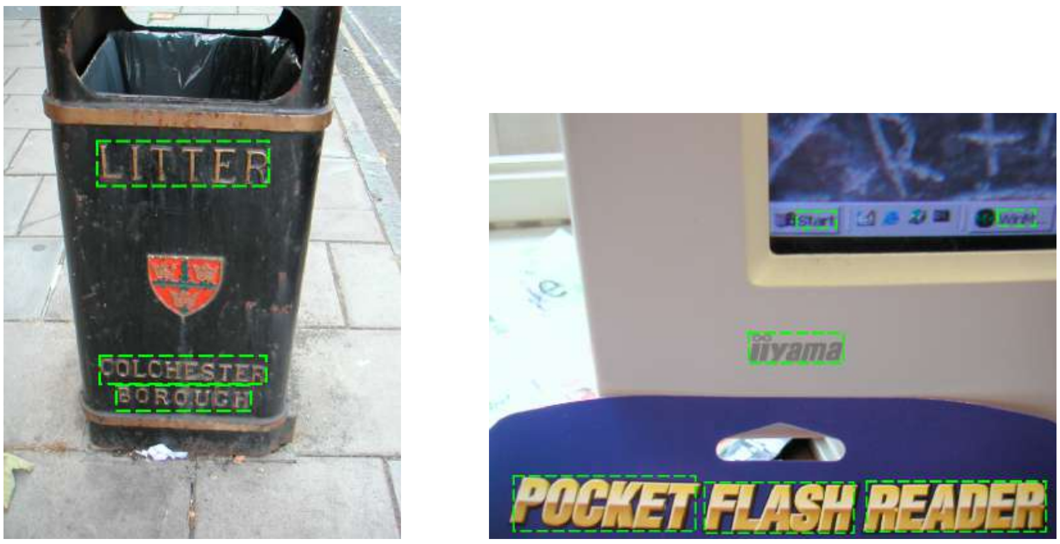
Table 3.1: A description of the various *word recognition* datasets evaluated on. The images are cropped word images, with the task to recognise the word depicted. A per-image lexicon provides a shortlist of words to simplify the search space.

Text Spotting Datasets			
Label	Description	Lexicon	# images
IC03	ICDAR 2003 [Lucas et al., 2003] test dataset.	-	251
IC03-50	ICDAR 2003 [Lucas et al., 2003] test dataset with fixed lexicon.	50	251
IC03-Full	ICDAR 2003 [Lucas et al., 2003] test dataset with fixed lexicon.	860	251
SVT	SVT [Wang et al., 2010] test dataset.	-	249
SVT-50	SVT [Wang et al., 2010] test dataset with fixed lexicon.	50	249
IC11	ICDAR 2011 [Shahab et al., 2011] test dataset.	-	255
IC13	ICDAR 2013 [Karatzas et al., 2013] test dataset.	-	233

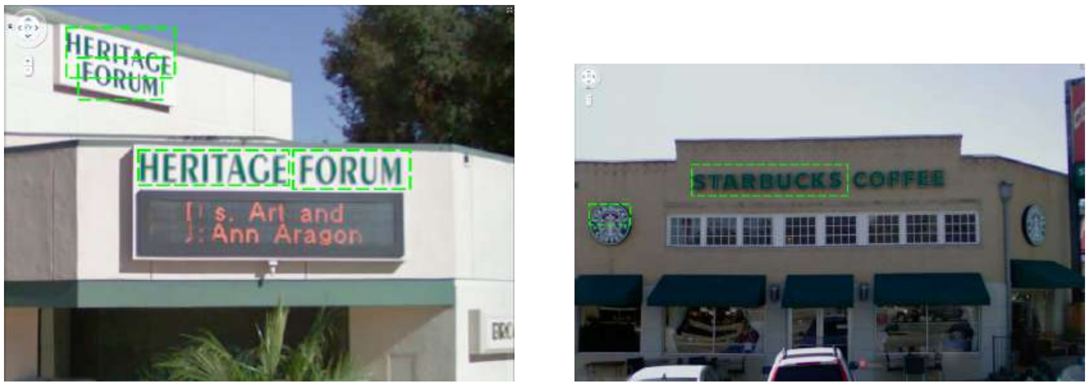
Table 3.2: A description of the various *text spotting* datasets evaluated on. The images are of full scenes, with the task to localise and recognise the text contained within the scenes. A per-image lexicon provides a shortlist of words to simplify the search space.

3.1 Public Datasets

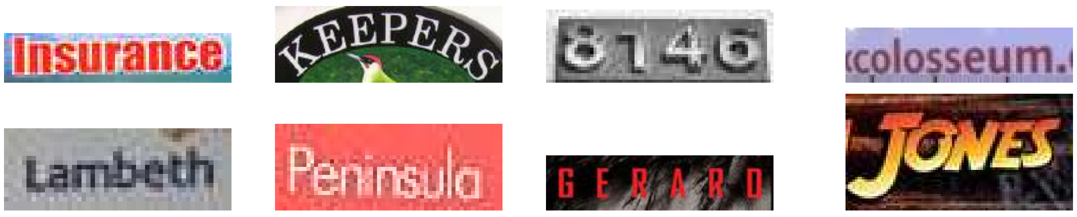
This section describes the various publicly available text spotting and recognition datasets contributed from other authors. For simplicity, the datasets used for *evaluation* of different text spotting tasks are summarised in Table 3.1, Table 3.2, and Table 3.3. Examples from the datasets are shown in Figure 3.1 and Figure 3.2.



(a)



(b)



(c)

Figure 3.1: Some text spotting data examples from (a) IC03 and (b) SVT, showing the word level bounding box annotations (green dashed). (c) Some examples from the IIIT5k word recognition test dataset.

Text Retrieval Datasets			
Label	Description	# queries	# images
IC11	ICDAR 2011 [Shahab et al., 2011] test dataset.	538	255
SVT	SVT [Wang et al., 2010] test dataset.	427	249
STR	IIIT STR [Mishra et al., 2013] text retrieval dataset.	50	10k
Sports	IIIT Sports-10k [Mishra et al., 2013] text retrieval dataset.	10	10k

Table 3.3: A description of the various *text retrieval* datasets evaluated on. The task is to retrieve all the images that contain the query word text.

3.1.1 ICDAR datasets

There are a number of datasets introduced by the International Conference of Document Analysis and Recognition (ICDAR), labelled by year.

All the datasets listed below are scene text datasets, containing high resolution images (average size of 940×770) of scenes which contain a variable amount of text within them. The images are taken with a range of cameras around urban areas, with varying levels of annotation. Many of the images are shared between the different years of ICDAR datasets, including across training and test splits, so care must be taken when training on one year's training dataset to evaluate on the test set of another year to avoid training on test data.

ICDAR 2003 (IC03) [Lucas et al., 2003]. Consists of 181 training and 251 test images with word level and character level bounding boxes as well as their case-sensitive transcriptions. Wang et al., 2011 additionally define per-image lexicons to constrain the recognition output for test images. **IC03-50** denotes the IC03 dataset with a per-image small lexicon of approximately 50 words, and **IC03-Full** denotes the IC03 dataset with a lexicon shared across all images of the 563 test words. When used for evaluation, a lexicon constrained dataset (*e.g.* IC03-50 or IC03-Full) reduces the recognition problem to choosing the correct groundtruth word from the shortlist defined by the lexicon, whereas in the absence of a lexicon (*e.g.* IC03) there is no shortlist to choose from.

ICDAR 2005 (IC05) [Lucas, 2005]. Consists of 1001 training and 489 test images with word and character level bounding boxes and case-sensitive labels.

ICDAR 2011 (IC11) [Shahab et al., 2011]. Consists of 229 training and 255 test images with word and character level bounding boxes and case-sensitive labels.

ICDAR 2013 (IC13) [Karatzas et al., 2013]. Consists of 229 training and 233 test images with word and character level bounding boxes and case-sensitive labels.

3.1.2 Street View Text dataset

The Street View Text dataset (**SVT**) [Wang et al., 2011] consists of 349 high resolution images (average size 1260×860) downloaded from Google StreetView of road-side scenes. The training set contains 100 images and the test set 249 images. Only word level bounding boxes are provided with case-insensitive labels. It is important to note that there are many words that have not been annotated. This is a challenging dataset with a lot of noise and since much of the text in the images is attributed to shop signs, there is a wide range of fonts and graphic styles. Per-image 50 word lexicons are also provided (**SVT-50**).

3.1.3 IIIT datasets

A number of scene text datasets have come from the International Institute of Information Technology (IIIT). The images for these datasets are collected with Google image search.

IIIT 5k Word (IIIT5k) [Mishra et al., 2012]. This is a word recognition dataset, comprising of 5k images, 2k training and 3k test images. Each image is a cropped word image of scene text with case-insensitive labels, so this dataset is used purely for cropped word recognition (not detection). Per-image 50 word and 1k word lexicons are also provided: **IIIT5k-50** and **IIIT5k-1k** respectively.

IIIT Scene Text Retrieval (STR) [Mishra et al., 2013]. This is a text based image retrieval dataset. There are 50 query words, with each word having an associated list of 10-50 images that contain the query word. There are also a large number of distractor images with no text downloaded from Flickr. In total there are 10k images, mainly of urban and roadside scenes. No localisation annotation (word or character bounding boxes) is provided, so this dataset can only be used for retrieval evaluation.

IIIT Sports-10k (Sports) [Mishra et al., 2013]. Sports is another text based image retrieval dataset constructed from frames of sports video footage. The images are low resolution (480×360 and 640×352) and often noisy or blurred due to camera movement, with text generally located on advertisements and signboards, making this a challenging retrieval task. 10 query words are provided with 10k total images, without any localisation annotations.

3.1.4 KAIST dataset

The **KAIST** dataset [Lee et al., 2010] consists of 3000 images of indoor and outdoor scenes containing text, with both a mixture of photos from a high resolution (1600×1200) digital camera and a lower resolution (640×480) mobile phone camera. Word and character bounding boxes are provided as well as segmentation maps of characters, and the words are a mixture of English and Korean. However, not all instances of text are labelled.

3.1.5 Oxford Cornmarket Scene Text dataset

This dataset from Posner et al., 2010 provides very high resolution images (average size 3870×2590) of busy street scenes, with case-insensitive, word level bounding box annotations.

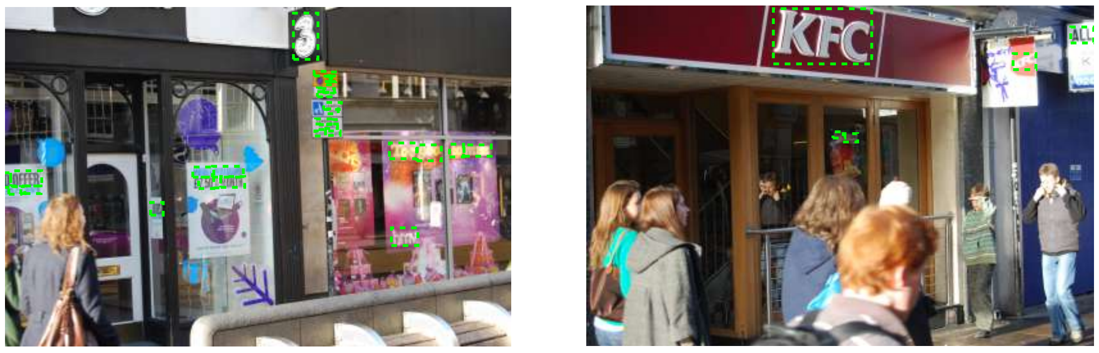
3.1.6 Character Datasets

Character image datasets can be generated by using the character level annotations of the previously described datasets (where provided). There are also the **Chars74k** dataset [Campos et al., 2009] and the **StanfordSynth** dataset [Wang et al., 2012]. Both datasets contain small single-character images of 62 characters (0-9, a-z, A-Z). Chars74k comprises of a set of characters from natural images, as well as a set of synthetically generated characters. The StanfordSynth characters are all synthetically generated, but are very representative of natural scene characters due to the large range of fonts, colours and blending with natural images used to generate the character samples, whereas the Chars74k synthetic characters are not as they are simple black on white renderings of fonts.

3.2 Automatically Annotating Data

To help increase the pool of available training data, in this section we contribute two methods for automatically generating annotations from existing data. The first is to automatically mine photo sharing websites to acquire word and character level annotated data utilising a weak baseline system (Section 3.2.1). The second is to use a word-fitting algorithm to generate character level annotations when only word level annotations exist (Section 3.2.2).

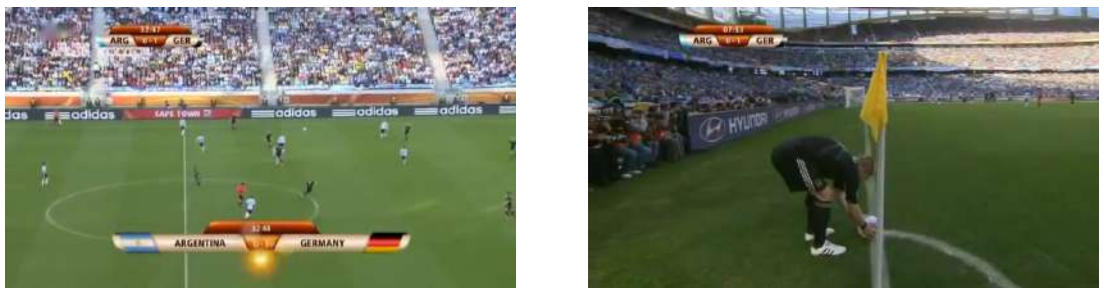
The notion of automatically annotating data for training has previously been exploited successfully in Bissacco et al., 2013a at Google. The authors make use of the data produced by some of the company's mobile apps, where users take photos of text for search or translation and the text spotting results from their existing system is stored. It is hypothesised that much of the text photographed exists somewhere on the internet in digital form. Therefore, the existing, often partially incorrect, transcriptions of the app photos are searched for across the internet, and aligned to



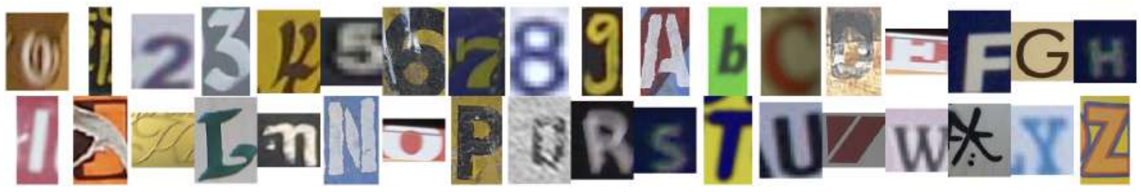
(a)

ADIDAS

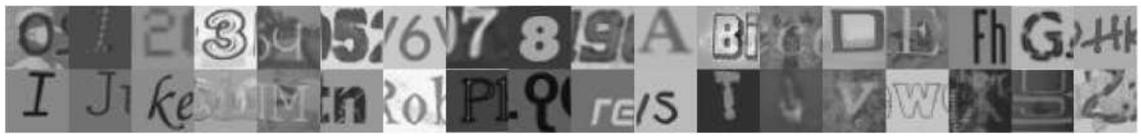
HYUNDAI



(b)



(c)



(d)

Figure 3.2: (a) Some text spotting data examples from Oxford Cornmaket showing the word level bounding box annotations (green dashed). (b) Some example images from Sports showing the associated query word above the image. Some examples from (c) the Chars74k and (d) the StanfordSynth character datasets.

the digital versions found. This allows the incorrect character labels predicted by the existing system to be corrected, producing groundtruth labels with the character bounding boxes coming from the existing text spotting system. This is a form of self-supervision, and allows Bissacco et al., 2013a to automatically generate 4 million extra character data samples. This approach inspires our own data mining methods which we now describe.

3.2.1 Flickr Mining

Photo sharing websites such as Flickr[1] contain a large range of scenes, including those containing text. In particular, the "Typography and Lettering" group on Flickr[2] contains mainly photos or graphics containing text. As the text depicted in the scenes are the focus of the images, the user given titles of the images often include the text in the scene. Capitalising on this weakly supervised information, we develop a system to find title text within the image, automatically generating word and character level bounding box annotations.

We use a weak baseline text spotting system based on the Stroke Width Transform (SWT) [Epshtein et al., 2010], described fully in Section 3.2.1.1, to generate candidate word detections for each image downloaded from Flickr. The weak baseline system outputs word and character level bounding boxes. If a detected word is recognised as the same as any of the image's title text words and there are the same number of characters from the SWT detection phase as word characters, we say that this is an accurate word detection, and use this detection as positive text training data. This harsh constraint – requiring perfect recognition of the title text words – results in very low recall (out of 130000 images, only 15000 words were found), but the precision is greater than 99% (estimated from inspecting samples of the data). This means the precision is high enough for the mined Flickr data to be used as

[1] `http://www.flickr.com`

[2] `http://www.flickr.com/groups/type`

positive training data, but the recall is too low for it to be used for background no-text training data. We will refer to this dataset as **FlickrType**, which contains 6792 images, 14920 words, and 71579 characters. Figure 3.3 shows some examples from this mining process.

This procedure will generate a bias in the data collected towards scene text that can be found with the weak baseline system. However, because the problem is highly constrained to only recognise the text contained with the images' titles, we can ignore the confidence score produced by the weak baseline system, and collect data samples that would normally not have a high enough score to be recognised in a purely unconstrained scenario. Therefore the data generated by this method can still be very useful for training text spotting models.

3.2.1.1 Weak Baseline Text Spotting System

This section describes a weak baseline end-to-end text spotting system based on the Stroke Width Transform (SWT) Epshtein et al., 2010.

First, the SWT word detection algorithm is run as described in Epshtein et al., 2010. The SWT labels each pixel with a value of the width of the stroke it is estimated to belong to. Regions of similar stroke width are combined into connected components and are character candidates. Using simple heuristics [Epshtein et al., 2010], these character candidates are grouped together to form words, generating word bounding boxes, character bounding boxes, as well as rough character segmentations. This process is run with multiple sets of parameters, as the SWT is very sensitive to changes in them, producing a large number of word detection candidates. A random forest classifier based on Anthimopoulos et al., 2011 is then used to produce a text saliency map for each candidate bounding box, rejecting false positive SWT word detections. For each remaining word detection candidate, the rough SWT character segmentations are used to generate a colour model to fully segment characters using Graph Cut [Boykov and Jolly, 2001], after which the word detections are filtered based on profile features described in Dutta et al., 2012. Finally, the segmented word detection is fed in to an off-the-shelf OCR package, Tesseract 3.02[3], for case-insensitive character recognition.

When tested on the standard benchmarks, this baseline system achieves an unconstrained end-to-end word recognition F-measure of 0.50 on ICDAR 2003 (Neumann and Matas, 2011 get 0.41, higher is better) and 0.42 on ICDAR 2011 (Neumann and Matas, 2013 get 0.45).

This weak baseline system is used only for the Flickr mining method described before.

3.2.2 Automatic Character Annotation

Some existing text spotting datasets contain word level annotations but don't contain character level annotations, meaning the data cannot be used to train character classifiers. However, we can use a word recognition system which models characters to automatically generate *character* bounding box annotations for datasets which only have *word* level bounding box annotations.

For each cropped word image, we perform the optimal fitting of the groundtruth text to the character map using the method described in Section 4.2.2. This places inter-character breakpoints with implied character centers, which can be used as rough character bounding boxes. We do this for the SVT and Oxford Cornmarket datasets which generates 22k character annotations from those datasets. These character annotations can of course be used to crop the word images and create character classifier training and test data.

3.3 Synthetic Data Generation

While there are some publicly available datasets, and we can also mine Flickr for more data as in the previous section, the number of full word image samples is only in the thousands, and the vocabulary is very limited. To push past this limit imposed due to lack of public data, we propose a scene text rendering algorithm, which allows millions of realistic word image samples to be generated without any human cost. This section described our proposed synthetic data engine which allows us to generate unlimited amounts of data to train word recognition models.

Following the success of some synthetic character datasets described in Section 3.1 [Campos et al., 2009; Wang et al., 2012], we create a synthetic word data generator, capable of emulating the distribution of scene text images. This is a reasonable goal, considering that much of the text found in natural scenes is restricted to a limited set of computer-generated fonts, and only the physical rendering process (*e.g.* printing, painting) and the imaging process (*e.g.* camera, viewpoint, illumination, clutter) are not controlled by a computer algorithm.

Figure 3.4 illustrates the generative process and some resulting synthetic data samples. These samples are composed of three separate image-layers – a background image-layer, foreground image-layer, and optional border/shadow image-layer – which are in the form of an image with an alpha channel. The synthetic data generation process is as follows:

1. **Font rendering** – a font is randomly selected from a catalogue of over 1400 fonts downloaded from Google Fonts. The kerning, weight, underline, and other properties are varied randomly from arbitrarily defined probability distributions. The word is rendered on to the foreground image-layer's alpha channel with either a horizontal bottom text line or following a random curve.

[3] `https://code.google.com/p/tesseract-ocr`

2. **Border/shadow rendering** – an inset border, outset border, or shadow with a random width may be rendered from the foreground.

3. **Base colouring** – each of the three image-layers are filled with a different uniform colour sampled from clusters over natural images. The clusters are formed by k-means clustering the RGB components of each word image of the ICDAR training datasets into three clusters.

4. **Projective distortion** – the foreground and border/shadow image-layers are distorted with a random, full projective transformation, simulating the 3D world. The random projections are sampled from a mixture of normal distributions with a mean of the identity projection.

5. **Natural data blending** – each of the image-layers are blended with a randomly-sampled crop of an image from the training datasets of ICDAR 2003 and SVT. The amount of blend and alpha blend mode (*e.g.* normal, add, multiply, burn, max, *etc.*) is dictated by a random process, and this creates an eclectic range of textures and compositions. The three image-layers are also blended together in a random manner, to give a single output image.

6. **Noise** – Elastic distortion similar to Simard et al., 2003, Gaussian noise, blur, resampling noise, and JPEG compression artefacts are introduced to the image.

This process produces a wide range of synthetic data samples, being drawn from a multitude of random distributions, mimicking real-world samples of scene text images. The exact details of the process can be found in the source code[4].

Using this process we generate two datasets which we use in this thesis:

[4]`https://bitbucket.org/jaderberg/text-renderer/src/42cf691414a7d6f8c2d303673daac70a813871a2/word_renderer.py`

Synth90k. This synthetically generated dataset consists of 9 million word images, with equal numbers of word samples from a 90k word dictionary. We use 900k of these images for a testing dataset, 900k for validation, and the remaining for training. The 90k dictionary consists of the English dictionary from Hunspell[5], a popular open source spell checking system. This dictionary consists of 50k root words, and we expand this to include all the prefixes and suffixes possible, as well as adding in the test dataset words from the ICDAR, SVT and IIIT datasets – 90k words in total. We will refer to the dictionary as **90kDict**. This dataset is publicly available at `http://www.robots.ox.ac.uk/~vgg/data/text/`.

SynthRand. This is another synthetic dataset of word images, with a training set of 8 million training images and a test set of 900k images. The words generated are completely random strings of characters, with characters uniformly sampled and with variable lengths up to ten characters long. In this corpus there are very few word repetitions (in addition to the random rendering variations).

These two datasets, Synth90k and SynthRand, can be used for training word recognition models. There is a wide range of difficulty in this dataset, from perfectly readable text to almost impossible to read samples. As we show in Chapter 5 and Chapter 6, this synthetic data can be used in place of real-world data to achieve state-of-the-art performance on real-world test data. In Section 5.6.3.1 we explicitly investigate the effect of each stage of the synthetic data generation process on the accuracy of a text recognition system trained on synthetic data and testing on real-world data.

[5]`http://hunspell.sourceforge.net/`

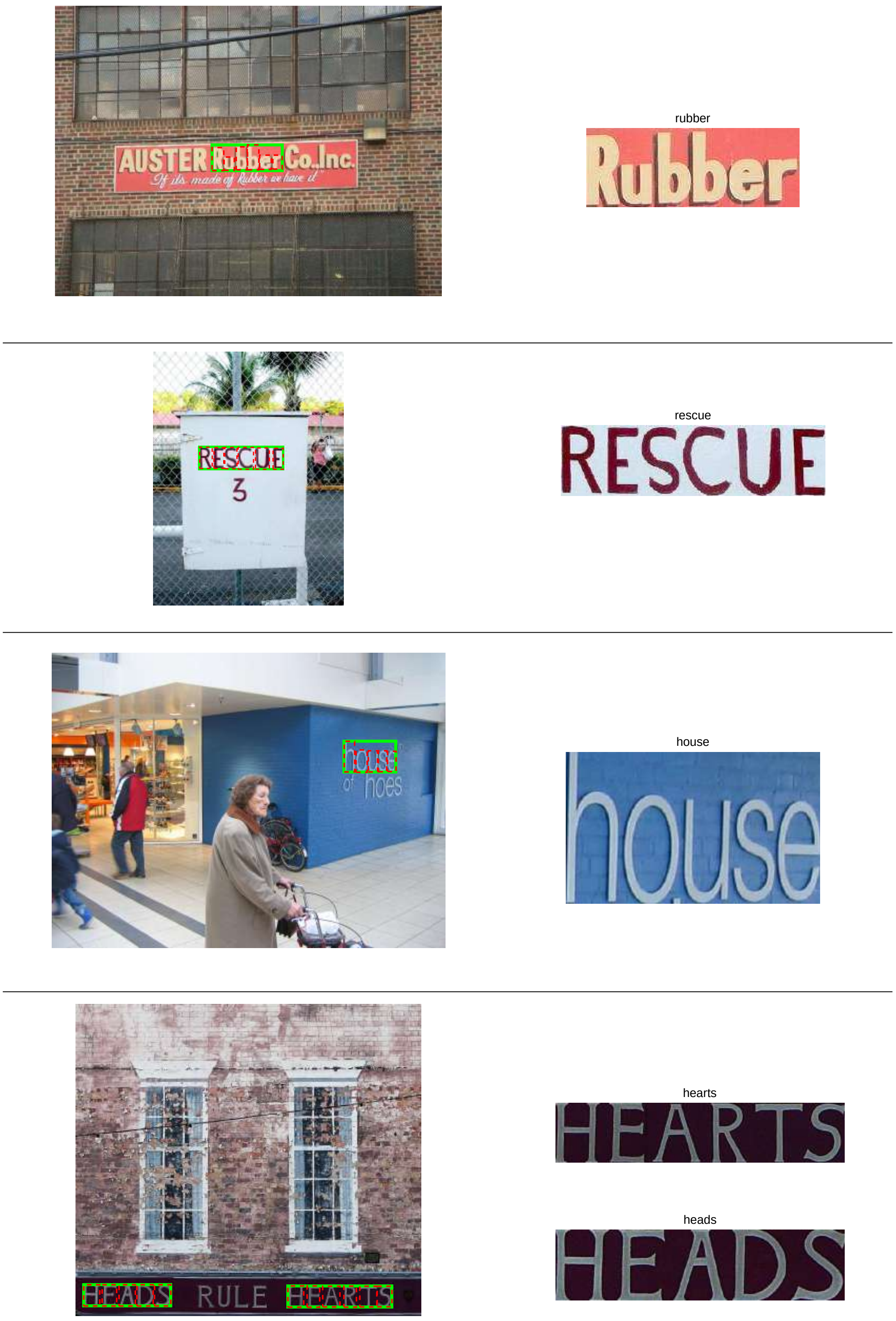
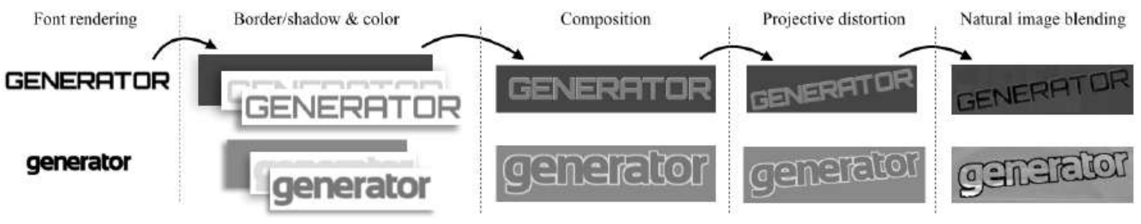
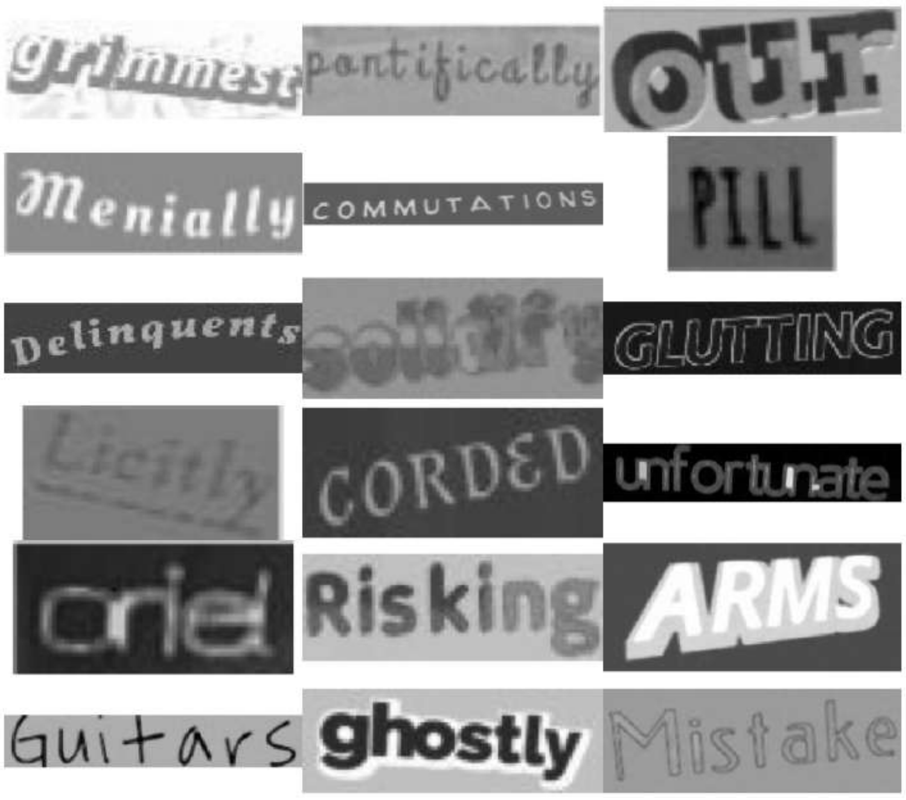


Figure 3.3: Some random examples form the automatically mined and annotated Flickr-Type dataset, showing word level and character level annotations on the original images (left) and the cropped found words with their groundtruth label (right). There are many missed words due to the process being tuned for high precision, and the fact that the only words that can be found are contained within the title of each image.



(a)



(b)

Figure 3.4: (a) The text generation process after font rendering, creating and colouring the image-layers, applying projective distortions, and after image blending. (b) Some randomly sampled data from Synth90k, created by the synthetic text engine.

Chapter 4

Character-centric Text Spotting

In this chapter we present an end-to-end text spotting system, taking a *character-centric* approach. We treat characters as the atomic building blocks for text, modelling them explicitly, and constructing text lines and words by grouping character detections together.

There are many previous works that have taken a character-centric approach successfully, such as [Alsharif and Pineau, 2014; Wang et al., 2012] which are described in Section 2.2. However, in this chapter we show that using a very high quality character classifier can improve over the state of the art for both the word detection and recognition tasks. This is achieved using a convolutional neural network to generate a per-pixel text/no-text saliency map, a case-sensitive and case-insensitive character saliency map, and a bigram saliency map. The text saliency map drives the proposal of word bounding boxes, while the character and bigram saliency maps assist in recognising the word within each bounding box through a combination of soft costs.

The key novelty of this chapter is that we introduce a method to share features which allows us to extend our character classifiers to other tasks such as character detection and bigram classification at a very small extra cost: we first generate a single rich feature set, by training a strongly supervised character classifier, and then use the intermediate hidden layers as features for text detection, character case-sensitive and insensitive classification, and bigram classification. Sharing features has shown success in general object recognition [Torralba et al., 2004], and here we show that this is also the case for text recognition. This procedure makes best use of the available training data: plentiful for character/non-character but less so for the other tasks. All the classifiers are trained on character-level data, making use of the data mining procedures from Section 3.2 to train on more data and learn stronger classifiers.

Another novelty in the context of text detection is to leverage the convolutional structure of the CNN to process the entire image in one go instead of running CNN classifiers on each cropped character proposal [LeCun et al., 1998]. This allows us to generate efficiently, in a single pass, all the features required to detect word bounding boxes, and that we use for recognising words from a fixed lexicon using the Viterbi algorithm. We also make a technical contribution in showing that our CNN architecture using maxout [Goodfellow et al., 2013a] as the non-linear activation function (described in the next section) has superior performance to the more standard rectified linear unit.

In the following sections we first introduce the CNN architecture and training procedure for the character classifiers in Section 4.1. Our end-to-end (image in, text out) text spotting pipeline is described in Section 4.2. Finally, Section 4.3 evaluates the method on a number of standard benchmarks. We show that the performance exceeds the state of the art across multiple measures.

4.1 CNN Feature Learning

The workhorse of a character-centric text-spotting system is the character classifier. The output of this classifier is used to recognise words and, in our system, to detect image regions that contain text. Text-spotting systems appear to be particularly sensitive to the performance of character classification; for example, in Bissacco et al., 2013b increasing the accuracy of the character classifier by 7% led to a 25% increase in word recognition. In this section we therefore concentrate on maximising the performance of this component.

To classify an image patch $\mathtt{x}$ as one of the possible characters (or background), we extract a set of features $\Phi(\mathtt{x}) = (\phi_1(\mathtt{x}), \phi_2(\mathtt{x}), ..., \phi_K(\mathtt{x}))$ and then learn a binary classifier f_c for each character c of the alphabet $\mathcal{C}$. Classifiers are learnt to yield a posterior probability distribution $p(c|\mathtt{x}) = f_c(\Phi(\mathtt{x}))$ over characters and the latter is maximised to recognise the character $\bar{c}$ contained in patch $\mathtt{x}$:

$$\bar{c} = \operatorname*{argmax}_{c \in \mathcal{C}} p(c|\mathtt{x}). \tag{4.1}$$

Traditionally, feature extractors Φ are manually engineered and optimised through a laborious trial-and-error cycle involving adjusting the features and re-learning the classifiers. In this chapter, we propose instead to learn the representation using a CNN, jointly optimising the performance of the features as well as of the classifiers. As noted in the recent literature (Section 2.1.2), a well designed learnable representation of this type can in fact yield substantial performance gains.

CNNs are obtained by stacking multiple layers of features as defined in Section 2.1.1. Usually sigmoid or ReLU functions are used as the non-linear activation functions h_i within the network, however, in our experiments it was found empirically that *maxout* [Goodfellow et al., 2013a] yields marginally superior performance for character classification.

Maxout is an alternative non-linear function that operates in channel space, taking the maximum response across multiple channels at a each spatial position. It can be thought of, in particular when used in the final classification layer, as

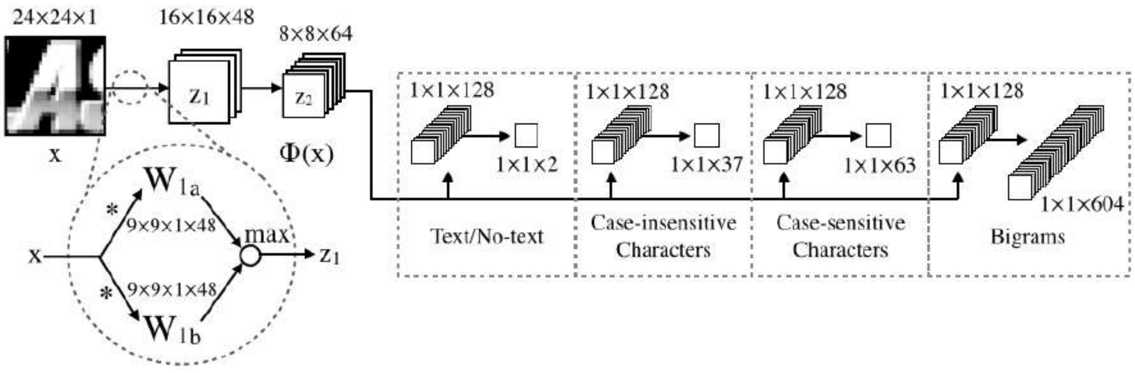


Figure 4.1: The convolutional neural network architecture employed. The method uses four CNNs. These share the first two layers, computing "generic" character features and terminate in layers specialised for text/no-text classification, case-insensitive and case-sensitive character classification, and bigram classification. Each connection between feature maps consists of convolutions with maxout groups.

taking the maximum response over a mixture of linear models, allowing the CNN to easily model multiple modes of the data. The maxout of two feature channels z_i^1 and z_i^2 is simply their point-wise maximum: $h_i(z_i(u,v)) = \max\{z_i^1(u,v), z_i^2(u,v)\}$. More generally, the n'-th maxout operator $h_i^{n'}$ is obtained by selecting a subset $\mathcal{G}_i^{n'} \subset \{1,2,\ldots,N_i\}$ of N_i feature channels and computing the maximum over them: $h_i^{n'}(z_i(u,v)) = \max_{n\in\mathcal{G}_i^{n'}} z_i^n(u,v)$. While different grouping strategies are possible, here groups are formed by taking g consecutive channels of the input map: $\mathcal{G}_i^1 = \{1,2,\ldots,g\}$, $\mathcal{G}_i^2 = \{g+1, g+2, \ldots, 2g\}$ and so on. Hence, given N_i feature channels as input, maxout constructs $N_i' = N_i/g$ new channels.

4.1.1 Training and Implementation Details

This section discusses the details of learning the various character classifiers. Training is divided into two stages. In the first stage, a case-insensitive CNN character classifier is learnt. In the second stage, the feature extractors resulting from the learnt CNN are applied to other classification problems as needed. The products of these two stages are four state-of-the-art CNN classifiers: a character/background classifier, a case-insensitive character classifier, a case-sensitive character classifier, and a bigram classifier.

4.1.1.1 Stage 1: Bootstrapping the case-insensitive classifier.

The case-insensitive classifier uses a four-layer CNN outputting a probability $p(c|\mathtt{x})$ over an alphabet C including all 26 letters, 10 digits, and a noise/background (no-text) class, giving a total of 37 classes (Figure 4.1). The input $\mathtt{z}_1 = \mathtt{x}$ of the CNN are grayscale cropped character images of 24×24 pixels, zero-centred and normalised by subtracting the patch mean and dividing by the standard deviation.

Unlike conventional CNN architectures, no spatial pooling or downsampling is performed. Starting from the first layer, the input image is convolved with 96 filters of size 9×9, resulting in a map of size 16×16 (to avoid boundary effects) and 96 channels. The 96 channels are then pooled with maxout in group of size $g = 2$, resulting in 48 channels. The sequence continues by convolving with 128, 512, 148 filters of side 9, 8, 1 and maxout groups of size $g = 2, 4, 4$, resulting in feature maps with 64, 128, 37 channels and size $8 \times 8, 1 \times 1, 1 \times 1$ respectively. The last 37 channels are fed into a softmax to convert them into character probabilities. In practice we use 48 channels in the final classification layer rather than 37 as the software we use, based on `cuda-convnet` [Krizhevsky et al., 2012], is optimised for multiples of 16 convolutional filters – we do however use the additional 12 classes as extra no-text classes, abstracting this to 37 output classes.

We train using stochastic gradient descent and back-propagation, and also use dropout [Hinton et al., 2012] in all layers except the first convolutional layer to help prevent overfitting. Dropout simply involves randomly zeroing a proportion of the parameters; the proportion we keep for each layer is 1, 0.5, 0.5, 0.5. The training data is augmented by random rotations and noise injection. By omitting any downsampling in our network and ensuring the output for each class is one pixel in size, it is immediate to apply the learnt filters on a full image in a convolutional

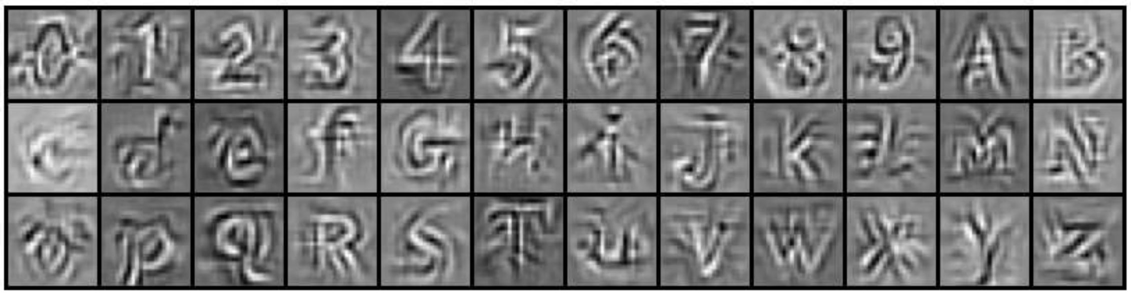


Figure 4.2: Visualisations of each character class learnt from the 37-way case-insensitive character classifier CNN. Each image is synthetically generated by maximising the posterior probability of a particular class. This is implemented by back-propagating the error from a cost layer that aims to maximise the score of that class [Simonyan et al., 2013; Erhan et al., 2009].

manner to obtain a per-pixel output without a loss of resolution, as shown in the second image of Figure 4.3. We can also probe the resulting CNNs to visualise the representations learnt by them. Figure 4.2 illustrates this using the visualisation technique of Simonyan et al., 2013.

4.1.1.2 Stage 2: Learning the other character classifiers.

Training on a large amount of annotated data, and also including a no-text class in our alphabet, means the hidden layers of the network produce feature maps highly adept at discriminating characters, and can be adapted for other classification tasks related to text. We use the outputs of the second convolutional layer as our set of discriminative features, $\Phi(\mathtt{x}) = \mathtt{z}_2$. From these features, we train a 2-way text/no-text classifier[1], a 63-way case-sensitive character classifier, and a bigram classifier, each one using a two-layer CNN acting on $\Phi(\mathtt{x})$, depicted in Figure 4.1. The last two layers of each of these three CNNs result in feature maps with 128-2, 128-63, and 128-604 channels respectively, all resulting from maxout grouping of size $g = 4$. These are all trained with $\Phi(\mathtt{x})$ as input, with dropout of 0.5 on all layers, and fine-tuned

[1]Training a dedicated classifier was found to yield superior performance to using the background class in the 37-way case-sensitive character classifier.

by adaptively reducing the learning rate. The bigram classifier recognises instances of two adjacent characters, *e.g.* Figure 4.5, with a class for each combination of two characters possible.

Discussion. These CNNs could have been learned independently, but found better performance by enforcing feature sharing (see Section 4.3.2). However, sharing the first two layers has two key advantages. First, the low-level features learned from case-insensitive character classification allows *sharing training data* among tasks, reducing overfitting and improving performance in classification tasks with less informative labels (text/no-text classification), or tasks with fewer training examples (case-sensitive character classification, bigram classification). Second, it allows *sharing computations*, significantly increasing the efficiency when using the network for inference on an image.

We also tried a number of different network architectures, varying the number of layers and filters, as well as altering the non-linearity used to ReLU and a sigmoid, implementing the unsupervised pre-training of Wang et al., 2012, and varying the input size and normalisation – the given architecture empirically achieved the lowest classification errors. Also, our design decision to omit down-sampling and pooling stages from the network had no effect on the maximum achieved performance compared to when they are added and the architecture optimised.

4.2 End-to-End Pipeline

This section describes the various stages of the proposed end-to-end text spotting system, making use of the features learnt in Section 4.1. The pipeline starts with a detection phase (Section 4.2.1) that takes a raw image and generates candidate bounding boxes of words, making use of the text/no-text classifier. The words contained within these bounding boxes are then recognised against a fixed lexicon of words (Section 4.2.2), driven by the character classifiers, bigram classifier, and

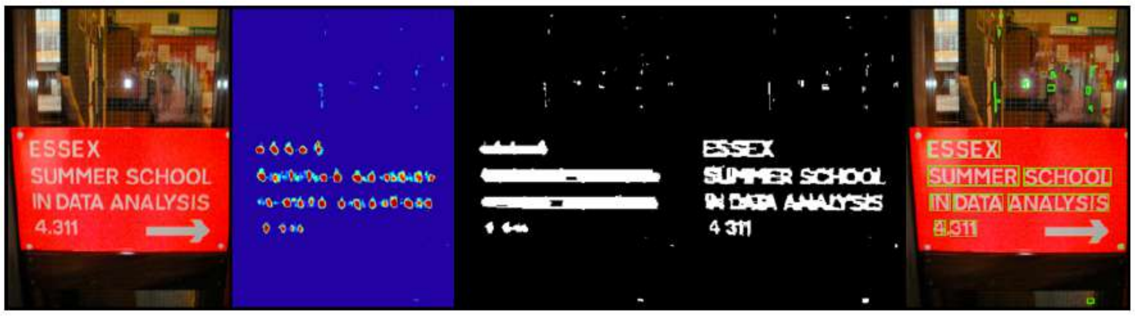


Figure 4.3: The text detection phase for a single scale. From left to right: input image, CNN generated text saliency map using that text/no-text classifier, after the run length smoothing phase, after the word splitting phase, the implied bounding boxes. Subsequently, the bounding boxes will be combined at multiple scales and undergo filtering and non-maximal suppression.

other geometric cues.

4.2.1 Text Detection

The aim of the detection phase is to start from a large, raw pixel input image and generate a set of rectangular bounding boxes, each of which should contain the image of a word. This detection process (Figure 4.3) is tuned for high recall, and generates a set of candidate word bounding boxes.

The process starts by computing a **text saliency map** by evaluating the character/background CNN classifier in a sliding window fashion across the image, which has been appropriately zero-padded so that the resulting text saliency map is the same resolution as the original image. As the CNN is trained to detect text at a single canonical height, this process is repeated for 16 different scales to target text heights between 16 and 260 pixels by resizing the input image.

Given these saliency maps, word bounding boxes are generated independently at each scale in two steps. The first step is to **identify lines of text**. To this end, the probability map is first thresholded to find local regions of high probability. Then these regions are connected in text lines by using the *run length smoothing algorithm*

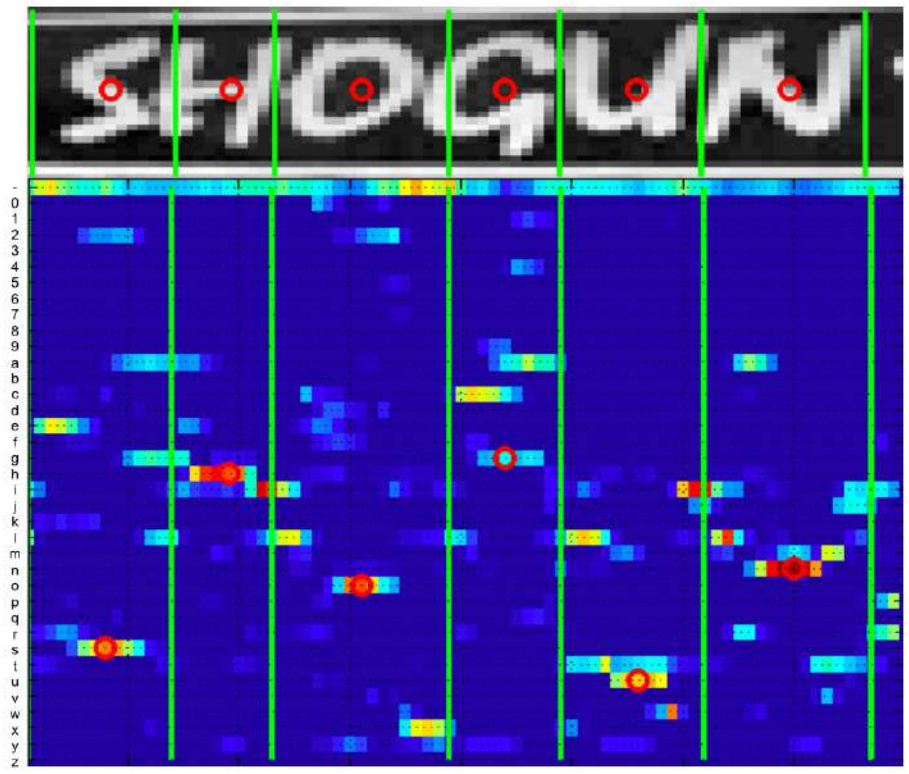
(RLSA): for each row of pixels the mean μ and standard deviation σ of the spacings between probability peaks are computed and neighbouring regions are connected if the space between them is less than $3\mu - 0.5\sigma$. Finding connected components of the linked regions results in candidate text lines.

The next step is to **split text lines into words**. For this, the image is cropped to just that of a text line and Otsu thresholding [Otsu, 1979] is applied to roughly segment foreground characters from background. Adjacent connected components (which are hopefully segmented characters) are then connected if their horizontal spacings are less than the mean horizontal spacing for the text line, again using RLSA. The resulting connected components give candidate bounding boxes for individual words, which are then added to the global set of bounding boxes at all scales. Finally, these bounding boxes are filtered based on geometric constraints (box height, aspect ratio, *etc.*) and undergo non-maximal suppression sorting them by decreasing average per-pixel text saliency score.

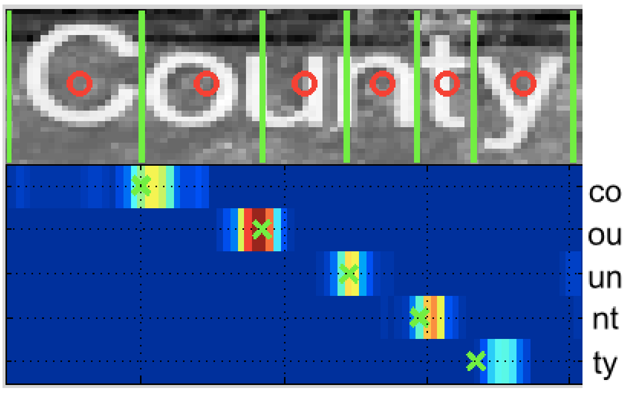
4.2.2 Word Recognition

The aim of the word recognition stage is to take the candidate cropped word images $\mathtt{I} \in \mathbb{R}^{W \times H}$ of width W and height H and estimate the text contained in them. In order to recognise a word from a fixed lexicon, each word hypothesis is scored using a generative model that combines multiple visual cues. The computational complexity is therefore linear in the lexicon size.

The inputs to the word recognition are the 2D character probability maps (case sensitive and insensitive) and bigram probability maps generated using the CNN classifiers. Restricted to the cropped word region, this results in a $W \times H$ map for each character hypothesis. These $W \times H$ maps are reduced to $W \times 1$ responses by averaging along the columns (averaging uses a Gaussian weight centred on the middle row). Grouping the 1D responses per classifier type, this result in matrices



(a)



(b)

Figure 4.4: (a) The optimal placing of breakpoints for the word "SHOGUN" in the image, with the 1D character response map for 37 character classes below. Each row of the response map is the flattened CNN response for a particular character, with classes in row order from top to bottom: no-text, 0-9, a-z (*i.e.* first row is the no-text class response, last row is the "z" class response). (b) The optimal breakpoint placing for "County" with the bigram responses of only the groundtruth text bigrams shown below. Green lines show the placed breakpoints b^w with red circles showing the implied character center.

$\mathtt{P} \in \mathbb{R}^{37\times W}$, $\mathtt{Q} \in \mathbb{R}^{63\times W}$, $\mathtt{R} \in \mathbb{R}^{604\times W}$ for the case-insensitive, case-sensitive, and bigram classifier classifiers respectively. Figure 4.4 (a) shows the case-insensitive response matrix $\mathtt{P}$ for a particular example, and Figure 4.4 (b) shows a select few rows from the bigram response matrix $\mathtt{R}$ for another example.

Given matrices $\mathtt{P}, \mathtt{Q}, \mathtt{R}$, the next step is to score each word hypothesis $w = (c_1, c_2, \dots, c_{L_w})$, where L_w is the number of characters in the word. Let $b^w = (b^w_1, b^w_2, \dots, b^w_{L_w+1})$ denote the $L_w + 1$ positions of breakpoints between characters – b^w_1 marks the beginning of the first character, $b^w_{L_w}$ the beginning of the last character (and therefore also the end of the $L_w - 1$'th character), and $b^w_{L_w+1}$ the end of the last character. The word-breakpoints hypothesis (w, b^w) receives a score

$$s(w, b^w, \mathtt{P}, \mathtt{Q}, \mathtt{R}) = \frac{1}{|b^w|}\left(\sum_{i=1}^{|b^w|} m_i(b^w_i, \mathtt{R}) + \sum_{i=2}^{|b^w|} \phi(b^w_i, b^w_{i-1}, \mathtt{P}, \mathtt{Q}, \mathtt{R})\right). \tag{4.2}$$

This score represents how well the word w is explained by the breakpoints b^w and the response matrices $\mathtt{P}$, $\mathtt{Q}$, and $\mathtt{R}$. This score is computed for each word hypothesis w and maximised by selecting the optimal location of breakpoints using dynamic programming. The final recognised result w^* is taken as the word with the best score:

$$w^* = \arg\max_w \left\{\max_{b^w} s(w, b^w, \mathtt{P}, \mathtt{Q}, \mathtt{R})\right\}. \tag{4.3}$$

The unary score $m_i(b^w_i, \mathtt{R})$ of Equation 4.2 combines the following cues: distance from expected breakpoint placement, distance to out of image bounds, no-text class score, the bigram score, and, for the first and last breakpoint, the distance from the edge of the image. The pairwise score $\phi(b^w_i, b^w_{i-1}, \mathtt{P}, \mathtt{Q}, \mathtt{R})$ combines: the character score at the midpoint between breakpoints, the no-text score at the character center, the deviation from the average width of the character, and a dynamic contribution from the left and right bigram responses relative to the character score – this allows

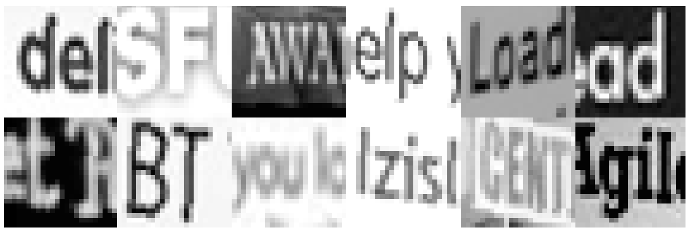


Figure 4.5: Some training samples used for bigram classification as seen by the CNN. From left to right, the top row labels are “de”, “sf”, “aw”, “lp”, “oa”, and “ad”.

bigram responses to take control when it is difficult to classify the character in focus, but easy to classify characters on either side. Also it is ensured that there is no violation of the sequence of characters in the word and that the character centres are all in the region of the word image. Each score is weighted and linearly combined, with the parameters found by grid search on a validation set.

Given the recognised word, the bounding box is adjusted to match the estimated breakpoints and added to a list of candidate recognised word regions. The final step is to perform non-maximal suppression based on overlap on this set of bounding boxes in order to eliminate duplicate detections.

4.3 Experiments

This section evaluates our method on a number of standard text-spotting benchmarks. Data and technical details are discussed next, with results in Section 4.3.2. A full description of all the datasets referenced can be found in Section 3.1.

4.3.1 Classifier Training Data

The case-insensitive character classifier is trained on 163k cropped 24×24 pixel characters from ICDAR 2003, 2005, 2011, 2013 training sets, KAIST, the natural images from Chars74k (we do not use the synthetically generated images), and StanfordSynth. We also use the characters from FlickrType – our own automatically generated dataset created by mining Flickr (see Section 3.2.1).

After this first round of training, the characters in the SVT training set and Oxford Cornmarket dataset are automatically annotated using the process described in Section 3.2.2. We continue training including the newly annotated samples, giving a total of 186k training character images. No-text data is generated from all four ICDAR training datasets (all other datasets do not annotate all text occurrences). The case-sensitive character classifier is trained on the same data, excluding FlickrType and automatically annotated characters due to the lack of case-sensitive labels, giving 107k training samples. Wherever possible, the characters were cropped and resized to maintain their original aspect ratio.

The bigram classifier performs 604-way classification – the number of unique bigrams present in the ICDAR 2003 and SVT lexicons. We train on 24×24 pixel samples, generated by centering a window with width 1.5 times that of the height at the breakpoint between two characters, cropping the image, and resizing to 24×24 pixels, thus squashing the aspect ratio. We use the character annotations from the ICDAR 2003, 2005, 2011, 2013 training sets, KAIST dataset, FlickrType and the automatically annotated datasets giving a total of 92k samples. However, due to the relative distribution of bigrams in natural text, some bigram classes have no training data, while others have thousands of samples – on average there are 152 image samples per bigram class.

4.3.2 Results

This section compares the performance of our system against the standard benchmarks and state-of-the-art. It also reports the performance of the individual components of the system.

4.3.2.1 CNN Classifiers

Table 4.1 shows the results of the trained classifiers tested on ICDAR 2003 cropped characters (IC03) and SVT automatically annotated cropped characters, as well as

	Character Classifier (%)		Case-sensitive Character Classifier (%)	Text/No-text Classifier (%)		Bigram Classifier (%)
Method	IC03	SVT	IC03	IC03	SVT	IC03
Wang et al., 2012	-	-	83.9	97.8*	-	-
Alsharif and Pineau, 2014	89.8	-	86.0	-	-	-
This chapter	**91.3**	**80.3**	**86.8**	**98.2**	**97.1**	**72.5**

Table 4.1: The accuracy of classifiers for 36-way character classification, 62-way case-sensitive character classification, 2-way text detection, and 604-way bigram classification on groundtruth cropped patches. For the SVT dataset the character-level annotation is automatically generated by our system. *Value read from graph.

ICDAR 2003 cropped bigrams. To make results comparable with published ones where the background class is not included in the test problem, we ignore the background class response. The 37-way case-insensitive character classifier and case-sensitive classifier both achieve state-of-the-art performance, as does the text/no-text classifier used for detection. Our bigram classifier gives a recognition accuracy of 72.5%, a good result for a problem with 604 classes.

Although the CNNs are large (2.6 million parameters for the case-insensitive classifier), the method does not require any unsupervised pre-training as is done in Wang et al., 2012, and incorporating the unsupervised approach described in Wang et al., 2012; Coates et al., 2011 gives no improvement. Empirically, maxout and dropout were found to be essential to achieve this performance. For example, replacing maxout with ReLU non-linearities (this equates to reducing the number of filters to give the same layer output dimensionality) causes slight overfitting hence worse accuracy (−3.3% accuracy for case-insensitive character classification). We also found experimentally that pooling and downsampling have no effect on classifier accuracy.

Sharing feature maps between tasks improves results compared to learning independent models: +3% accuracy for text/no-text, +1% accuracy for the case-sensitive character classifier, and +2.5% accuracy for the bigram classifier. Including the FlickrType mined data also gives an extra 0.8% accuracy for the case-insensitive

Method	Cropped Words		
	IC03-50	IC03-Full	SVT-50
Wang et al., 2011	76.0	62.0	57.0
Mishra et al., 2012	81.8	67.8	73.2
Novikova et al., 2012	82.8	-	72.9
Wang et al., 2012	90.0	84.0	70.0
Alsharif and Pineau, 2014	93.1	88.6	74.3
Goel et al., 2013	-	-	77.3
Bissacco et al., 2013b	-	-	**90.4**
This chapter	**96.2**	**91.5**	86.1

Table 4.2: Ground-truth cropped word recognition accuracy (%) on different datasets.

Method	End-to-End		
	IC03-50	IC03-Full	SVT-50
Wang et al., 2011	68	51	38
Weak Baseline (Section 3.2.1.1)	-	55	41
Wang et al., 2012	72	67	46
Alsharif and Pineau, 2014	77	70	48
This chapter	**80**	**75**	**56**

Table 4.3: End-to-end word recognition F-measure results (%). These methods report PASCAL VOC style 50% overlap match measure for the detection.

character classifier, illustrating the importance of more training data. On the contrary, learning on extra synthetic data from the Chars74k dataset (very simple black and white renderings of different fonts) and Wang et al., 2011 (more realistic textured renderings of fonts) harmed recognition accuracy by causing the CNN to overfit to the synthetic data. While these improvements are seemingly small, when integrated across a textline even a small reduction in character classifier error can result in a noticeable reduction in word recognition error.

The classifiers, including code to reproduce Table 4.1 can be found at `https://bitbucket.org/jaderberg/eccv2014_textspotting`.

4.3.2.2 Cropped Word Recognition

The cropped word recognition accuracy of the recognition sub-system (Table 4.2) is evaluated following the protocol of Wang et al., 2011 (in particular, words smaller

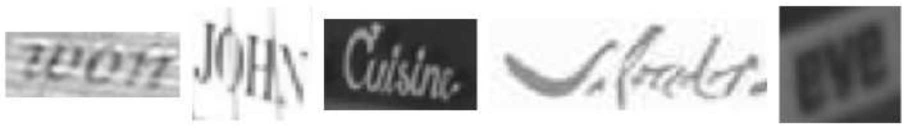


Figure 4.6: Five randomly chosen cropped groundtruth cropped words out of only 33 that were recognised incorrectly in the IC03-50 cropped word benchmark.

than two characters are ignored). The datasets used for this evaluation are summarised in Table 3.1.

For each test image, the correct word must be picked out of a list of words, a lexicon, consisting of the groundtruth word as well as a number of distractor words. These distractors are: in IC03-Full the full lexicon, in IC03-50 the 50 words from Wang et al., 2011, and in SVT, 50 selected words. These datasets are described fully in Section 3.1.

Our word recognition system gives state of the art accuracy on the ICDAR 2003 benchmarks, improving on state of the art by 3.1% for the 50 word lexicon and 2.4% for the full lexicon. The total recognition accuracy of 96.2% for the 50 word lexicon makes only 33 mistakes out of 860 test images, and many of the misclassified examples can be very difficult to classify even for a human (Figure 4.6). On the SVT dataset, we achieve an accuracy of 86.1%, which while 4.3% off state-of-the-art, improves on the next best result [Goel et al., 2013] by 8.8%. This is a competitive result considering our method is trained on two orders of magnitude less data, and so must use a smaller model than the state-of-the-art PhotoOCR [Bissacco et al., 2013b] method.

4.3.2.3 Text Spotting

The results of our end-to-end text spotting system are evaluated on the 251 ICDAR 2003 test images with different sized lexicons (Section 3.1.1) and the 249 image SVT dataset (Section 3.1.2). The results are shown in Table 4.3 and the associated precision recall curves in Figure 5.9. A recognition result is considered to be correct if the bounding box has at least 50% overlap with the groundtruth and the text is correct. The overlap for bounding boxes b_1 and b_2 is defined as the ratio of intersection over union (IoU): $\frac{|b_1 \cap b_2|}{|b_1 \cup b_2|}$.

The detector described in Section 4.2.1 is tuned for high recall and generates word bounding boxes with localisation P/R (precision/recall) of 0.19/0.80 (IC03) and 0.04/0.72 (SVT). After word recognition, the detection results are re-ranked by their word recognition scores, P/R curves generated, and the P/R/F-measure at the maximum F-measure working point is reported [Wang et al., 2012; Wang et al., 2011; Alsharif and Pineau, 2014]. Our end-to-end pipeline outperforms previous works, with P/R/F-measure of 0.90/0.73/0.80 for IC03-50, 0.89/0.66/0.75 for IC03-Full, and 0.73/0.45/0.56 for SVT-50. Interestingly, due to the fact that our pre-recognition bounding boxes are generated by a detector trained from the same data as the character recogniser, we find that the difference between the localisation and recognition scores to be inline with the cropped word recognition results: at maximum recall, 95% of correctly *localised* words are subsequently correctly *recognised* in IC03-50. Some example end-to-end text spotting results on IC03-50 are shown in Figure 4.7.

When removing the bigram response maps from the word recognition process, F-measure drops significantly from 0.8 to 0.76 for IC03-50 and from 0.56 to 0.50 for SVT-50. This clearly shows the representational power of modelling bigrams in addition to single characters.

Limitations. There are a number of limitations to the text spotting method in this chapter. The main issue is the large amount of time taken to process a single image. Due to the multiple scales and aspect ratios that must be processed in a sliding window manner by the text detection CNN, creating the text saliency maps takes 90s on average for an image from the SVT dataset on a single GPU. Also, although fast for small dictionary sizes (hundreds of words), the computational complexity for finding the optimal word with the generative model of Equation (4.3) is linear in the number of words, making it unfeasible for large dictionaries (tens of thousands of words). This leaves space for faster, more scalable detection and recognition algorithms. In terms of accuracy, one of the main limiting factors is the lack of recall achieved by the detection method – only 72% of words are detected, capping the upper-bound of end-to-end recall. These are issues we hope to address in the next chapter.

4.4 Conclusions

In this chapter we have presented a text spotting system following a conventional character-based detection and recognition framework. However, using a single set of rich, learnt features, allows us to achieve state-of-the-art performance on a number of benchmarks. These results illustrate the power of jointly learning features to build multiple strong classifiers, combining data and labels across different classification tasks. Finally, it should be noted that this framework is not only limited to fixed lexicon recognition, as the major contributions in this chapter are agnostic as to whether a lexicon is used or not.

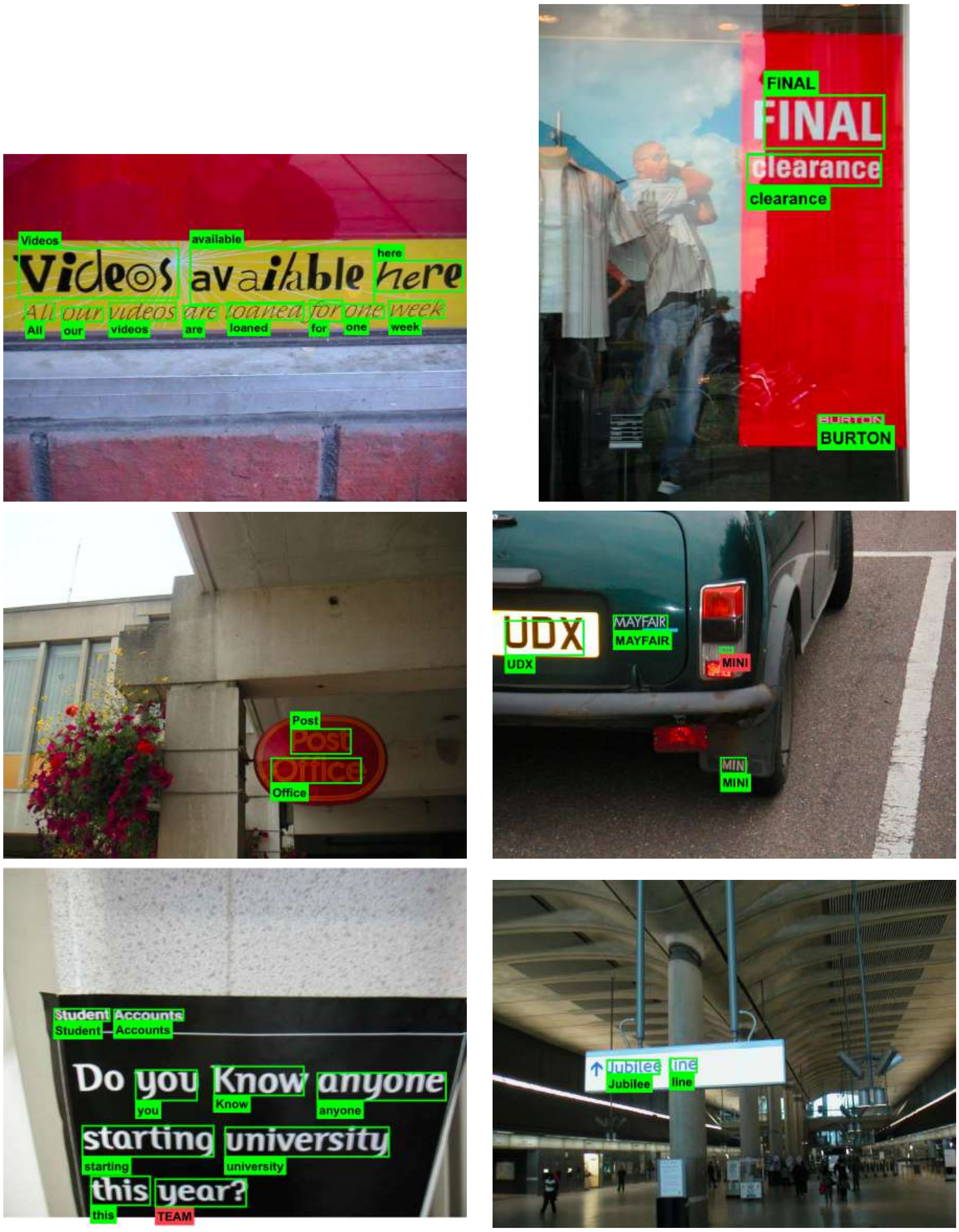
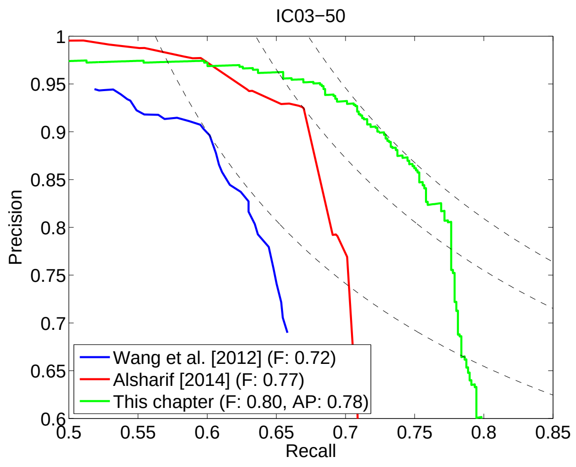
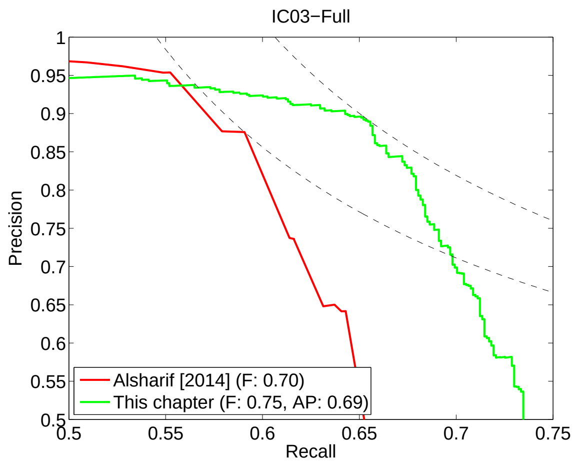


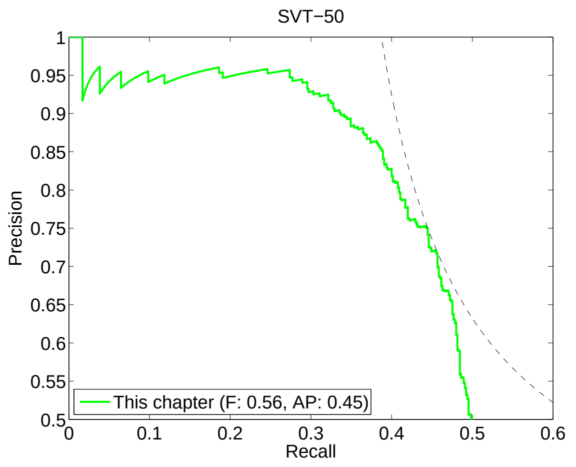
Figure 4.7: Some end-to-end text spotting results from IC03-50. Incorrectly recognised words are labelled in red.



(a)



(b)



(c)

Figure 4.8: The precision/recall curves on the IC03-50 dataset (a), IC03-Full (b), and SVT-50 dataset (c) with lines of constant F-measure. The results from [Wang et al., 2012; Alsharif and Pineau, 2014] were extracted from the papers. Given values are maximum F-measure, and additionally, average precision for the proposed method of this chapter.

Chapter 5

Word-centric Text Spotting

The majority of full text spotting systems, as well as our own framework described in Chapter 4, take a character-centric approach to text detection and recognition, which has been shown to work successfully. However, within these pipelines, algorithms assume that characters occur in clusters (words, text lines, paragraphs) and include heuristics and grouping mechanisms to discover these clusters which are subsequently recognised. Indeed, a single, lone character occurring in an image can be a very ambiguous object, and it is often impossible to distinguish between character-like objects that are part of text from those that are just by-products of the rest of the world, when isolated from their context. For example the visual shapes of the characters “I”, “x”, and “o” occur frequently in natural images without any connection to text. Only in the context of other spatially-near, character-like objects can these shapes be distinguished as characters and not background structures. Even when looking just at a single word, the similarity between characters (such as “I” and “1”) and the inter-character spaces can cause confusion when trying to recognise single characters without any context. To help alleviate this issue in Chapter 4, we increased the context of character recognition by including a bigram classifier, which was shown to greatly increase the recognition accuracy of the proposed system.

Aiming to completely remove the ambiguity of single-character context, in this chapter, we move away from modelling individual characters, and deal solely with words. This chapter describes a new end-to-end text spotting system that takes a *word-centric* approach, visually modelling whole words for detection and recognition. By focussing our visual models on whole words rather than single characters, we hope to remove the ambiguity that arises by forcing context and pooling evidence from entire words, helping both detection and recognition performance. In fact, there is a lot of evidence to show that reading words as a whole, rather than character-by-character, is much closer to the way humans read [Coltheart et al., 2001].

In this chapter we continue to build upon the success of deep learning methods and propose a new model for text recognition. This model is in the form of a deep convolutional neural network which takes the *whole word image as input* to the network. Evidence is gradually pooled from across the image to perform classification of the word across a huge dictionary, such as the 90k-word dictionary evaluated in this chapter. Remarkably, our model is trained *purely on synthetic data*, without incurring the cost of human labelling. Due to the scale of this modelling task, we also propose an *incremental learning* method to successfully train a model with such a large number of classes. Our recognition framework is exceptionally powerful, substantially outperforming previous state of the art on real-world scene text recognition, without using any real-world labelled training data.

We also describe a new text detection strategy: the use of fast region proposal methods to perform word detection. We use a combination of an object-agnostic region proposal method and a learnt sliding window detector. This gives very high recall coverage of individual word bounding boxes, resulting in around 98% word recall on both ICDAR 2003 and Street View Text datasets with a manageable number of proposals. After the initial proposal stage, false-positive candidate word bounding boxes are filtered with a stronger random forest classifier and the remaining proposals adjusted using a CNN trained to regress the bounding box coordinates.

The performance of each part of the pipeline is exposed in experiments, showing that we can maintain the high recall of the initial proposal stage while gradually boosting precision as more complex models and higher order information is incorporated. The recall of the detection stage is shown to be significantly higher than that of previous text detection methods, and the accuracy of the word recognition stage higher than all previous methods. The result is an end-to-end text spotting system that outperforms all previous methods by a large margin. We demonstrate this for the text spotting task (detecting and recognising text in images) across a large range of standard text spotting datasets, as well as in a retrieval scenario (retrieving a ranked list of images that contain the text of a query string) for standard datasets.

The following section gives an overview of this chapter's pipeline. Sections 5.2-5.5 present the stages of our pipeline. We extensively test all elements of our pipeline in Section 5.6 and include the details of datasets and the experimental setup. Finally, Section 5.7 summarises and concludes.

5.1 Overview of the Approach

The stages of the approach in this chapter are as follows: word bounding box proposal generation (Section 5.2), proposal filtering and adjustments (Section 5.3), text recognition (Section 5.4) and final merging for the specific task (Section 5.5). The full process is illustrated in Figure 5.1.

This process loosely follows the detection/recognition separation [Chen and Suter, 2004] – a word detection stage followed by a word recognition stage as discussed in Section 2.2. However, these two stages are not wholly distinct, as we use the information gained from word recognition to vote for, merge and refine detection results at the end, leading to a stronger holistic text spotting system.

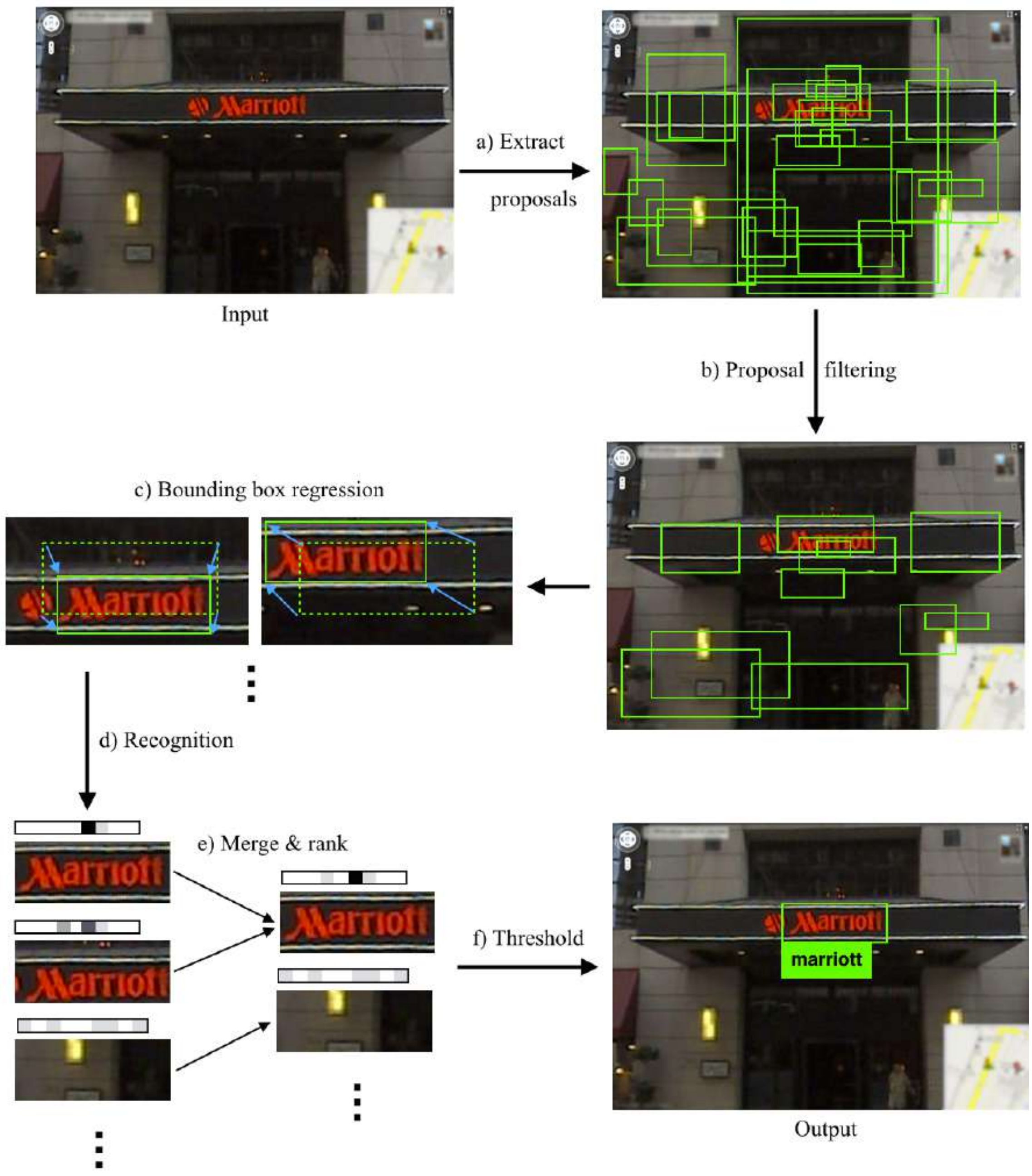


Figure 5.1: The end-to-end text spotting pipeline proposed. a) A combination of region proposal methods extracts many word bounding box proposals. b) Proposals are filtered with a random forest classifier reducing number of false-positive detections. c) A CNN is used to perform bounding box regression for refining the proposals. d) A CNN performs text recognition on each of the refined proposals. e) Detections are merged based on proximity and recognition results and assigned a score. f) Thresholding the detections produces in the final text spotting result.

The detection stage of our pipeline is based on weak-but-fast detection methods to generate word bounding-box proposals. This draws on the success of the R-CNN object detection framework of Girshick et al., 2014 where region proposals are mapped to a fixed size for CNN recognition. The use of region proposals avoids the computational complexity of evaluating an expensive classifier with exhaustive multi-scale, multi-aspect-ratio sliding window search. We use a combination of Edge Box proposals [Zitnick and Dollár, 2014] and a trained aggregate channel features detector [Dollár et al., 2014] to generate candidate word bounding boxes. Due to the large number of false-positive proposals, we then use a random forest classifier to filter the number of proposals to a manageable size – this is a stronger classifier than those found in the proposal algorithms. Finally, inspired by the success of bounding box regression in DPM [Felzenszwalb et al., 2010] and R-CNN [Girshick et al., 2014], we regress more accurate bounding boxes from the seeds of the proposal algorithms which greatly improves the average overlap ratio of positive detections with groundtruth. However, unlike the linear regressors of Felzenszwalb et al., 2010 and Girshick et al., 2014 we train a non-linear CNN specifically for regression. We discuss these design choices in each section.

The second stage of our framework produces a text recognition result for each proposal generated from the detection stage. We take a whole-word approach to recognition, providing the entire cropped region of the word as input to a deep convolutional neural network. We present a dictionary model which poses the recognition task as a multi-way classification task across a dictionary of 90k possible words. Due to the mammoth training data requirements of classification tasks of this scale, these models are trained *purely from synthetic data*. Our synthetic data engine, described in Section 3.3, is capable of rendering sufficiently realistic and variable word image samples that the models trained on this data translate to the domain of real-world word images giving state-of-the-art recognition accuracy.

Finally, we use the information gleaned from recognition to update our detection results with multiple rounds of box merging, non-maximal suppression, bounding box regression and recognition on the new set of bounding boxes. This sequence is repeated multiple times, iteratively incorporating the cues from the recognition model into the detection model, increasing the overall precision and recall.

5.2 Proposal Generation

The first stage of our word-centric text spotting pipeline relies on the generation of word bounding boxes. This is word detection – in an ideal scenario we would be able to generate word bounding boxes with high recall and high precision, achieving this by extracting the maximum amount of information from each bounding box candidate possible. However, in practice a precision/recall tradeoff is required to reduce computational complexity. With this in mind we opt for a fast, high recall initial phase, using computationally cheap classifiers, and gradually incorporate more information and more complex models to improve precision by rejecting false-positive detections resulting in a cascade. To compute recall and precision in a detection scenario, a bounding box is said to be a true-positive detection if it has overlap with a groundtruth bounding box above a defined threshold. As a reminder, the overlap for bounding boxes b_1 and b_2 is defined as the ratio of intersection over union (IoU): $\frac{|b_1 \cap b_2|}{|b_1 \cup b_2|}$.

Though never applied to word detection before, region proposal methods have gained a lot of attention for generic object detection. Region proposal methods [Uijlings et al., 2013; Alexe et al., 2012; Cheng et al., 2014; Zitnick and Dollár, 2014] aim to generate object region proposals with high recall, but at the cost of a large number of false-positive detections. Even so, this still reduces the search space drastically compared to sliding window evaluation of the subsequent stages of a detection pipeline. Effectively, region proposal methods can be viewed as a weak detector.

In this work we combine the results of two detection mechanisms – the Edge Boxes region proposal algorithm (Zitnick and Dollár, 2014, Section 5.2.1) and a weak aggregate channel features detector (Dollár et al., 2014, Section 5.2.2).

5.2.1 Edge Boxes

We use the formulation of Edge Boxes as described in Zitnick and Dollár, 2014. The key intuition behind Edge Boxes is that since objects are generally self contained, the number of contours wholly enclosed by a bounding box is indicative of the likelihood of the box containing an object. Edges tend to correspond to object boundaries, and so if edges are contained inside a bounding box this implies objects are contained within the bounding box, whereas edges which cross the border of the bounding box suggest there is an object that is not wholly contained by the bounding box.

The notion of an object being a collection of boundaries is especially true when the desired objects are words – collections of characters with sharp boundaries.

Following Zitnick and Dollár, 2014, we compute the edge response map using the Structured Edge detector [Dollár and Zitnick, 2013; Dollár and Zitnick, 2014] and perform non-maximal suppression orthogonal to the edge responses, sparsifying the edge map. A candidate bounding box b is assigned a score s_b based on the number of edges wholly contained by b, normalised by the perimeter of b. The complete details can be found in Zitnick and Dollár, 2014.

The boxes b are evaluated in a sliding window manner, over multiple scales and aspect ratios, and given a score s_b. Finally, the boxes are sorted by score and non-maximal suppression is performed: a box is removed if its overlap with another box of higher score is more than a threshold. This results in a set of candidate bounding boxes for words $\mathcal{B}_e$.

5.2.2 Aggregate Channel Feature Detector

Another method for generating candidate word bounding box proposals is by using a conventional trained detector. We use the aggregate channel features (ACF) detector framework of Dollár et al., 2014 for its speed of computation. This is a conventional sliding window detector based on ACF features coupled with an AdaBoost classifier. ACF based detectors have been shown to work well on pedestrian detection and general object detection, and here we use the same framework for word detection.

For each image $\mathtt{I}$ a number of feature channels are computed, such that channel $\mathtt{C} = \Omega(\mathtt{I})$, where Ω is the channel feature extraction function. We use the same channels to those in Dollár et al., 2010 – normalised gradient magnitude, histogram of oriented gradients (6 channels) – as well as the additional raw greyscale input as an extra channel. We follow the same process as in Dollár et al., 2010: each channel $\mathtt{C}$ is smoothed, divided into blocks, and the pixels in each block are summed and smoothed again, resulting in aggregate channel features.

The ACF features are not scale invariant, so for multi-scale detection we need to extract features at many different scales – a feature pyramid. In a standard detection pipeline, the channel features for a particular scale s are computed by resampling the image and recomputing the channel features $\mathtt{C}_s = \Omega(\mathtt{I}_s)$ where $\mathtt{C}_s$ are the channel features at scale s and $\mathtt{I}_s = R(\mathtt{I}, s)$ is the image resampled by s. Resampling and recomputing the features at every scale is computationally expensive. However, as shown in Dollár et al., 2010; Dollár et al., 2014, the channel features at scale s can be approximated by resampling the features at a different scale, such that $\mathtt{C}_s \approx R(\mathtt{C}, s) \cdot s^{-\lambda_\Omega}$, where λ_Ω is a channel specific power-law factor. Therefore, fast feature pyramids can be computed by evaluating $\mathtt{C}_s = \Omega(R(\mathtt{I}, s))$ at only a single scale per octave ($s \in \{1, \frac{1}{2}, \frac{1}{4}, \dots\}$) and at intermediate scales, $\mathtt{C}_s$ is computed using $\mathtt{C}_s = R(\mathtt{C}_{s'}, s/s')(s/s')^{-\lambda_\Omega}$ where $s' \in \{1, \frac{1}{2}, \frac{1}{4}, \dots\}$. This results in much faster feature pyramid computation.

The sliding window classifier is an ensemble of weak decision trees, trained using AdaBoost [Friedman et al., 2000], using the aggregate channel features. We evaluate the classifier on every block of aggregate channel features in our feature pyramid, and repeat this for multiple aspect ratios to account for different length words, giving a score for each box. Thresholding on score gives a set of word proposal bounding boxes from the detector, $\mathcal{B}_d$.

Discussion. We experimented with a number of region proposal algorithms. Some were too slow to be useful [Uijlings et al., 2013; Alexe et al., 2012]. A very fast method, BING [Cheng et al., 2014], gives good recall when re-trained specifically for word detection but achieves a low overlap ratio for detections and poorer overall recall. We found Edge Boxes to give the best recall and overlap ratio. We also experimented with using existing text detection methods such as the stroke width transform [Epshtein et al., 2010] and ER grouping [Neumann and Matas, 2012] to provide word region proposals (these algorithms are described in Section 2.2). These had high precision but comparatively low recall so were not used.

It was also observed that independently, neither Edge Boxes nor the ACF detector achieve particularly high recall, 92% and 70% recall respectively (see Section 5.6.2 for experiments), but when the proposals are combined achieve 98% recall (recall is computed at 0.5 overlap). In contrast, combining BING, with a recall of 86% with the ACF detector gives a combined recall of only 92%. This suggests that the Edge Box and ACF detector methods are very complementary when used in conjunction, and so we compose the final set of candidate bounding boxes $\mathcal{B} = \{\mathcal{B}_e \cup \mathcal{B}_d\}$.

5.3 Filtering & Refinement

The proposal generation stage of Section 5.2 produces a set of candidate word bounding boxes $\mathcal{B}$. However, to achieve a high recall, thousands of bounding boxes are generated, most of which are false-positives. We therefore aim to use a stronger classifier to further filter these to a number that is computationally manageable for the more expensive full text recognition stage. This word/no-word classifier is described in Section 5.3.1. We also observe that the overlap of many of the bounding boxes with the groundtruth is unsatisfactorily low, and therefore train a regressor to refine the location of the bounding boxes, as described in Section 5.3.2.

5.3.1 Word Classification

To reduce the number of false-positive word detections, we seek a classifier to perform word/no-word binary classification. For this we use a random forest classifier [Breiman, 2001] acting on HOG features [Felzenszwalb et al., 2010].

For each bounding box proposal $b \in \mathcal{B}$ we resample the cropped image region to a fixed size and extract HOG features, resulting in a descriptor $\mathbf{h}$. The descriptor is then classified with a random forest classifier, with decision stump nodes. The random forest classifies every proposal, and the proposals falling below a certain threshold are rejected, leaving a filtered set of bounding boxes $\mathcal{B}_f$.

For the details on training the random forest classifier please see Section 5.6.2.

5.3.2 Bounding Box Regression

Although our proposal mechanism and filtering stage give very high recall, the overlap of these proposals can be quite poor.

While an overlap of 0.5 is usually acceptable for general object detection [Everingham et al., 2010], for accurate text recognition this can be unsatisfactory. This is especially true when one edge of the bounding box is predicted accurately but not the other – *e.g.* if the height of the bounding box is computed perfectly, the width of the bounding box can be either double or half as wide as it should be and still achieve 0.5 overlap. This is illustrated in Figure 5.2. Both predicted bounding boxes have 0.5 overlap with the groundtruth, but for text this can amount to only seeing

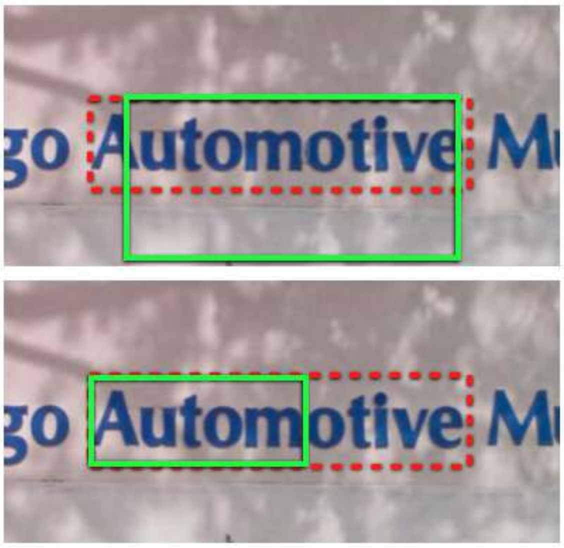


Figure 5.2: The shortcoming of using an overlap ratio for text detection of 0.5. The two examples of proposal bounding boxes (green solid box) have approximately 0.5 overlap with groundtruth (red dashed box). In the bottom case, a 0.5 overlap is not satisfactory to produce accurate text recognition results.

half the word if the height is correctly computed, and so it would be impossible to recognise the correct word in the bottom example of Figure 5.2. Note that both proposals contain text, so neither are filtered by the word/no-word classifier.

Due to the large number of region proposals, we could hope that there would be some proposals that would overlap with the groundtruth to a satisfactory degree, but we can encourage this by explicitly refining the coordinates of the proposed bounding boxes – we do this within a regression framework.

Our bounding box coordinate regressor takes each proposed bounding box $b \in \mathcal{B}_f$ and produces an updated estimate of that proposal b^*. A bounding box is parametrised by its top-left and bottom-right corners, such that bounding box $b = (x_1, y_1, x_2, y_2)$. The full image $\mathtt{I}$ is cropped to a rectangle centred on the region b, with the width and height inflated by a scale factor. The resulting image is resampled to a fixed size $W \times H$, giving $\mathtt{I}_b$, which is processed by the CNN to regress the four values of b^*. We do not regress the absolute values of the bounding box coordinates directly, but rather encoded values. The top-left coordinate is encoded by the top-left quadrant of $\mathtt{I}_b$, and the bottom left coordinate by the bottom-left quadrant of $\mathtt{I}_b$ as illustrated by Figure 5.3. This normalises the coordinates to generally fall in the interval $[0, 1]$, but allows the breaking of this interval if required.

In practice, we inflate the cropping region of each proposal by a factor of two. This gives the CNN enough context to predict a more accurate location of the proposal bounding box. The CNN is trained with example pairs of $(\mathtt{I}_b, b_{gt})$ to regress the groundtruth bounding box b_{gt} from the sub-image $\mathtt{I}_b$ cropped from $\mathtt{I}$ by the estimated bounding box b. This is done by minimising the L_2 loss between the encoded bounding boxes, *i.e.*

$$\min_{\psi} \sum_{b \in \mathcal{B}_{train}} \| g(\mathtt{I}_b; \psi) - q(b_{gt}) \|_2^2 \tag{5.1}$$

over the network parameters ψ on a training set $\mathcal{B}_{train}$, where g is the CNN forward pass function and q is the bounding box coordinate encoder.

Discussion. The choice of which features and the classifier and regression methods to use was made through experimenting with a number of different options. This included using a CNN for classification, with a dedicated classification CNN, and also by jointly training a single CNN to perform both classification and regression simultaneously with multi-task learning. However, the classification performance of the CNN was not significantly better than that of HOG with a random forest, but requires more computations and a GPU for processing. We therefore chose the random forest classifier to reduce the computational cost of our pipeline without impacting end results.

The bounding box regression not only improves the overlap to aid text recognition for each individual sample, but also causes many proposals to converge on the same bounding box coordinates for a single instance of a word, therefore aiding the

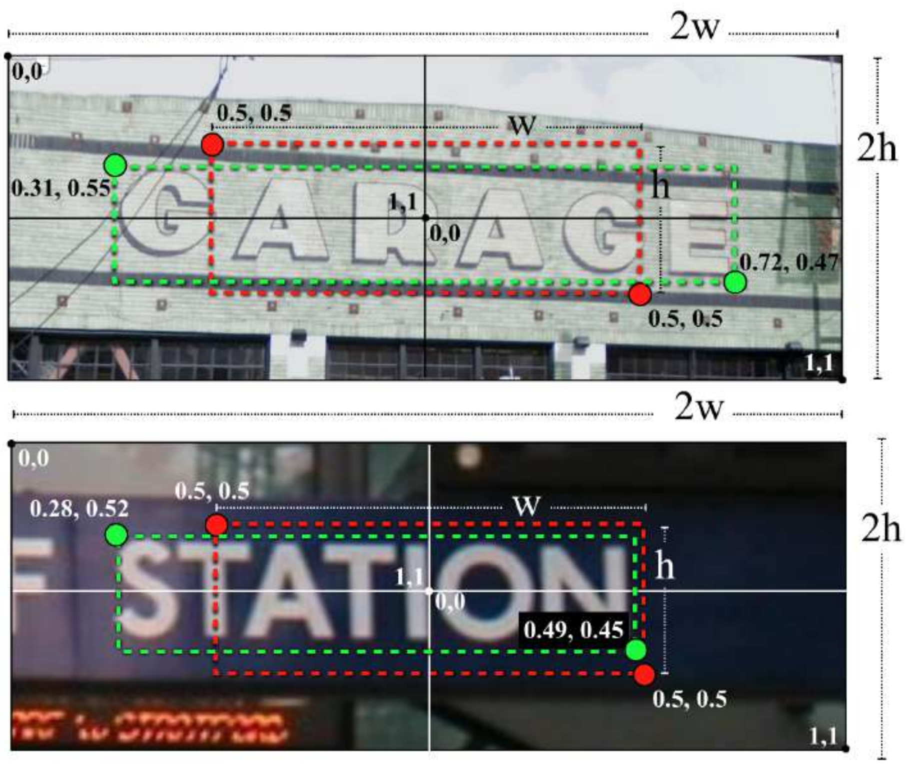


Figure 5.3: The bounding box regression encoding scheme showing the original proposal (red) and the adjusted proposal (green). The cropped input image shown is always centred on the original proposal, meaning the original proposal always has implied encoded coordinates of $(0.5, 0.5, 0.5, 0.5)$.

voting/merging mechanism described in Section 5.6.3.2 with duplicate detections.

5.4 Text Recognition

At this stage of our processing pipeline, a pool of accurate word bounding box proposals has been generated as described in the previous sections. We now turn to the task of recognising words inside these proposed bounding boxes. To this end we use a deep CNN to perform classification across a pre-defined dictionary of words – dictionary encoding – which explicitly models natural language. The cropped image of each of the proposed bounding boxes is taken as input to the CNN, and the CNN produces a probability distribution over all the words in the dictionary. The word with the maximum probability can be taken as the recognition result.

The model, described fully in Section 5.4.1, can scale to a huge dictionary of 90k words, encompassing the majority of the commonly used English language (see Section 5.6.1 for details of the dictionary used). However, to achieve this, many training samples of every different possible word must be amassed. Such a training dataset does not exist, so we instead use our synthetic data engine, described previously in Section 3.3, to train our CNN. This synthetic data is so realistic that the CNN can be trained purely on the synthetic data but still applied to real world data.

The synthetic data engine produces a wide range of synthetic data samples, being drawn from a multitude of random distributions, mimicking real-world samples of scene text images. Here, the synthetic data is used in place of real-world data, and the labels are generated from a corpus or dictionary as desired. By using a synthesised training dataset many orders of magnitude larger than what has been available before, we are able to use data-hungry deep learning algorithms to train a richer, whole-word-based model.

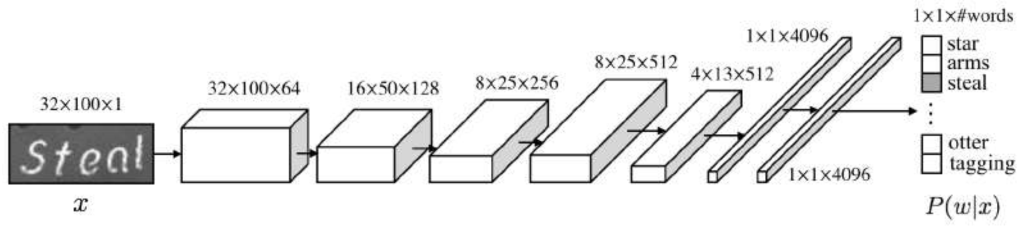


Figure 5.4: A schematic of the CNN used for text recognition by word classification. The dimensions of the featuremaps at each layer of the network are shown.

5.4.1 CNN Model

This section describes our model for word recognition. We formulate recognition as a multi-class classification problem, with one class per word, where words w are constrained to be selected in a pre-defined dictionary $\mathcal{W}$. While the dictionary $\mathcal{W}$ of a natural language may seem too large for this approach to be feasible, in practice an advanced English vocabulary, including different word forms, contains only around 90k words, which is large but manageable.

In detail, we propose to use a CNN classifier where each word $w \in \mathcal{W}$ in the lexicon corresponds to an output neuron. We use a CNN with five convolutional layers and three fully-connected layers, with the exact details described in Section 5.6.2. The final fully-connected layer performs classification across the dictionary of words, so has the same number of units as the size of the dictionary we wish to recognise.

The predicted word recognition result w^* out of the set of all dictionary words $\mathcal{W}$ in a language $\mathcal{L}$ for a given input image $\mathtt{x}$ is given by

$$w^* = \arg\max_{w \in \mathcal{W}} P(w|\mathtt{x}, \mathcal{L}). \tag{5.2}$$

With the assumptions that $\mathtt{x}$ is conditionally independent of $\mathcal{L}$ given w, *i.e.* $P(\mathtt{x}|w, \mathcal{L}) = P(\mathtt{x}|\mathcal{L})$, then $P(w|x, \mathcal{L})$ can be written as

$$P(w|\mathtt{x}, \mathcal{L}) = \frac{P(w|\mathtt{x})P(w|\mathcal{L})P(\mathtt{x})}{P(\mathtt{x}|\mathcal{L})P(w)} \tag{5.3}$$

and assuming that prior to any knowledge of our language all words are equally probable, our scoring function reduces to

$$w^* = \arg\max_{w \in \mathcal{W}} P(w|\mathtt{x})P(w|\mathcal{L}). \tag{5.4}$$

The per-word output probability $P(w|\mathtt{x})$ is modelled by the softmax output of the final fully-connected layer of the recognition CNN, and the language based word prior $P(w|\mathcal{L})$ can be modelled by a lexicon or word frequency counts. A schematic of the network is shown in Figure 5.4.

One limitation of this CNN model is that the input $\mathtt{x}$ must be a fixed, pre-defined size. This is problematic for word images, as although the height of the image is always one character tall, the width of a word image is highly dependent on the number of characters in the word, which can range between one and 23 characters. To overcome this issue, we simply resample the word image to a fixed width and height. Although this does not preserve the aspect ratio, the horizontal frequency distortion of image features most likely provides the network with word-length cues. We also experimented with different padding regimes to preserve the aspect ratio, but found that the results are not quite as good as performing naive resampling.

To summarise, for each proposal bounding box $b \in \mathcal{B}_f$ for image $\mathtt{I}$ we compute $P(w|\mathtt{x}_b, \mathcal{L})$ by cropping the image to $\mathtt{I}_b = c(b, \mathtt{I})$, resampling to fixed dimensions $W \times H$ such that $\mathtt{x}_b = R(\mathtt{I}_b, W, H)$, and compute $P(w|\mathtt{x}_b)$ with the text recognition CNN and multiply by $P(w|\mathcal{L})$ (task dependent) to give a final probability distribution over words $P(w|\mathtt{x}_b, \mathcal{L})$.

5.5 Merging & Ranking

At this point in the pipeline, we have a set of word bounding boxes for each image $\mathcal{B}_f$ with their associated word probability distributions $\mathcal{P}_{\mathcal{B}_f} = \{p_b : b \in \mathcal{B}_f\}$, where

$p_b = P(w|b, \mathtt{I}) = P(w|\mathtt{x}_b, \mathcal{L})$. However, this set of detections still contains a number of false-positive and duplicate detections of words, so a final merging and ranking of detections must be performed depending on the task at hand: text spotting or text based image retrieval.

5.5.1 Text Spotting

As defined in Chapter 1, the goal of text spotting is to localise and recognise the individual words in the image. Each word should be labelled by a bounding box enclosing the word and the bounding box should have an associated text label.

For this task, we assign each bounding box in $b \in \mathcal{B}_f$ a label w_b and score s_b according to b's maximum word probability:

$$w_b = \arg\max_{w \in \mathcal{W}} P(w|b, \mathtt{I}), \quad s_b = \max_{w \in \mathcal{W}} P(w|b, \mathtt{I}) \tag{5.5}$$

To cluster duplicate detections of the same word instance, we perform a greedy non maximum suppression (NMS) on detections with *the same word* label, aggregating the scores of suppressed proposals. This can be seen as positional voting for a particular word. Subsequently, we perform NMS to suppress non-maximal detections of *different words* with some overlap.

Our text recognition CNN is able to accurately recognise text in very loosely cropped word sub-images. Because of this, we find that some valid text spotting results have less than 0.5 overlap with groundtruth, but we require greater than 0.5 overlap for some applications (see Section 5.6.3.2).

To improve the overlap of detection results, we additionally perform multiple rounds of bounding box regression as in Section 5.3.2 and NMS as described above to further refine our detections. This can be seen as a recurrent regressor network. Each round of regression updates the prediction of each word's localisation, giving the next round of regression an updated context window to perform the next regression, as

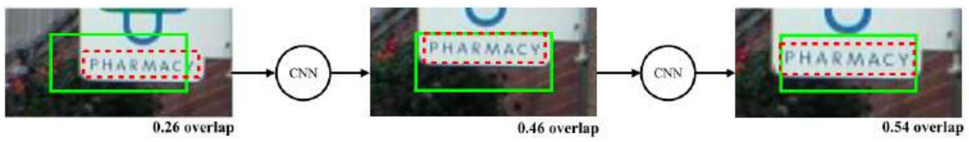


Figure 5.5: An example of the improvement in localisation of the word detection `pharmacy` through multiple rounds of recurrent regression.

shown in Figure 5.5. Performing NMS between each regression causes bounding boxes that have become similar after the latest round of regression to be grouped as a single detection. This generally causes the overlap of detections to converge on a higher, stable value with only a few rounds of recurrent regression.

The refined results, given by the tuple (b, w_b, s_b), are ranked by their scores s_b and a threshold determines the final text spotting result. For the direct comparison of scores across images, we normalise the scores of the results of each image by the maximum score for a detection in that image.

5.5.2 Image Retrieval

For the task of text based image retrieval, we wish to retrieve the list of images which contain the given query words. Localisation of the query word is not required, only optional for giving evidence for retrieving that image.

Retrieval is achieved by, at query time, assigning each image $\mathtt{I}$ a score $s_{\mathtt{I}}^{\mathcal{Q}}$ for the query words $\mathcal{Q} = \{q_1, q_2, \ldots\}$, and sorting the images in the database $\mathcal{I}$ in descending order of score. It is also required that the score for all images can be computed fast enough to scale to databases of millions of images, allowing fast retrieval of visual content by text search. While retrieval is often performed for just a single query word ($\mathcal{Q} = \{q\}$), we generalise our retrieval framework to be able to handle multiple query words.

We estimate the per-image probability distribution across word space $P(w|\mathtt{I})$ by averaging the word probability distributions across all detections $\mathcal{B}_f$ in an image

$$p_{\mathtt{I}} = P(w|\mathtt{I}) = \frac{1}{|\mathcal{B}_f|} \sum_{b \in \mathcal{B}_f} p_b. \tag{5.6}$$

This distribution is computed offline for all $\mathtt{I} \in \mathcal{I}$.

At query time, we can simply compute a score for each image $s_{\mathtt{I}}^{\mathcal{Q}}$ representing the probability that the image $\mathtt{I}$ contains any of the query words $\mathcal{Q}$. Assuming independence between the presence of query words

$$s_{\mathtt{I}}^{\mathcal{Q}} = \sum_{q \in \mathcal{Q}} \log P(q|\mathtt{I}) = \sum_{q \in \mathcal{Q}} \log p_{\mathtt{I}}(q) \tag{5.7}$$

where $p_{\mathtt{I}}(q)$ is just a lookup of the log probability of word q in the word distribution $p_{\mathtt{I}}$. These scores can be computed very quickly and efficiently by constructing an inverted index of $p_{\mathtt{I}}\ \forall\ \mathtt{I} \in \mathcal{I}$.

After a one-time, offline pre-processing to compute $p_{\mathtt{I}}$ and assemble the inverted index, a query can be processed across a database of millions of images in less than a second.

5.6 Experiments

In this section we evaluate the word-centric pipeline in this chapter on a number of standard text spotting and text based image retrieval benchmarks.

We next describe the datasets The datasets used for evaluation are given Section 5.6.1, we give the exact implementation details and performance of each part of our pipeline in Section 5.6.2, and finally present the results on cropped text recognition, text spotting and image retrieval benchmarks in Section 5.6.3.

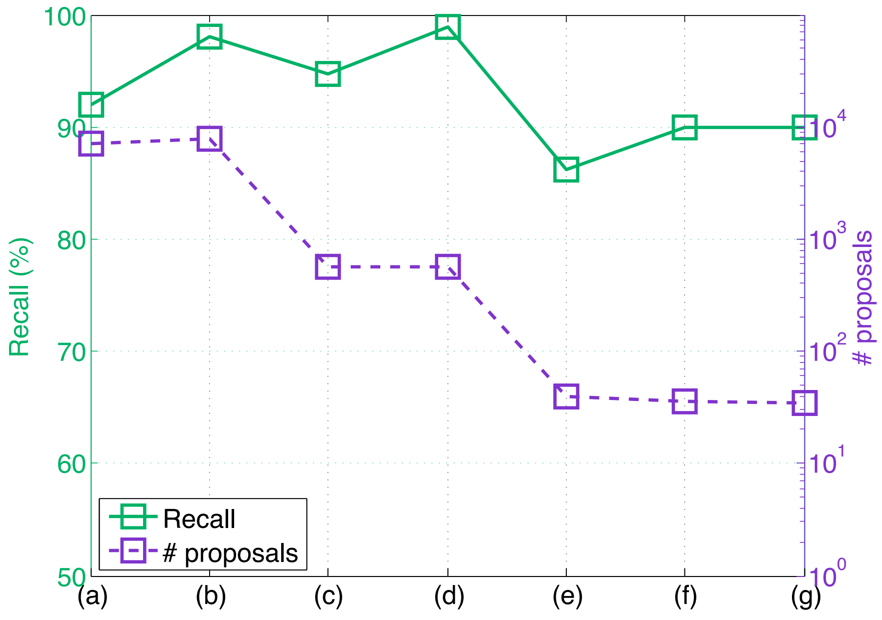


Figure 5.6: The recall and the average number of proposals per image at each stage of the pipeline on IC03. (a) Edge Box proposals, (b) ACF detector proposals, (c) Proposal filtering, (d) Bounding box regression, (e) Regression NMS round 1, (f) Regression NMS round 2, (g) Regression NMS round 3. The recall computed is detection recall across the dataset (*i.e.* ignoring the recognition label) at 0.5 overlap.

5.6.1 Datasets

We evaluate our pipeline on an extensive number of datasets. Due to different levels of annotation, the datasets are used for a combination of text recognition, text spotting, and image retrieval evaluation. The datasets are summarised in Table 3.1, Table 3.2, and Table 3.3. The smaller lexicons provided by some datasets are used to reduce the search space to just text contained within the lexicons.

We evaluate these tasks on the standard ICDAR, SVT and IIIT datasets. Additionally, to assess text based image retrieval we use the queries and datasets released by Mishra et al., 2012. Full details of the datasets are found in Section 3.1.

5.6.2 Implementation Details

We train a single model for each of the stages in our pipeline, and hyper parameters are selected using training datasets of ICDAR and SVT. Exactly the same pipeline, with the same models and hyper parameters are used for all datasets and experiments. This highlights the generalisability of our end-to-end framework to different datasets and tasks. The progression of detection recall and the number of proposals as the pipeline progresses can be seen in Figure 5.6.

5.6.2.1 Edge Boxes & ACF Detector

The Edge Box detector has a number of hyper parameters, controlling the stride of evaluation and non maximal suppression. We use the default values of $\alpha = 0.65$ and $\beta = 0.75$ (see Zitnick and Dollár, 2014 for details of these parameters). In practice, we saw little effect of changing these parameters in combined recall.

For the ACF detector, we set the number of decision trees to be 32, 128, 512 for each round of bootstrapping. For feature aggregation, we use 4×4 blocks smoothed with [1 2 1]/4 filter, with 8 scales per octave. As the detector is trained for a particular aspect ratio, we perform detection at multiple aspect ratios in the range $[1, 1.2, 1.4, \dots, 3]$ to account for variable sized words. We train on 30k cropped 32×100 positive word samples amalgamated from all the ICDAR, SVT, KAIST and FlickrType datasets in Section 3.1. Negative/background patches are randomly sampled from 11k images which do not contain text.

Figure 5.7 shows the performance of our proposal generation stage. The recall at 0.5 overlap of groundtruth labelled words in the IC03 and SVT datasets is shown as a function of the number of proposal regions generated per image. The maximum recall achieved using Edge Boxes is 92%, and the maximum recall achieved by the ACF detector is around 70%. However, combining the proposals from each method increases the recall to 98% at 6k proposals and 97% at 11k proposals for IC03 and

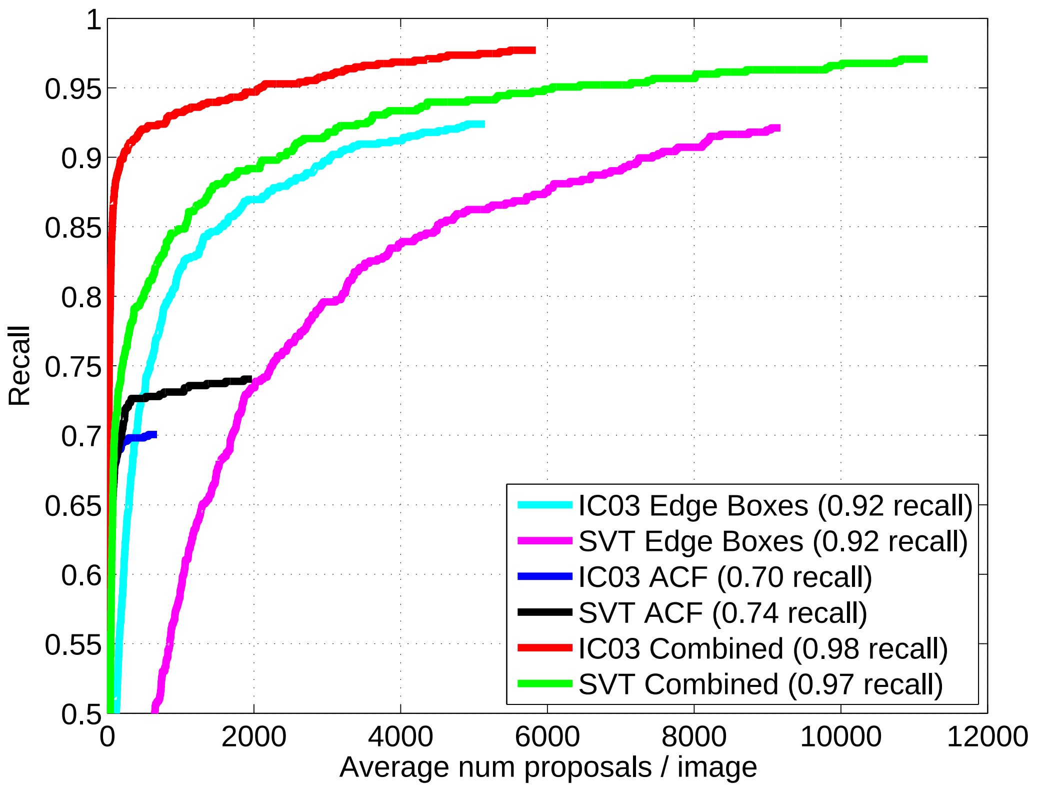


Figure 5.7: The 0.5 overlap recall of different region proposal algorithms. The recall displayed in the legend for each method gives the maximum recall achieved. The curves are generated by decreasing the minimum score for a proposal to be valid, and terminate when no more proposals can be found.

SVT respectively. The average maximum overlap of a particular proposal with a groundtruth bounding box is 0.82 on IC03 and 0.77 on SVT, suggesting the region proposal techniques produce some accurate detections amongst the thousands of false-positives.

This high recall and high overlap gives a good starting point for the rest of our pipeline, and has greatly reduced the search space of word detections from the tens of millions of possible bounding boxes to around 10k proposals per image.

5.6.2.2 Random Forest Word Classifier

The random forest word/no-word binary classifier acts on cropped region proposals. These are resampled to a fixed 32×100 size, and HOG features extracted with a cell size of 4, resulting in $h \in \mathbb{R}^{8\times25\times36}$, a 7200-dimensional descriptor. The random forest classifier consists of 10 trees with a maximum depth of 64.

For training, region proposals are extracted as we describe in Section 5.2 on the training datasets of ICDAR and SVT, with positive bounding box samples defined as having at least 0.5 overlap with groundtruth, and negative samples as less than 0.3 with groundtruth. Due to the abundance of negative samples, we randomly sample a roughly equal number of negative samples to positive samples, giving 300k positive and 400k negative training samples.

Once trained, the result is a very effective false-positive filter. We threshold the probability at 0.5, giving 96.6% and 94.8% recall on IC03 and SVT positive proposal regions respectively. This filtering reduces the total number of region proposals to on average 650 (IC03) and 900 (SVT) proposals per image.

5.6.2.3 Bounding Box Regressor

The bounding box regression CNN consists of four convolutional layers with stride 1 with $\{filter\ size,\ number\ of\ filters\}$ of $\{5, 64\}$, $\{5, 128\}$, $\{3, 256\}$, $\{3, 512\}$ for each layer from input respectively, followed by two fully-connected layers with 4k units and 4 units (one for each regression variable). All hidden layers are followed by rectified linear non-linearities, the inputs to convolutional layers are zero-padded to preserve dimensionality, and the convolutional layers are followed by 2×2 max pooling. The fixed sized input to the CNN is a 32×100 greyscale image which is zero centred by subtracting the image mean and normalised by dividing by the standard deviation.

The CNN is trained with stochastic gradient descent (SGD) with dropout on the fully-connected layers to reduce overfitting, minimising the L_2 distance between the estimated and groundtruth bounding boxes (Equation 5.1). We used 700k training examples of bounding box proposals with greater than 0.5 overlap with groundtruth, computed on the ICDAR and SVT training datasets.

Before the regression stage, the average positive proposal region (with over 0.5 overlap with groundtruth) had an overlap of 0.61 and 0.60 on IC03 and SVT. The CNN regressor improves this average positive overlap to 0.88 and 0.70 for IC03 and SVT.

5.6.2.4 Text Recognition CNN

The text recognition CNN consists of eight weight layers – five convolutional layers and three fully-connected layers. The convolutional layers have the following $\{filter\ size,\ number\ of\ filters\}$: $\{5, 64\}$, $\{5, 128\}$, $\{3, 256\}$, $\{3, 512\}$, $\{3, 512\}$. The first two fully-connected layers have 4k units and the final fully-connected layer has the same number of units as as number of words in the dictionary – 90k words in our case as we use the 90kDict defined in Section 3.3. The final classification layer is followed by a softmax normalisation layer. Rectified linear non-linearities follow every hidden layer, and all but the fourth convolutional layers are followed by 2×2 max pooling. The inputs to convolutional layers are zero-padded to preserve dimensionality. The fixed sized input to the CNN is a 32×100 greyscale image which is zero centred by subtracting the image mean and normalised by dividing by the standard deviation.

We train the network on the Synth90k training data (Section 3.3), back-propagating the standard multinomial logistic regression loss. Optimisation uses SGD with dropout regularisation of fully-connected layers, and we dynamically lower the learning rate as training progresses. With uniform sampling of classes in training data, we found the SGD batch size must be at least a fifth of the total number of classes in order for the network to train.

For very large numbers of classes (*i.e.* over 5k classes), the SGD batch size required to train effectively becomes large, slowing down training a lot. Therefore, for large dictionaries, we perform *incremental training* to avoid requiring a prohibitively large batch size. This involves initially training the network with 5k classes until partial convergence, after which an extra 5k classes are added. The original weights are copied for the previously trained classes, with the extra classification layer weights being randomly initialised. The network is then allowed to continue training, with the extra randomly initialised weights and classes causing a spike in training error, which is quickly trained away. This process of allowing partial convergence on a subset of the classes, before adding in more classes, is repeated until the full number of desired classes is reached.

At evaluation-time we do not do any data augmentation. If a lexicon is provided, we set the language prior $P(w|\mathcal{L})$ to be equal probability for lexicon words, otherwise zero. In the absence of a lexicon, $P(w|\mathcal{L})$ is calculated as the frequency of word w in a corpus (we use the opensubtitles.org English corpus) with power law normalisation. In total, this model contains around 500 million parameters and can process a word in 2.2ms on a GPU with a custom version of Caffe [Jia, 2013].

The cropped word recognition results using this CNN is given in Section 5.6.3.1.

5.6.3 Results

This section describes the results of our pipeline on various problems. Section 5.6.3.1 investigates the performance of the text recognition CNN by looking at the isolated problem of cropped word recognition, and the performance of the entire end-to-end pipeline is evaluated in Section 5.6.3.2 for text spotting and Section 5.6.3.3 for text based image retrieval.

5.6.3.1 Cropped Word Recognition

We evaluate the accuracy of our text recognition model over a wide range of datasets and lexicon sizes. As in Chapter 4 we follow the standard evaluation protocol by Wang et al., 2011 and perform recognition on the words containing only alphanumeric characters and at least three characters.

The results are shown in Table 5.1, and highlight the exceptional performance of our deep CNN, DICT. Although we train on purely synthetic data, with no human annotation, our model obtains significant improvements on state-of-the-art accuracy across all standard datasets, outperforming the character classifier based recognition model of Chapter 4. On IC03-50, the recognition problem is largely solved with 98.7% accuracy – only 11 mistakes out of 860 test samples – and we significantly outperform the previous state-of-the-art [Bissacco et al., 2013a] on SVT-50 by 5% and IC13 by 3%. Compared to the ICDAR datasets, the SVT accuracy, without the constraints of a dataset-specific lexicon, is lower at 80.7%. This reflects the difficulty of the SVT dataset as image samples can be of very low quality, noisy, and with low contrast. The Synth90k dataset accuracy shows that our model really can recognise word samples consistently across the whole 90k dictionary.

Model Size Effects. To investigate the effect of model size and output dictionary size, we train some reduced capacity recognition CNNs.

The shallower CNN, dubbed DICTsmall, has the 4th convolutional layer and one of the 4k fully connected layers removed. The DICTsmall network has a greatly reduced accuracy, achieving 90.3% on Synth90k (-5% compared to DICT), 90.0% on IC03 (-3%), and 73.0% on SVT (-8%). We also define a recognition CNN which, instead of modelling the 90k words of 90kDict, models only the test words that appear in IC03 and SVT, denoted as DICTsmall-IC03-Full and DICTsmall-SVT-Full respectively. These models have significantly less words to model (563 for

DICTsmall-IC03-Full and 4282 for DICTsmall-SVT-Full) and so define a simpler recognition problem. As expected, these constrained models perform significantly better than when trained on the entire 90k word dictionary. DICTsmall-IC03-Full reaches 98.1% accuracy on IC03 and when constraining to the 50 word lexicon, IC03-50, gets 99.2% accuracy, only misreading 7 out of the 860 test words. On SVT, the DICTsmall-SVT-Full model gets 87.0%, and with the constrained 50 word lexicon, SVT-50, achieves 96.1% accuracy.

Synthetic Data Effects. As an additional experiment, we look into the contribution that the various stages of the synthetic data generation engine in Section 3.3 make to real-world recognition accuracy. For speed of computation we do this experiment on the reduced output models DICTsmall-IC03-Full and DICTsmall-SVT-Full, which are tested only on their respective datasets. We repeatedly train these models from scratch, with the same training procedure, but with increasing levels of sophistication of synthetic data. Figure 5.8 shows how the test accuracy of these models increases as more sophisticated synthetic training data is used. The addition of random image-layer colouring causes a notable increase in performance (+44% on IC03 and +40% on SVT), as does the addition of natural image blending (+1% on IC03 and +6% on SVT). It is interesting to observe a much larger increase in accuracy through incorporating natural image blending on the SVT dataset compared to the IC03 dataset. This is most likely due to the fact that there are more varied and complex backgrounds to text in SVT compared to in IC03.

5.6.3.2 Text Spotting

In the text spotting task, the goal is to localise and recognise the words in the test images. Unless otherwise stated, we follow the standard evaluation protocol by Wang et al., 2011 as in Chapter 4 and ignore all words that contain alphanumeric characters and are not at least three characters long. A positive recognition result

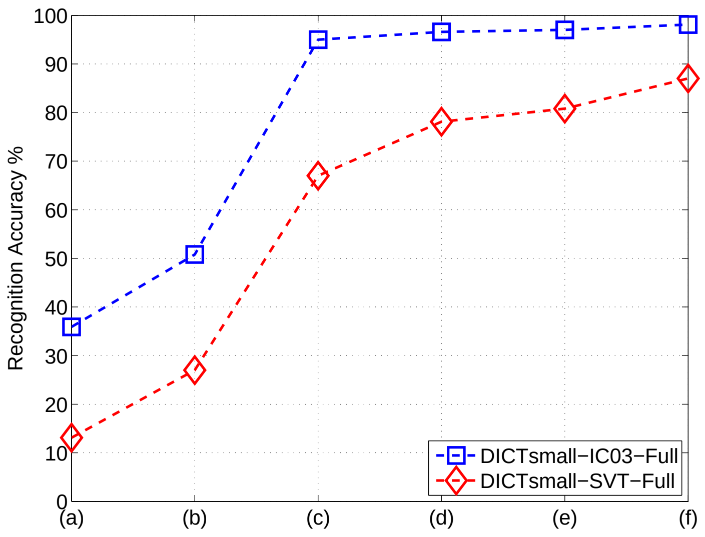
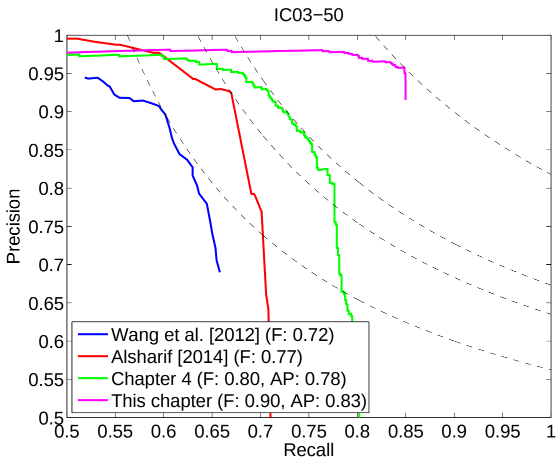
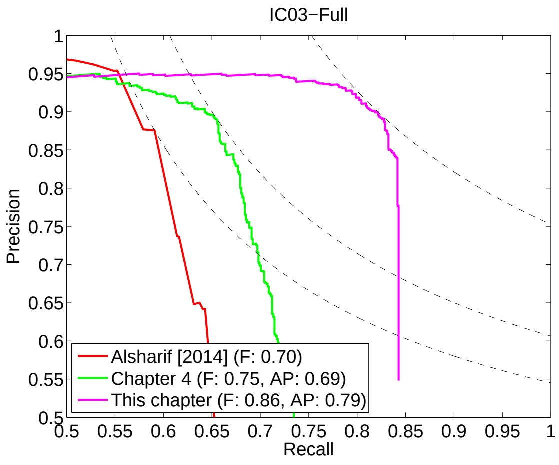


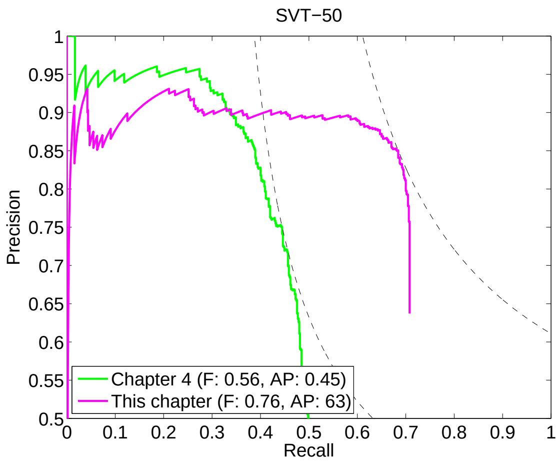
Figure 5.8: The recognition accuracies of text recognition models trained on just the IC03 lexicon (DICTsmall-IC03-Full) and just the SVT lexicon (DICTsmall-SVT-Full), evaluated on IC03 and SVT respectively. The models are trained on purely synthetic data with increasing levels of sophistication of the synthetic data. (a) Black text rendered on a white background with a single font, Droid Sans. (b) Incorporating all of Google fonts. (c) Adding background, foreground, and border colouring. (d) Adding perspective distortions. (e) Adding noise, blur and elastic distortions. (f) Adding natural image blending – this gives an additional 6.2% accuracy on SVT. The final accuracies on IC03 and SVT are 98.1% and 87.0% respectively.



(a)



(b)



(c)

Figure 5.9: The precision/recall curves on (a) the IC03-50 dataset, (b) the IC03-Full dataset, and (c) the SVT-50 dataset. The lines of constant F-measure are shown at the maximum F-measure point of each curve. The results from Wang et al., 2012; Alsharif and Pineau, 2014 were extracted from the papers.

	Cropped Word Recognition Accuracy (%)								
Model	Synth90k	IC03-50	IC03-Full	IC03	SVT-50	SVT	IC13	IIIT5k-50	IIIT5k-1k
ABBYY	-	56.0	55.0	-	35.0	-	-	24.3	-
Wang et al., 2011	-	76.0	62.0	-	57.0	-	-	-	-
Mishra et al., 2012	-	81.8	67.8	-	73.2	-	-	64.1	57.5
Novikova et al., 2012	-	82.8	-	-	72.9	-	-	-	-
Wang et al., 2012	-	90.0	84.0	-	70.0	-	-	-	-
Goel et al., 2013	-	89.7	-	-	77.3	-	-	-	-
Bissacco et al., 2013a	-	-	-	-	90.4	78.0	87.6	-	-
Alsharif and Pineau, 2014	-	93.1	88.6	85.1*	74.3	-	-	-	-
Almazán et al., 2014	-	-	-	-	89.2	-	-	91.2	82.1
Yao et al., 2014	-	88.5	80.3	-	75.9	-	-	80.2	69.3
Chapter 4	-	96.2	91.5	-	86.1	-	-	-	-
Gordo, 2014	-	-	-	-	90.7	-	-	93.3	86.6
This chapter	**95.2**	**98.7**	**98.6**	**93.1**	**95.4**	**80.7**	**90.8**	**97.1**	**92.7**

Table 5.1: Comparison to previous methods for text recognition accuracy – where the groundtruth cropped word image is given as input. The ICDAR 2013 results given are case-insensitive. Bold results outperform previous state-of-the-art methods. The baseline method is from a commercially available document OCR system, ABBYY, evaluated in [Wang et al., 2011; Yao et al., 2014]. *Recognition is constrained to a dictionary of 50k words.

is only valid if the detection bounding box has at least 0.5 overlap (IoU) with the groundtruth.

Table 5.2 shows the results of our text spotting pipeline compared to previous methods. We report the global F-measure over all images in the dataset. Across all datasets, our pipeline drastically outperforms all previous methods. On SVT-50, we increase the state-of-the-art by +20% to a P/R/F (precision/recall/F-measure) of 0.85/0.68/0.76 compared to 0.73/0.45/0.56 in Chapter 4. Similarly impressive improvements can be seen on IC03, where in all lexicon scenarios we improve F-measure by at least +10%, reaching a P/R/F of 0.96/0.85/0.90. Looking at the precision/recall curves in Figure 5.9, we can see that our pipeline manages to maintain very high recall, and the recognition score of our text recognition system is a strong cue to the suitability of a detection.

We also give results across all datasets when no lexicon is given. As expected, the F-measure suffers from the lack of lexicon constraints, though is still significantly higher than other comparable work. It should be noted that the SVT dataset is

	End-to-End Text Spotting (F-measure %)								
Model	IC03-50	IC03-Full	IC03	IC03$^+$	SVT-50	SVT	IC11	IC11$^+$	IC13
Neumann and Matas, 2011	-	-	-	41	-	-	-	-	-
Wang et al., 2011	68	51	-	-	38	-	-	-	-
Wang et al., 2012	72	67	-	-	46	-	-	-	-
Neumann and Matas, 2013	-	-	-	-	-	-	-	45	-
Alsharif and Pineau, 2014	77	70	63*	-	48	-	-	-	-
Chapter 4	80	75	-	-	56	-	-	-	-
This chapter	**90**	**86**	**78**	**72**	**76**	**53**	**76**	**69**	**76**
This chapter (0.3 IoU)	**91**	**87**	**79**	**73**	**82**	**57**	**77**	**70**	**77**

Table 5.2: Comparison to previous methods for end-to-end text spotting. Bold results outperform previous state-of-the-art methods. *Recognition is constrained to a dictionary of 50k words. $^+$Evaluation protocol described in Neumann and Matas, 2013.

	Text Based Image Retrieval					
Model	IC11 (mAP)	SVT (mAP)	STR (mAP)	Sports (mAP)	Sports (P@10)	Sports (P@20)
Wang et al., 2011*	-	21.3	-	-	-	-
Neumann and Matas, 2012*	-	23.3	-	-	-	-
Mishra et al., 2013	65.3	56.2	42.7	-	44.8	43.4
This chapter	**90.3**	**86.3**	**66.5**	**66.1**	**91.0**	**92.5**

Table 5.3: Comparison to previous methods for text based image retrieval. We report mean average precision (mAP) for IC11, SVT, STR, and Sports, and also report top-n retrieval to compute precision at n (P@n) on Sports. Bold results outperform previous state-of-the-art methods. *Experiments were performed in Mishra et al., 2013, not by the author of this thesis.

only partially annotated. This means that the precision (and therefore F-measure) is much lower than the true precision if fully annotated, since many words that are detected are not annotated and are therefore incorrectly recorded as false-positives. We can however report recall on SVT-50 and SVT of 71% and 59% respectively.

Interestingly, when the overlap threshold is reduced to 0.3 (last row of Table 5.2), we see a small improvement across ICDAR datasets and a large +8% improvement on SVT-50. This implies that our text recognition CNN is able to accurately recognise even loosely cropped detections. Ignoring the requirement of correctly recognising the words, *i.e.* performing purely word detection, we get an F-measure of 0.85 and 0.81 for IC03 and IC11.

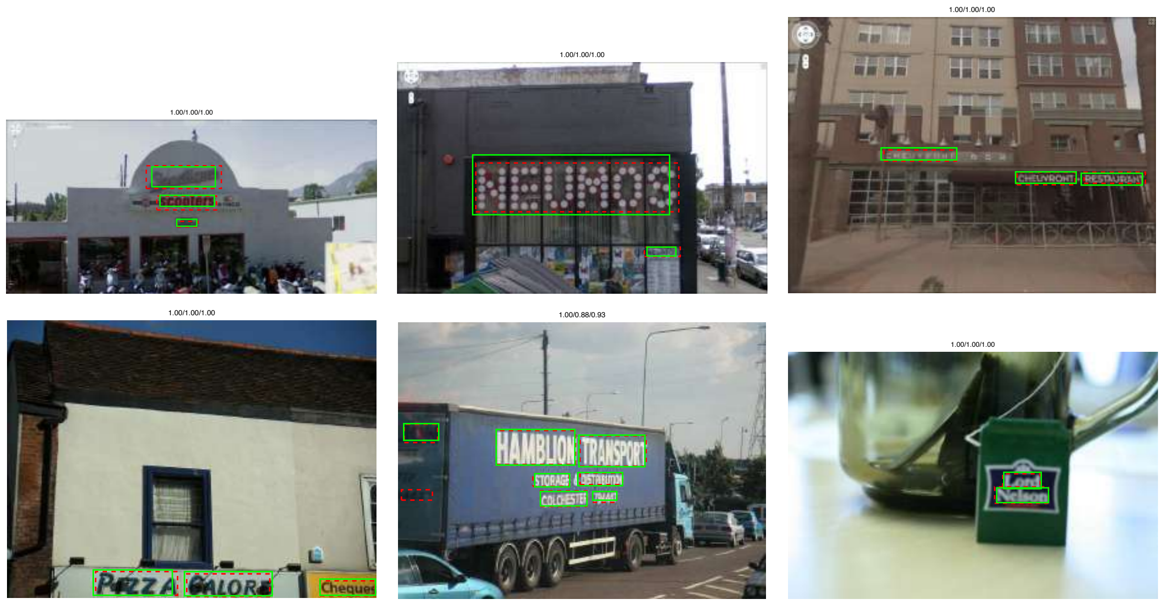


Figure 5.10: Some example text spotting results from SVT-50 (top row) and IC11 (bottom row). Red dashed shows groundtruth and green shows correctly localised and recognised results. P/R/F figures are given above each image.

Some example text spotting results are shown in Figure 5.10. Since our pipeline does not rely on connected component based algorithms or explicit character recognition, we can detect and recognise disjoint, occluded and blurry words.

A common failure mode of our system is the missing of words due to the lack of suitable proposal regions, especially apparent for slanted or vertical text, something which is not explicitly modelled in our framework. Also the detection of sub-words or multiple words together can cause false-positive results.

5.6.3.3 Image Retrieval

We also apply our pipeline to the task of text based image retrieval. Given a text query, the images containing the query text must be returned.

This task is evaluated using the framework of Mishra et al., 2013, with the results shown in Table 5.3. For each defined query, we retrieve a ranked list of all the images of the dataset and compute the average precision (AP) for each query, reporting the mean average precision (mAP) over all queries. We significantly outperform Mishra *et al.* across all datasets – we obtain an mAP on IC11 of 90.3%, compared to 65.3% from Mishra et al., 2013. Our method scales seamlessly to the larger Sports dataset, where our system achieves a precision at 20 images (P@20) of 92.5%, more than doubling that of 43.4% from Mishra et al., 2013. Mishra et al., 2013 also report retrieval results on SVT for released implementations of other text spotting algorithms. The method from Wang et al., 2011 achieves 21.3% mAP, the method from Neumann and Matas, 2012 achieves 23.3% mAP and the method proposed by Mishra et al., 2013 itself achieves 56.2% mAP, compared to our own result of 86.3% mAP.

However, as with the text spotting results for SVT, our retrieval results suffer from incomplete annotations on SVT and Sports datasets – Figure 5.11 shows how precision is hurt by this problem. The consequence is that the true mAP on SVT is higher than the reported mAP of 86.3%.

Depending on the image resolution, our algorithm takes approximately 5-20s to compute the end-to-end results per image on a single CPU core and single GPU. We analyse the time taken for each stage of our pipeline on the SVT dataset, which has an average image size of 1260×860, showing the results in Table 5.4. Since we reduce the number of proposals throughout the pipeline, we can allow the processing time per proposal to increase while keeping the total processing time for each stage stable. This affords us the use of more computationally complex features and classifiers as the pipeline progresses. Our method can be trivially parallelised, meaning we can process 1-2 images per second on a high-performance workstation with 16 physical CPU cores and 4 commodity GPUs.

The high precision and speed of our pipeline allows us to process huge datasets fro practical search applications. This is demonstrated on a 5000 hour BBC News dataset in Chapter 8.

Stage	# proposals	Time	Time/proposal
(a) Edge Boxes	$> 10^7$	2.2s	< 0.002ms
(b) ACF detector	$> 10^7$	2.1s	< 0.002ms
(c) RF filter	10^4	1.8s	0.18ms
(d) CNN regression	10^3	1.2s	1.2ms
(e) CNN recognition	10^3	2.2s	2.2ms
	Total	9.5s	

Table 5.4: The processing time for each stage of the pipeline evaluated on the SVT dataset on a single 3.2Ghz CPU core and single Nvidia GeForce GTX Titan GPU. As the pipeline progresses from (a)-(e), the number of proposals is reduced (starting from all possible bounding boxes), allowing us to increase our computational budget per proposal while keeping the overall processing time for each stage comparable.

Limitations. There are limitations to this system. At present there is no support for reading vertical or significantly rotated text, as this is not modelled in the detection phase, and during recognition the height resizing to 32px will cause rotated text to be unreadably distorted. Another issue is that there seems to be a number of false positive detections of short words (one, two, and three letter words). In general, there is no dataset level or image level context for the text spotting, meaning that it could be greatly beneficial to incorporate a word frequency model and even an image level language model, which could help give context to the individual detections within a single image and prevent these false positive results. Finally, by design, the recognition part of this system can only recognise words that are contained within the large dictionary supplied – an alternative text recognition model for out-of-dictionary recognition is explored in the next chapter.

5.7 Conclusions

In this chapter we have presented an end-to-end text reading pipeline – a detection and recognition system for text in natural scene images. By taking a word-centric approach, this general system works remarkably well for both text spotting and image retrieval tasks, significantly improving the performance on both tasks over

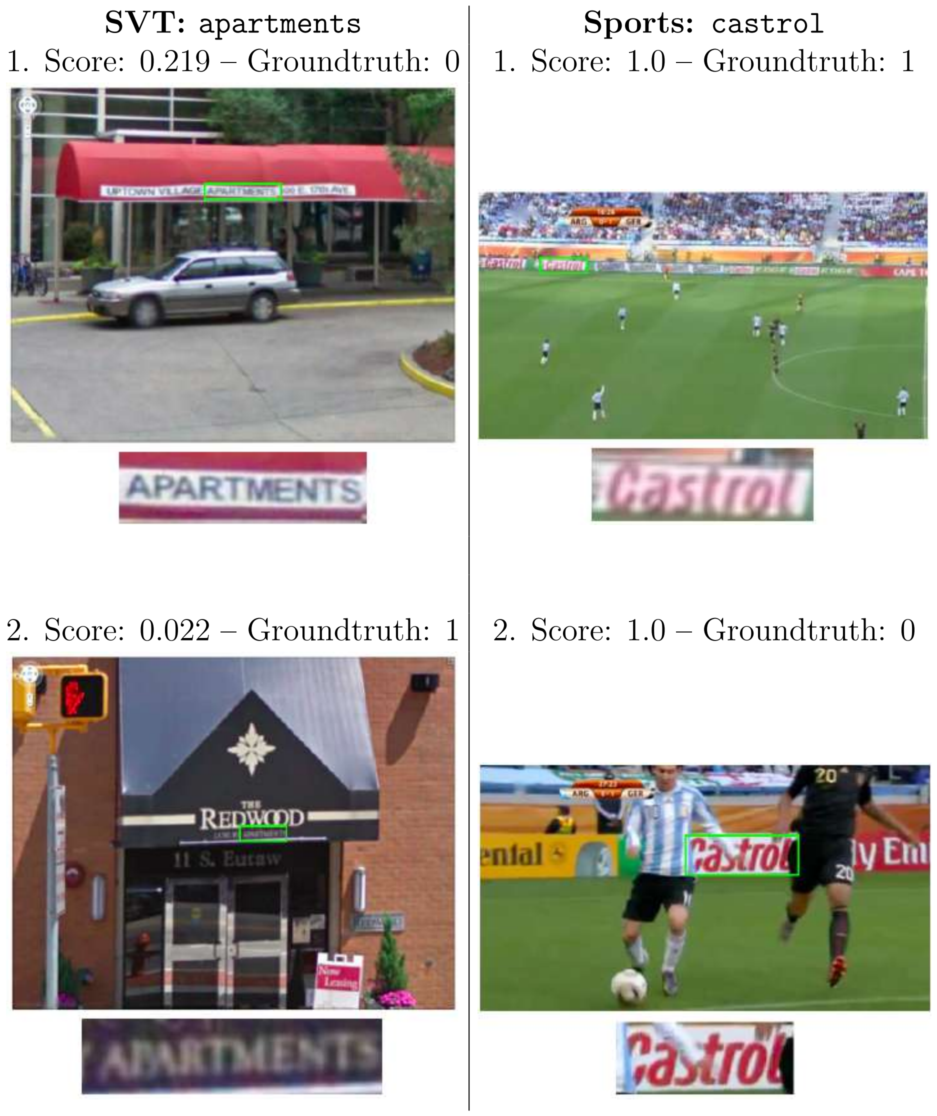

SVT: apartments
1. Score: 0.219 – Groundtruth: 0
APARTMENTS
2. Score: 0.022 – Groundtruth: 1
APARTMENTS
Sports: castrol
1. Score: 1.0 – Groundtruth: 1
Castrol
2. Score: 1.0 – Groundtruth: 0
Castrol

Figure 5.11: An illustration of the problems with incomplete annotations in test datasets. We show examples of the top two results for the query `apartments` on the SVT dataset and the query `castrol` on the Sports dataset. All retrieved images contain the query word (green box), but some of the results have incomplete annotations, and so although the query word is present the result is labelled as incorrect. This leads to a reported AP of 0.5 instead of 1.0 for the SVT query, and a reported P@2 of 0.5 instead of the true P@2 of 1.0.

all previous methods, on all standard datasets, without any dataset-specific tuning. This is largely down to a very high recall proposal stage and a text recognition model that achieves superior accuracy to all previous systems. Our system is fast and scalable – we demonstrate seamless scalability from datasets of hundreds of images to being able to process datasets of millions of images for instant text based image retrieval without any perceivable degradation in accuracy. Additionally, the ability of our recognition model to be trained purely on synthetic data allows our system to be easily re-trained for recognition of other languages or scripts, without any human labelling effort.

Chapter 6

Unconstrained Text Recognition

The text spotting systems described in Chapter 4 and Chapter 5 have recognition stages that both assume the presence of a dictionary or fixed lexicon of words that can be recognised. While this dictionary can be extremely large (90k words in Chapter 5) this still imposes a limitation on the expressivity and ability to generalise to any given text. To push past this boundary, in this chapter we tackle the problem of *unconstrained text recognition* – recognising text in natural images without restricting the words to a fixed lexicon or dictionary.

Since this is predominantly a recognition problem, this chapter focuses only on text recognition, assuming word bounding boxes have been found already, allowing it to be used with the detection stages of the previous two chapters.

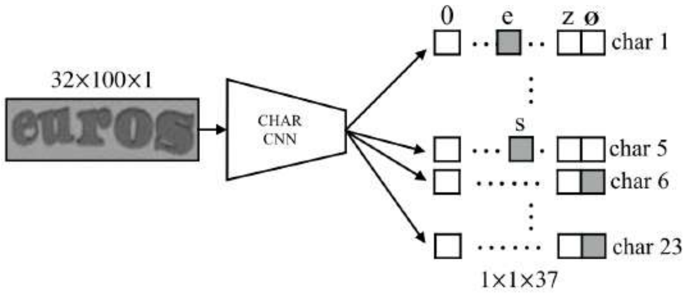
Our unconstrained text recognition model is based on deep convolutional neural networks, but unlike previous work which constrains or heavily weights results to be from a dictionary of known words [Alsharif and Pineau, 2014; Bissacco et al., 2013a; Gordo, 2014], we make no such assumptions, allowing our model to generalise to where previously unseen or non-language based text must be recognised – for example for generic alpha-numeric strings such as number plates or phone numbers.

The shift of focus towards a model which performs accurately without a fixed dictionary significantly increases the complexity of the text recognition problem. To solve this, we propose a novel CNN architecture (Figure 6.2) employing a *Conditional Random Field* (CRF) whose unary terms are outputs of a CNN character predictor, which are position-dependent, and whose higher order terms are outputs of a CNN N-gram predictor, which are position-independent. The recognition result is then obtained by finding the character sequence that maximises the CRF score, enforcing the consistency of the individual predictions.

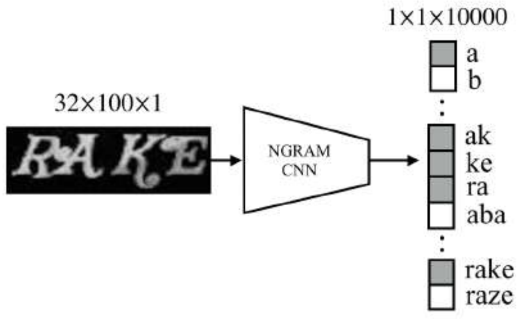
The two CNN components – the character sequence predictor CNN and the N-gram detection CNN – can be trained independently for word recognition. However, when combined in the joint CRF model the entire network can be jointly trained by defining a structured output learning problem, and back-propagating the corresponding *structured output loss*. This joint formulation results in multi-task learning of both the character and N-gram predictors, and additionally learns how to combine their representations in the CRF, resulting in more accurate text recognition than either sub-CNN alone.

The result is a highly flexible text recognition system that achieves excellent unconstrained text recognition performance as well as state-of-the-art recognition performance when using standard dictionary constraints. While performance is measured on real images from standard text recognition benchmarks, as with the recognition CNN of Chapter 5, all results are obtained by training the model purely on the synthetic data described in Section 3.3. In addition to real-world data tests, the model is evaluated on this synthetic data in order to study its performance under different scenarios.

We begin by introducing the character sequence model in Section 6.1.1 and the N-gram detection model in Section 6.1.2. Section 6.2 shows how these two predictors can be combined to form a joint CRF model and formulates the training of the latter as structured-output learning. Section 6.3 evaluates these models extensively and



(a)



(b)

Figure 6.1: (a) The character sequence model. A word image is recognised by predicting the character at each position in the output, spelling out the text character by character. Each positional classifier is learnt independently but shares a jointly optimised set of features. (b) The N-gram detection model. The recognised text is represented by its bag-of-N-grams. This can be thought of as 10k independently trained binary classifiers using a shared set of jointly learnt features, trained to detect the presence of a particular N-gram.

Section 6.4 summarises the findings of this chapter.

6.1 CNN Text Recognition Models

We now describe the component CNN models that form the basis of our joint model in Section 6.2. Both models use the same concept as the dictionary classification model in Section 5.4.1 of taking the entire word image as input to a CNN, but define different output encoding schemes.

6.1.1 Character Sequence Model

In this section we describe our character sequence model. This model encodes the character at each position in the word and so predicts the sequence of characters in an image region (hereafter we simply refer to the image region as an image). Each position in the word is modelled by an independent classifier acting on a shared set of features from a single CNN. By construction, this model makes no assumptions about the underlying language and allows completely unconstrained recognition.

A word w of length N is modelled as a sequence of characters such that $w = (c_1, c_2, \ldots, c_N)$ where each $c_i \in \mathcal{C} = \{1, 2, \ldots, 36\}$ represents a character at position i in the word, from the set of 10 digits and 26 letters. Each c_i can be predicted with a single classifier, one for each character in the word. However, since words have variable length N which is unknown at test time, we fix the number of characters to $N_{\max}$ (here set to 23), the maximum length of a word in the training set, and introduce a null character class. Therefore a word is represented by a string $w \in (\mathcal{C} \cup \{\phi\})^{N_{\max}}$.

For a given input image $\mathtt{x}$, we want to return the estimated word w^* which maximises $P(w^*|\mathtt{x})$. Since we seek an unconstrained recognition system with this model, we assume independence between characters leading to

$$w^* = \arg\max_{w} P(w|\mathtt{x}) = \arg \max_{c_1, c_2, \ldots, c_{N_{\max}}} \prod_{i=1}^{N_{\max}} P(c_i|\Phi(\mathtt{x})) \tag{6.1}$$

where $P(c_i|\Phi(\mathtt{x}))$ is given by the classifier for the i-th position acting on a single set of shared CNN features $\Phi(\mathtt{x})$. The word w^* can be computed by taking the most probable character at each position $c_i^* = \arg\max_{c_i \in \mathcal{C} \cup \{\phi\}} P(c_i|\Phi(\mathtt{x}))$.

The CNN (Figure 6.1 (a)) takes the whole word image $\mathtt{x}$ as input. Word images can be of different sizes, in particular due to the variable number of characters in the image. However, our CNN requires a fixed size input for all input images. This problem is overcome by simply resampling the original word image to a canonical height and width, without regard to preserving the aspect ratio, producing a fixed size input $\mathtt{x}$.

The base CNN has a number of convolutional layers followed by a series of fully connected layers, giving $\Phi(\mathtt{x})$. The full details of the network architecture are given in Section 6.3.2. $\Phi(\mathtt{x})$ is fed to $N_{\max}$ separate fully connected layers (one layer per character position) with 37 neurons each, one for each character class including the null character. These fully connected layers are independently softmax normalised and can be interpreted as the probabilities $P(c_i|\Phi(\mathtt{x}))$ of the width-resized input image $\mathtt{x}$.

The CNN is trained with multinomial logistic regression loss applied to each character position classifier, with the back-propagated gradients from each classifier accumulating on the penultimate fully connected layer, allowing the entire network to be trained through back-propagation and optimised with stochastic gradient descent (SGD) and dropout regularisation similar to Hinton et al., 2012.

6.1.2 N-gram Detection Model

This section describes our second word recognition model, which exploits compositionality to represent words. In contrast to the sequential character encoding of Section 6.1.1, words can be seen as a composition of an unordered set of character N-grams, a *bag-of-N-grams*. In the following, if $s \in \mathcal{C}^N$ and $w \in \mathcal{C}^M$ are two strings, the symbol $s \subset w$ indicates that s is a substring of w. An N-gram of word w is a substring $s \subset w$ of length $|s| = N$. We will denote with $\mathcal{G}_N(w) = \{s : s \subset w \wedge |s| \leq N\}$ the set of all N-grams of word w of length up to N and with $G_N = \cup_{w \in \mathcal{W}} \mathcal{G}_N(w)$ the set of all such grams in the language. For example,

$$\mathcal{G}_3(\mathtt{spires}) = \{\mathtt{s}, \mathtt{p}, \mathtt{i}, \mathtt{r}, \mathtt{e}, \mathtt{sp}, \mathtt{pi}, \mathtt{ir}, \mathtt{re}, \mathtt{es}, \mathtt{spi}, \mathtt{pir}, \mathtt{ire}, \mathtt{res}\}. \tag{6.2}$$

This method of encoding variable length sequences is similar to the *Wickelphone* phoneme-encoding methods [Wickelgran, 1969].

Even for small values of N, $\mathcal{G}_N(w)$ encodes each word $w \in \mathcal{W}$ nearly uniquely. For example, with $N = 4$, this map has only 7 collisions out of a dictionary of 90k English words. The encoding $\mathcal{G}_N(w)$ can be represented as a $|\mathcal{G}_N|$-dimensional binary vector of N-gram occurrences. This vector is very sparse, as on average $|\mathcal{G}_N(w)| \approx 22$ for an English word, whereas $|\mathcal{G}_N| = 10k$.

Using a CNN we can predict $\mathcal{G}_N(w)$ for a word w depicted in the input image $\mathtt{x}$. We can use the same architecture as in Section 6.1.1, but now have a final fully connected layer with $\mathcal{G}_N$ neurons to represent the encoding vector. The scores from the fully connected layer can be interpreted as probabilities of an N-gram being present in the image by applying the logistic function to each neuron. The CNN is therefore learning to recognise the presence of each N-gram somewhere within the input image, so is an N-gram detector.

With the applied logistic function, the training problem becomes that of $|\mathcal{G}_N|$ separate binary classification tasks, and so we back-propagate the logistic regression loss with respect to each N-gram class independently. To jointly train a whole range of N-grams, some of which occur very frequently and some barely at all, we have to scale the gradients for each N-gram class by the inverse frequency of their appearance in the training word corpus. We also experimented with hinge loss and simple regression to train but found frequency weighted binary logistic regression was superior. As with the other model, we use dropout and SGD.

In this model we exploit the statistics of our underlying language in choosing a subset of $|\mathcal{G}_N|$ N-grams from the space of all possible N-grams to be modelled. This can be seen as using a language model to compress the representation space of the encoding, but is not restraining the predictive capability for unconstrained recognition. While the encoding $\mathcal{G}_N(w)$ is almost always unique for words from

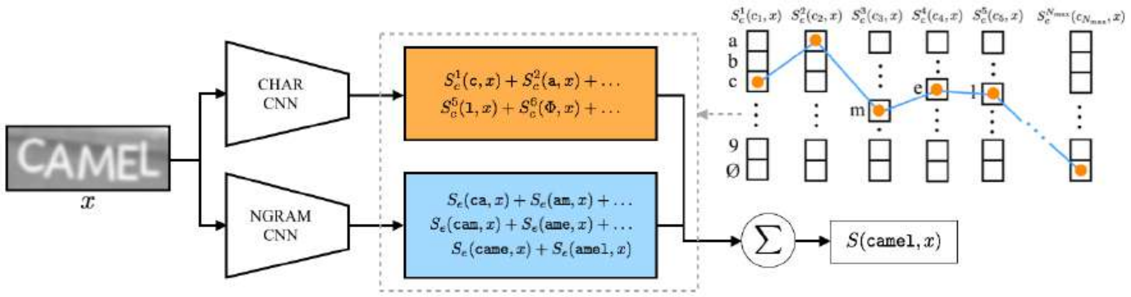

CAMEL
x
CHAR CNN
NGRAM CNN
$S_c^1(\mathtt{c}, x) + S_c^2(\mathtt{a}, x) + \ldots$
$S_c^5(\mathtt{l}, x) + S_c^6(\Phi, x) + \ldots$
$S_e(\mathtt{ca}, x) + S_e(\mathtt{am}, x) + \ldots$
$S_e(\mathtt{cam}, x) + S_e(\mathtt{ame}, x) + \ldots$
$S_e(\mathtt{came}, x) + S_e(\mathtt{amel}, x)$
Σ
$S(\mathtt{camel}, x)$
$S_c^1(c_1, x)$ $S_c^2(c_2, x)$ $S_c^3(c_3, x)$ $S_c^4(c_4, x)$ $S_c^5(c_5, x)$ $S_c^{N_{\max}}(c_{N_{\max}}, x)$
a
b
c
m
e
l
9
Ø

Figure 6.2: An illustration of the construction of the path score $S(\mathtt{camel}, \mathtt{x})$ for the word `camel`. The unary and edge terms used for the score are selected by the path through the graph of character positions shown in the upper right corner. The values of these terms, $S_c(c_i, \mathtt{x})$ and $S_e(s, \mathtt{x})$, where $s \subset w$, are given by the outputs of the character sequence model CNN (CHAR CNN, Section 6.1.1) and the N-gram encoding CNN (NGRAM CNN, Section 6.1.2).

natural language, non-language words often contain much fewer N-grams from the modelled set $\mathcal{G}_N$ leading to more ambiguous and non-unique encodings.

6.2 Joint Model

In Section 6.1.1, maximising the posterior probability of a character sequence (Equation 6.1) is equivalent to maximising the log-score

$$\log P(w|\mathtt{x}) = S(w, \mathtt{x}) = \sum_{i=1}^{N_{\max}} S_c^i(c_i, \mathtt{x}) \tag{6.3}$$

where $S_c^i(c_i, \mathtt{x}) = \log P(c_i|\Phi(\mathtt{x}))$ is the logarithm of the posterior probability of the character at position i in the sequence. The graph associated with this function is a set of nodes, one for each unary term $S_c^i(c_i, \mathtt{x})$, and does not contain any edges. Hence maximising the function reduces to maximising each term individually.

The model can now be extended to incorporate the N-gram predictors of Section 6.1.2, encoding the presence of N-grams in the word image $\mathtt{x}$. The N-gram scoring function $S_e(s, \mathtt{x})$ assigns a score to each string s of length $|s| \leq N$, where N is the maximum order of N-gram modelled. Note that, differently from the functions

S_c^i defined before, the function S_e is position-independent. However, it is applied repeatedly at each position i in the word:

$$S(w, \mathtt{x}) = \sum_{i=1}^{N_{\max}} S_c^i(c_i, \mathtt{x}) + \sum_{i=1}^{|w|} \sum_{n=1}^{\min(N,|w|-i+1)} S_e(c_i c_{i+1} \ldots c_{i+n-1}, \mathtt{x}). \tag{6.4}$$

As illustrated in Figure 6.2, the scores $S_c^i(c_i, \mathtt{x})$ are obtained from the CNN character predictors of Section 6.1.1 whereas the score $S_e(s, \mathtt{x})$ is obtained from the CNN N-gram predictor of Section 6.1.2; note that the N-gram scoring function is only defined for the subset $\mathcal{G}_N$ of N-grams modelled in the CNN; if $s \notin \mathcal{G}_N$, the score $S_e(s, \mathtt{x}) = 0$ is defined to be zero.

The graph associated with the function Equation 6.4 has cliques of order N; hence, when N is even moderately large, we resort to beam search [Russel, Norvig, et al., 1994] to maximise Equation 6.4 and find the predicted word w^*. Also, the score of Equation 6.4 can be interpreted as a potential function defining a word posterior probability as before; however, evaluating this probability would require computing a normalisation factor, which is non-trivial. Instead, the function is trained discriminatively, as explained in the next section.

6.2.1 Structured Output Loss

The unary and edge score functions $S_c^i(c_i, \mathtt{x})$ and $S_e(s, \mathtt{x})$, should incorporate the outputs of the character sequence model and N-gram encoding model respectively. A simple way to do this is to apply a weighting to the output of the CNNs after removing the softmax normalisation:

$$S(w, \mathtt{x}) = \sum_{i=1}^{N_{\max}} \alpha_{c_i}^i f_{c_i}^i(\mathtt{x}) + \sum_{i=1}^{|w|} \sum_{n=1}^{\min(N,|w|-i+1)} \beta_{c_i c_{i+1} \ldots c_{i+n-1}} g_{c_i c_{i+1} \ldots c_{i+n-1}}(\mathtt{x}), \tag{6.5}$$

where $f_{c_i}^i(\mathtt{x})$ is the output of the character sequence CNN for character c_i at position i and $g_s(\mathtt{x})$ is the output of the N-gram encoding CNN for the N-gram s. If desired, the character weights $\alpha = \{\alpha^i_{c_i}\}$ and edge weights $\beta = \{\beta_s\}$ can be constrained to be shared across different characters, character positions, different N-grams of the same order, or across all N-grams.

The sets of weights α and β in Equation 6.5, or any weight-constrained variant of Equation 6.5, can be learnt in a structured output learning framework, encouraging the score of the ground-truth word w_{gt} to be greater than or equal to the highest scoring *incorrect* word prediction plus a margin, *i.e.* $S(w_{\text{gt}}, \mathtt{x}) \geq \mu + S(w^*, \mathtt{x})$ where $S(w^*, \mathtt{x}) = \max_{w \neq w_{\text{gt}}} S(w, \mathtt{x})$. Enforcing this as a soft-constraint results in the convex loss

$$L(\mathtt{x}_i, w_{\text{gt},i}, S) = \max_{w \neq w_{\text{gt},i}} \max(0, \mu + S(w, \mathtt{x}) - S(w_{\text{gt},i}, \mathtt{x}_i)) \tag{6.6}$$

and averaging over M example pairs $(\mathtt{x}_i, w_{\text{gt},i})$ results in the regularised empirical risk objective

$$E(S) = \frac{\lambda_\alpha}{2}\|\alpha\|^2 + \frac{\lambda_\beta}{2}\|\beta\|^2 + \frac{1}{M}\sum_{i=1}^{M} L(\mathtt{x}_i, w_{\text{gt},i}, S). \tag{6.7}$$

However, in the general scenario of Equation 6.5, the weights can be incorporated into the CNN functions f and g, resulting in the score

$$S(w, \mathtt{x}) = \sum_{i=1}^{N_{\max}} f^i_{c_i}(\mathtt{x}) + \sum_{i=1}^{|w|} \sum_{n=1}^{\min(N,|w|-i+1)} g_{c_i c_{i+1} \dots c_{i+n-1}}(\mathtt{x}), \tag{6.8}$$

The functions f and g are defined by CNNs and so we can optimise the parameters of them to reduce the cost in Equation 6.7. This can be done through standard back-propagation and SGD. Differentiating the loss L with respect to S gives

$$\frac{\partial L(\mathtt{x}, w_{\text{gt}}, S)}{\partial S(w^*, \mathtt{x})} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases} \qquad \frac{\partial L(\mathtt{x}, w_{\text{gt}}, S)}{\partial S(w_{\text{gt}}, \mathtt{x})} = \begin{cases} -1 & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6.9}$$

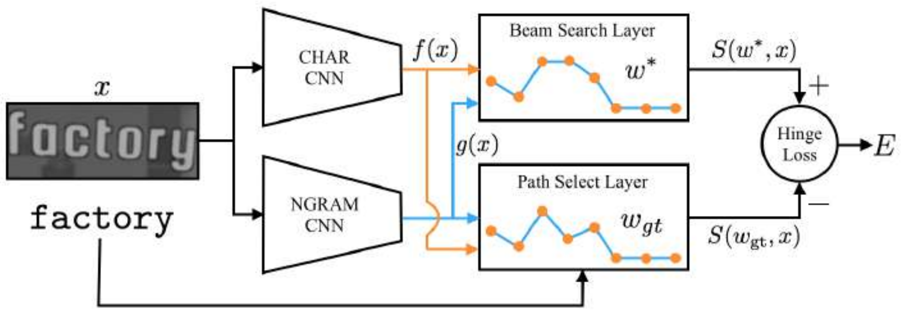


Figure 6.3: The architecture for training the joint model, comprising of the character sequence model (CHAR) and and the N-gram encoding model (NGRAM) with structured output loss. The Path Select Layer generates the score $S(w_{\text{gt}}, x)$ by summing the inputs of the groundtruth word. The Beam Search Layer uses beam search to try to select the path with the largest score $S(w^*, x)$ from the inputs. The hinge loss implements a ranking loss, constraining the highest scoring path to be the groundtruth path, and can be back-propagated through the entire network to jointly learn all the parameters.

where $z = \max_{w \neq w_{\text{gt},i}} \mu + S(w, \mathtt{x}) - S(w_{\text{gt}}, \mathtt{x})$. Differentiating the score function of Equation 6.8 with respect to the character sequence model and N-gram encoding model outputs $f^i_{c_i}$ and g_s gives

$$\frac{\partial S(w, \mathtt{x})}{\partial f^i_c} = \begin{cases} 1 & \text{if } c_i = c \\ 0 & \text{otherwise} \end{cases}, \qquad \frac{\partial S(w, \mathtt{x})}{\partial g_s} = \sum_{i=1}^{|w|-|s|+1} \mathbf{1}_{\{c_i c_{i+1} \ldots c_{i+|s|-1} = s\}} \tag{6.10}$$

This allows errors to be back-propagated to the entire network. Intuitively, the errors are back-propagated through the CNN outputs which are responsible for margin violations, since they contributed to form an incorrect score.

Using this structured output loss allows the parameters of the entire model to be jointly optimised within the structure imposed by Equation 6.8. Figure 6.3 shows the training architecture used. Due to the presence of high order scores in Equation 6.8, it is too expensive to exhaustively search the space of all possible paths to find w^*, even with dynamic programming, so instead we use beam search to find the approximate highest scoring path.

The structured output loss described in this section bares resemblance to the discriminative Viterbi training introduced by LeCun et al., 1998. However, our model includes higher-order terms, terms of a different nature (N-grams), and uses a structured-output formulation. Furthermore, our method incorporates only a very weak language model, limited to assigning a score of 0 to all N-grams outside a target set $\mathcal{G}_N$. Note that this does not mean that these N-grams cannot be recognised (this would require assigning to them a score of $-\infty$); instead it is a smoothing technique that assigns a nominal score to infrequent N-grams.

6.3 Evaluation

In this section we evaluate the three models introduced in the previous sections. The datasets used for training and testing are given in Section 6.3.1, the implementation details given in Section 6.3.2, and the results of experiments reported in Section 6.3.3.

6.3.1 Datasets

We evaluate our models on a number of real-world standard datasets – ICDAR 2003, ICDAR 2013, Street View Text, and IIIT5k. Training of the models is performed purely using the synthetic data data from Section 3.3. We also use the synthetic data, Synth90k and SynthRand, to test our models across a larger vocabulary and also completely random words. Full descriptions of the datasets can be found in Chapter 3 and summarised in Table 3.1.

6.3.2 Implementation Details

In the following, the character sequence model is referred to as CHAR, the N-gram encoding model as NGRAM, and the joint model as JOINT.

The CHAR and NGRAM models both have the same base CNN architecture. The base CNN has five convolutional layers and two fully connected layers. The input is a 32×100 greyscale image obtained by resizing the word image (ignoring its aspect ratio) and then subtracting its mean and dividing by its standard deviation. Rectified linear units are used throughout after each weight layer except for the last one. In forward order, the convolutional layers have 64, 128, 256, 512, and 512 square filters with an edge size of 5, 5, 3, 3, and 3. Convolutions are performed with stride 1 and there is input feature map padding to preserve spatial dimensionality. 2×2 max-pooling follows the first, second and third convolutional layers. The fully connected layers have 4096 units.

On top of this base CNN, the CHAR model has 23 independent fully connected layers with 37 units, allowing recognition of words of up to $N_{\text{max}} = 23$ characters long. The NGRAM model operates on a selection of 10k frequent N-grams of order $N \leq 4$ (identified as the ones that occur at least 10 times in the Synth90k word corpus, resulting in 36 1-grams, 522 2-grams, 3965 3-grams, and 5477 4-grams). This requires a final fully connected layer on top of the base CNN with 10k units. Therefore, the graph of function Equation 6.8 has cliques of sizes at most 4. Beam search uses a width of 5 during training and of 10 during testing. If a lexicon is used to constrain the output, instead of performing beam search, the paths associated with the lexicon words are scored with Equation 6.8, and the word with the maximum score is selected as the final result.

The three models are all trained with SGD and dropout regularisation. The learning rates are dynamically decreased as training progresses. The JOINT model is initialised with the pre-trained CHAR and NGRAM network weights and the convolutional layers' weights are frozen during training.

6.3.3 Experiments

We evaluate our models on a combination of real-world test data and synthetic data to highlight different operating characteristics.

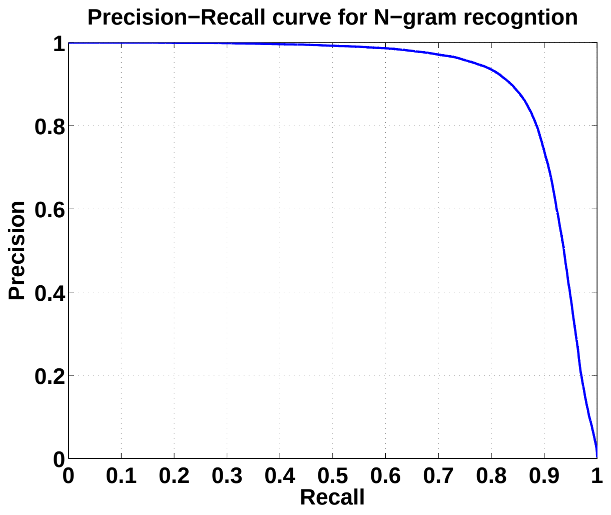


Figure 6.4: The precision-recall curve for N-gram detection by the NGRAM model, generated by varying the probability threshold on the Synth90k dataset. Maximum F-score is 0.87.

6.3.3.1 N-gram Detection Model

The NGRAM model predicts the N-grams contained in input word image. Due to the highly unbalanced nature of this problem (where only 10-20 N-grams are contained in any given image), results are reported as the maximum achieved F-score on the test dataset, computed as the harmonic mean of precision and recall. Precision and recall is computed at different operating points by varying the threshold probability for an N-gram to be classified as present in the word. The maximum achieved F-score on Synth90k is 87.0% and on IC03 is 87.1%. The precision/recall curve for Synth90k is shown in Figure 6.4.

This NGRAM model can in fact be used for constrained text recognition by itself. Since each N-gram detection vector represents a word, we can predict the N-gram detection vector with the CNN and then find the dictionary word which is most closely represented by the predicted vector. A simple way to achieve this is by finding the nearest neighbour (in Euclidean space) between the predicted N-gram vector and the groundtruth N-gram vectors of a dictionary of words (where

Training Data	Test Data	CHAR	JOINT
Synth90k	Synth90k	87.3	**91.0**
	Synth72k-90k	87.3	-
	Synth45k-90k	87.3	-
	IC03	85.9	**89.6**
	SVT	68.0	**71.7**
	IC13	79.5	**81.8**
Synth1-72k	Synth72k-90k	82.4	**89.7**
Synth1-45k	Synth45k-90k	80.3	**89.1**
SynthRand	SynthRand	**80.7**	79.5

Table 6.1: The accuracy (%) of the character sequence model, CHAR, and the joint model, JOINT. Different combinations of training and test data are evaluated. Synth$x$-y refers to a subset of the Synth90k that only contains words in the label interval $[x, y]$ (word label indicies are in random, non-alphabetical order). Training and testing on completely distinct words demonstrates the power of a general, unconstrained recognition model.

an element is 1 if the N-gram is contained in the word and 0 if not). Doing this for the ICDAR 2003 test dataset constrained to a lexicon of 50 words per image (IC03-50, Section 3.1.1) results in a recognition accuracy of 94.2%. This extremely naive method still gives competitive performance, illustrating the discriminative nature of N-grams for word recognition. Instead, one could learn a linear SVM mapping from N-gram encoding to dictionary words, allowing for scalable word recognition through an inverted index of these mappings. Training an SVM for each dictionary word in the ICDAR 2003 test set results in 96.5% accuracy on IC03-50. This is impressive compared to the 98.7% accuracy of the dictionary classification model in Chapter 5, considering the CNN is not trained explicitly for word classification.

The performance of the N-gram detector, both in terms of F-score and on the constrained recognition tests, illustrates the predictive capability of the NGRAM CNN alone.

6.3.3.2 Character Sequence Model and Joint Model

We now turn our attention to the character sequence model, CHAR, and the jointly trained formulation with the N-gram detection model incorporated as well, JOINT.

The CHAR and JOINT models are evaluated on standard as well as synthetic benchmarks (Table 6.1). When trained on the Synth90k data, the CHAR model achieves good performance, with an accuracy of 85.9% on IC03 and 87.3% on the huge Synth90k test dataset. However, the JOINT model consistently outperforms the lone CHAR model, with an accuracy improvement of as much as +4% on IC03, Synth90k and SVT. Figure 6.6 shows some example results using the JOINT model.

To investigate the effect of model size on the accuracy of the CHAR model, we train a reduced capacity model with the 4th convolutional layer and a fully connected layer removed from the base CNN architecture. This shallower model performs significantly worse, with 77.0% accuracy on IC03 (-9%) and 56.4% accuracy on SVT (-12%). This clearly illustrates the importance of depth in CNN architectures, something reinforced in the recognition CNN of Chapter 5 and the work of Simonyan and Zisserman, 2015.

Next, we evaluate the ability of our model to generalise by recognising words unseen during training. This effectively amounts to zero-shot learning and is the key contribution of this chapter over previous word recognition systems. In order to evaluate this, the training vocabulary (90kDict, Section 3.3) is split into two parts, with one part (50% or 80%) used for training and the other one for evaluation (50% or 20%). This means that the models are tested solely on words that have never been seen before during training. In this case the CHAR model is significantly penalised, but the JOINT model can recover most of the performance. For instance, on the 50/50 split, the JOINT model accuracy is 89.1%, only -2% compared to the 91.0% obtained when the training and testing vocabularies are equal.

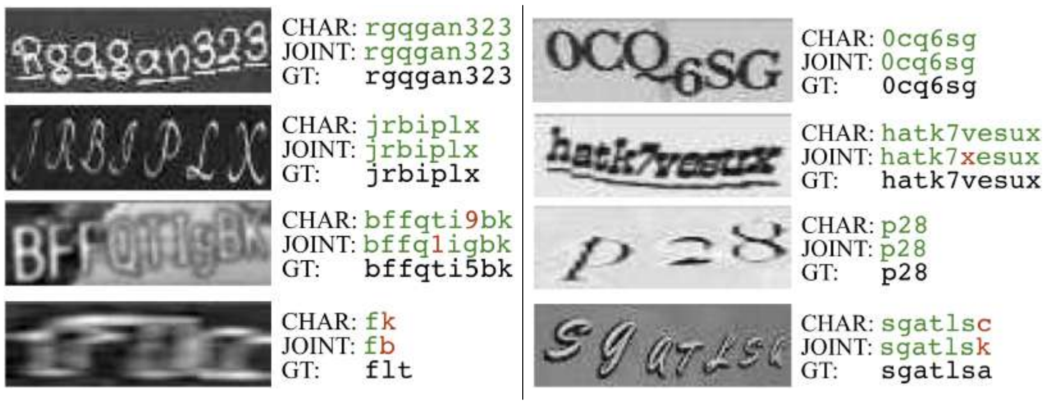


Figure 6.5: Some results of the CHAR and JOINT model on the SynthRand test dataset. Letters in red have been predicted incorrectly with the groundtruth (GT) shown below. Notice the range in difficulty of the SynthRand data.

The final test pushes generalisation by training and testing on completely random strings from SynthRand. As this dataset is a lot less regular than a natural language, the performance of the CHAR model suffers, dropping to 80.7% accuracy. Furthermore, as could be expected form the absence of common N-grams in the random language, the JOINT model performs *slightly worse* at 79.5% accuracy. However this drop is very small because N-grams are not used as hard constraints on the predicted words, but rather to nudge the word scores based on further visual cues. Some example results are shown in Figure 6.5.

6.3.3.3 Comparison to the state-of-the-art

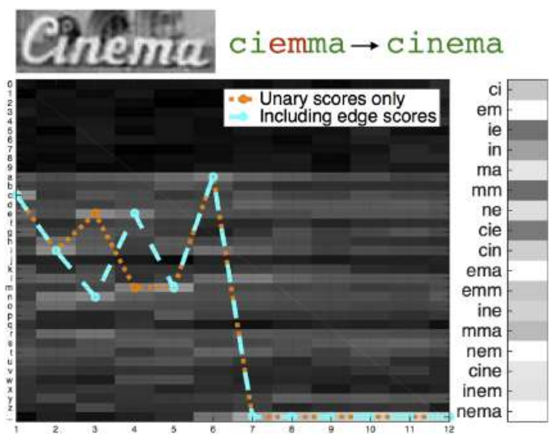
Table 6.2 compares the accuracy of CHAR and JOINT to previous works. Whereas these works make use of strong language models, our models make minimal assumptions about the language. In the constrained lexicon cases (the starred columns of Table 6.2), both CHAR and JOINT are very close to the state-of-the-art dictionary classification model of Chapter 5. We will refer to the dictionary classification model in Chapter 5 as DICT. Furthermore, if the same 90k dictionary used by DICT – 90kDict – is used to constrain the output of the JOINT model, the performance is identical to that achieved in Chapter 5, at 93.1% accuracy on IC03, and better than the word recognition stage of Chapter 4. While in the constrained lexicon experiments the lexicon is limited at test time, these results are still remarkable because, differently from DICT, CHAR and JOINT are not formulated for a specific dictionary. In particular, DICT would not be able to operate on random strings.

The recognition results without a lexicon are still behind that of some constrained models, however the JOINT model provides competitive performance and is far more flexible to recognise unseen words than previous works, while still achieving state-of-the-art performance if a lexicon is then applied as a constraint at test time. Figure 6.6 shows some example results where the CHAR model does not recognise the word correctly but the JOINT model succeeds.

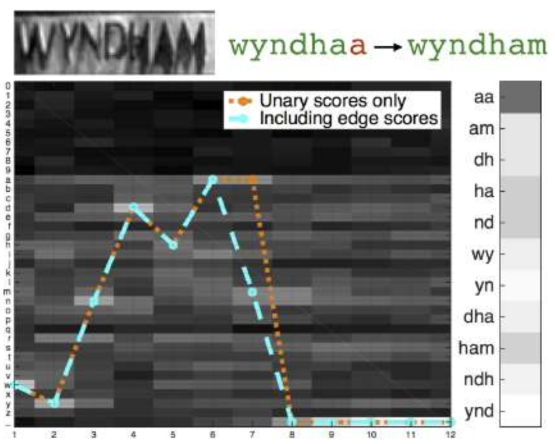
Limitations. An issue with the recognition models proposed in this chapter is that only fixed width images are modelled, causing squashing of long words and the destruction of visual information. The introduction of a recurrent neural network, with the model proposed in this chapter being unrolled over multiple positions across the input image could help address this limitation. There is also a large disparity in accuracy when testing on synthetic data (Synth90k) and easy real-world data (IC03) compared to harder real-world data (SVT). This calls for more realistic synthetic training data or for fine-tuning the model on hard, real-world data.

6.4 Conclusion

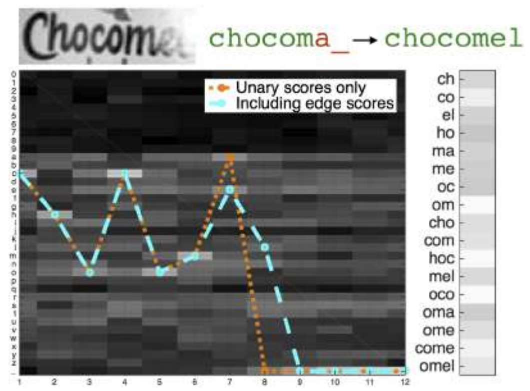
In this chapter we have introduced a new formulation for word recognition, designed to be used identically in language and non-language scenarios. By modelling character positions and the presence of common N-grams, we can define a joint graphical model. This can be trained effectively by back propagating structured output loss, and results in a more accurate word recognition system than predicting characters alone. We show impressive results for unconstrained text recognition with the ability to generalise recognition to previously unseen words, and match state-of-the-art



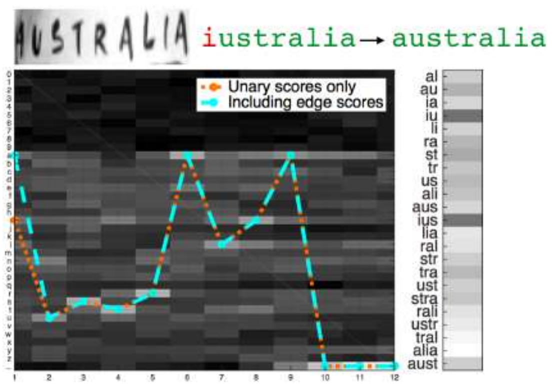
(a)



(b)



(c)



(d)

Figure 6.6: Results where the unary terms of the JOINT model cannot solely recognise the word correctly, but the addition of the edge scores result in correct recognition, from SVT (a,b) and IC13 (c,d). The input image is shown in the top left corner. The unary scores for characters (rows) at each position (columns, up to 12 out of 23 characters) are shown, with the selected path using only the unary score term S_c^i (orange) and when edge scores S_e are incorporated (cyan). The bars show the NGRAM strengths, with lighter colour representing larger values.

Model	IC03-50*	IC03-Full*	IC03-50k*	IC03	SVT-50*	SVT	IC13	IIIT5k-50*	IIIT5k-1k*
ABBYY	56.0	55.0	-	-	35.0	-	-	24.3	-
Wang et al., 2011	76.0	62.0	-	-	57.0	-	-	-	-
Mishra et al., 2012	81.8	67.8	-	-	73.2	-	-	-	-
Novikova et al., 2012	82.8	-	-	-	72.9	-	-	64.1	57.5
Wang et al., 2012	90.0	84.0	-	-	70.0	-	-	-	-
Goel et al., 2013	89.7	-	-	-	77.3	-	-	-	-
Bissacco et al., 2013a	-	-	-	-	90.4	78.0	87.6	-	-
Alsharif and Pineau, 2014	93.1	88.6	85.1	-	74.3	-	-	-	-
Almazán et al., 2014	-	-	-	-	89.2	-	-	91.2	82.1
Yao et al., 2014	88.5	80.3	-	-	75.9	-	-	80.2	69.3
Chapter 4	96.2	91.5	-	-	86.1	-	-	-	-
Gordo, 2014	-	-	-	-	90.7	-	-	93.3	86.6
Chapter 5	**98.7**	**98.6**	93.3	**93.1**	**95.4**	**80.7**	**90.8**	**97.1**	**92.7**
CHAR (This chapter)	98.5	96.7	92.3	85.9	93.5	68.0	79.5	95.0	89.3
JOINT (This chapter)	97.8	97.0	**93.4**	89.6	93.2	71.7	81.8	95.5	89.6

Table 6.2: Comparison to previous methods. The baseline method is from a commercially available OCR system. Note that the training data for DICT includes the lexicons of the test sets, so it has the capacity to recognise all test words. The baseline method is from a commercially available document OCR system, ABBYY, evaluated in [Wang et al., 2011; Yao et al., 2014]. *Results are constrained to the lexicons described in Section 6.3.1.

accuracy when comparing in lexicon constrained scenarios.

Chapter 7

Speeding up Convolutional Neural Networks

In this chapter we depart from the specific applications of deep learning to text detection and recognition, and focus on speeding up convolutional neural networks.

We have demonstrated in previous chapters that CNNs can be extremely powerful machines for text spotting, and as discussed in Section 2.1.2 CNNs achieve state-of-the-art results in a range of other computer vision tasks [Simonyan and Zisserman, 2015; Taigman et al., 2014; Sermanet et al., 2013; Tompson et al., 2014]. However, these high performing CNNs come with a large computational cost due to the use of chains of several convolutional layers, often requiring implementations on GPUs [Krizhevsky et al., 2012; Jia, 2013] or highly optimised distributed CPU architectures [Vanhoucke et al., 2011] to process large datasets. The increasing use of these networks for detection in sliding window approaches [Sermanet et al., 2013; Farabet et al., 2012; Oquab et al., 2014] and the desire to apply CNNs in real-world production systems to cater for millions of users, means the speed of inference becomes an important factor for applications.

To tackle this issue, we introduce an easy-to-implement method for significantly speeding up pre-trained CNNs requiring minimal modifications to existing CNN frameworks. Since this method is based on approximations, there can be a small associated loss in performance, but this is tuneable to a desired accuracy level. For example, we show that a $4.5\times$ speedup can still give state-of-the-art performance in our application to the network of Chapter 4.

While a few other CNN acceleration methods exist (see Section 2.1.3 for a review of previous methods), our key insight is to exploit the redundancy that exists between different feature channels and filters, highlighted in the work of Denil et al., 2013. In particular, we build upon the work of Rigamonti et al., 2013 which shows that multiple image filters can be approximated by a shared set of separable (rank-1) filters, allowing large speedups with minimal loss in accuracy.

We contribute two approximation schemes to do so (Section 7.1) and two optimisation methods for each scheme (Section 7.3). Both schemes are orthogonal to other architecture-specific optimisations and can be easily applied to existing CPU and GPU software. Performance is evaluated empirically in Section 7.4 and results are summarised in Section 7.5.

Note, the methods we propose are not specific to any processing architecture or application, and can be combined with many of the other speedup methods given in the literature.

7.1 Filter Approximations

Filter banks are used widely in computer vision as a method of feature extraction, and when used in a convolutional manner, generate *feature maps* from input images. For an input $\mathtt{x} \in \mathbb{R}^{H\times W}$, the set of output feature maps $\mathcal{Y} = \{\mathtt{y}_1, \mathtt{y}_2, \ldots, \mathtt{y}_N\}$, $\mathtt{y}_n \in \mathbb{R}^{H'\times W'}$ are generated by convolving $\mathtt{x}$ with N filters $\mathcal{F} = \{\mathtt{f}_i\}\ \forall i \in [1 \ldots N]$ such that $\mathtt{y}_i = \mathtt{f}_i * \mathtt{x}$. The collection of filters $\mathcal{F}$ can be learnt, for example, through dictionary learning methods [Kavukcuoglu et al., 2010; Lee et al., 2009; Rigamonti et al., 2011] or CNNs, and are generally full rank and expensive to convolve with large images. Using a direct implementation of convolution, the complexity of convolving a single channel input image with a bank of N 2D filters of size $d \times d$ is $\mathcal{O}(d^2NH'W')$. We next introduce our method for accelerating this computation that takes advantage of the fact that there exists significant redundancy *between* different filters and feature channels.

One way to exploit this redundancy is to approximate the filter set by a linear combination of a smaller basis set of M filters [Rigamonti et al., 2013; Song et al., 2012; Song et al., 2013]. The basis filter set $\mathcal{S} = \{\mathtt{s}_i\}\ \forall i \in [1 \dots M]$ is used to generate basis feature maps which are then linearly combined such that

$$\mathtt{y}_i \simeq \sum_{k=1}^{M} a_{ik}\mathtt{s}_k * \mathtt{x} \tag{7.1}$$

This can lead to a speedup in feature map computation as a smaller number of filters need be convolved with the input image, and the final feature maps are composed of a cheap linear combination of these. The complexity in this case is $\mathcal{O}((d^2M + MN)H'W')$, so a speedup can be achieved if $M < \frac{d^2N}{d^2+N}$.

As shown in [Rigamonti et al., 2013], further speedups can be achieved by choosing the filters in the approximating basis to be rank-1 and so making individual convolutions *separable*. This means that each basis filter can be decomposed in to a sequence of horizontal and vertical filters $\mathtt{s}_i * \mathtt{x} = \mathbf{v}_i * (\mathbf{h}_i{}^\top * x)$ where $\mathtt{s}_i \in \mathbb{R}^{d\times d}$, $\mathbf{v}_i \in \mathbb{R}^{d\times 1}$, and $\mathbf{h}_i{}^\top \in \mathbb{R}^{1\times d}$. Using this decomposition, the convolution of a separable filter $\mathtt{s}_i$ can be performed in $\mathcal{O}(2dH'W')$ operations instead of $\mathcal{O}(d^2H'W')$.

The separable filters of Rigamonti et al., 2013 are a low-rank approximation, but performed in the *spatial* filter dimensions. We expand upon this and show that in CNNs substantial speedups can be achieved by also exploiting the cross-channel redundancy to perform low-rank decomposition in the *channel* dimension as well.

We explore both of these low-rank approximations in the following section.

Note that the FFT speedup method demonstrated by [Mathieu et al., 2013] could be used as an alternative method to accelerate individual convolutions in combination with our low-rank cross-channel decomposition scheme. However, separable convolutions have several practical advantages: they are significantly easier to implement than a well tuned FFT implementation, particularly on GPUs; they do not require feature maps to be padded to a special size, such as a power of two as in Mathieu et al., 2013; they are far more memory efficient; and, they yield a good speedup for small image and filter sizes too (which can be common in CNNs), whilst FFT acceleration tends to be better for large filters due to the overheads incurred in computing the FFTs.

7.2 Approximating CNN Filter Banks

We now describe the approximation schemes applied to a single layer, but this can of course be applied to each layer of a CNN.

The N filters $\mathtt{w}_n$ learnt for each layer are full rank, 3D filters with the same depth as the number of channels of the input, such that $\mathbf{w}_n(u,v) \in \mathbb{R}^C$. For example, for a 3-channel color image input, $C = 3$. The convolution $\mathtt{w}_n * \mathtt{z}$ of a 3D filter $\mathtt{w}_n$ with the 3D image $\mathtt{z}$ is the 2D image $\mathtt{w}_n * \mathtt{z} = \sum_{c=1}^{C} \mathtt{w}_n^c * \mathtt{z}^c$, where $\mathtt{w}_n^c \in \mathbb{R}^{d\times d}$ is a single channel of the filter. This is a sum of 2D convolutions so we can think of each 3D filter as being a collection of 2D filters, whose output is collapsed to a 2D signal. However, since N such 3D filters are applied to $\mathtt{z}$, the overall output is a new 3D image with N channels. This process is illustrated for the case $C = 1, N > 1$ in Figure 7.1 (a). The resulting computational cost for a convolutional layer with N filters of size $d \times d$ acting on C input channels is $\mathcal{O}(CNd^2H'W')$.

We now propose two schemes to approximate a convolutional layer of a CNN to reduce the computational complexity and discuss their training in Section 7.3.

Both schemes follow the same intuition: that CNN filter banks can be approximated using a low rank basis of filters that are separable in the spatial domain.

7.2.1 Scheme 1

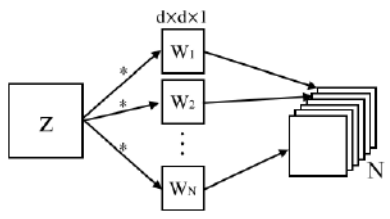
The first method for speeding up convolutional layers is to directly apply the method suggested in Section 7.1 to the filters of a CNN (Figure 7.1 (b)). As described above, a single convolutional layer with N filters $\mathtt{w}_n \in \mathbb{R}^{d\times d\times C}$ requires evaluating NC 2D filters $\mathcal{F} = \{\mathtt{w}_n^c \in \mathbb{R}^{d\times d} : n \in [1 \dots N], c \in [1 \dots C]\}$. Note that there are N filters $\{\mathtt{w}_n^c : n \in [1 \dots N]\}$ operating on each input channel $\mathtt{z}^c$. These can be approximated as linear combinations of a basis of $M < N$ (separable) filters $\mathcal{S}^c = \{\mathtt{s}_m^c : m \in [1 \dots M]\}$ as $\mathtt{w}_n^c \simeq \sum_{m=1}^M a_n^{cm} \mathtt{s}_m^c$. Since convolution is a linear operator, filter reconstruction and image convolution can be swapped, yielding the approximation

$$\mathtt{w}_n * z = \sum_{c=1}^{C} \mathtt{w}_n^c * \mathtt{z}^c \simeq \sum_{c=1}^{C}\sum_{m=1}^{M} a_n^{cm}(\mathtt{s}_m^c * \mathtt{z}^c). \tag{7.2}$$

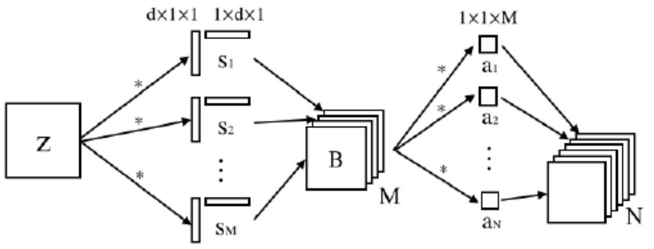
To summarize, the direct calculation involves computing NC 2D filters $\mathtt{w}_n^c * \mathtt{z}^c$ with cost $O(NCd^2H'W')$, while the approximation involves computing MC 2D filters $\mathtt{s}_m^c * \mathtt{z}^c$ with cost $O(MC(d^2+N)H'W')$ – the additional $MCNH'W'$ term accounting for the need to recombine the basis response linearly. Due to the second term, the approximation is efficient only if $M \ll d^2$, *i.e.* if the number of filters in the basis is less than the filter area.

The first cost term $CMd^2H'W'$ would also suggest that efficiency requires the condition $M \ll N$; however, this can be considerably ameliorated by using *separable filters* in the basis. In this case the approximation cost is reduced to $O(MC(d + N)H'W')$; together with the former condition, Scheme 1 is then efficient if $M \ll d \min\{d, N\}$.

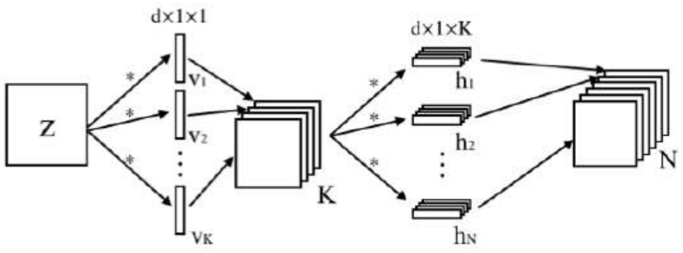
Note that this scheme uses C filter basis $\mathcal{S}^1, \mathcal{S}^2, \dots, \mathcal{S}^C$ as each operates on a different input channel. In practice, we choose $\mathcal{S}^1 = \mathcal{S}^2 = \dots = \mathcal{S}^C = \mathcal{S}$ because



(a)



(b)



(c)

Figure 7.1: (a) The original convolutional layer acting on a single-channel input *i.e.* C=1. (b) The approximation to that layer using the method of Scheme 1. (c) The approximation to that layer using the method of Scheme 2. Individual filter dimensions are given above the filter layers.

empirically there is no actual gain in performance and a single channel basis is simpler and more compact.

The bank of separable filters $\mathcal{S}$ can be thought of as producing MC feature maps $\mathcal{B}$, which are then linearly combined by N convolutional filters of size $1 \times 1 \times MC$.

7.2.2 Scheme 2

Scheme 1 focuses on approximating 2D filters. As a consequence, each input channel z^c is approximated by a particular basis of 2D separable filters. Redundancy amongst feature channels is exploited, but only in the sense of the N *output channels*. In contrast, Scheme 2 is designed to take advantage of both input and output redundancies by considering 3D filters throughout. The idea is simple: each convolutional layer is factored as a sequence of two regular convolutional layers but with rectangular (in the spatial domain) filters, as shown in Figure 7.1 (c). The first convolutional layer has K filters of spatial size $d \times 1$ resulting in a filter bank $\{\mathtt{v}_k \in \mathbb{R}^{d\times 1\times C} : k \in [1 \dots K]\}$ and producing output feature maps $\mathtt{V}$ such that $\mathbf{V}(u,v) \in \mathbb{R}^K$. The second convolutional layer has N filters of spatial size $1 \times d$ resulting in a filter bank $\{\mathtt{h}_n \in \mathbb{R}^{1\times d\times K} : n \in [1 \dots N]\}$. Differently from Scheme 1, the filters operate on multiple channels simultaneously. The rectangular shape of the filters is selected to match a separable filter approximation. To see this, note that convolution by one of the original filters $\mathtt{w}_n * \mathtt{z} = \sum_{c=1}^{C} \mathtt{w}_n^c * \mathtt{z}^c$ is approximated by

$$\begin{aligned} \mathtt{w}_n * \mathtt{z} \simeq \mathtt{h}_n * \mathtt{V} = \sum_{k=1}^{K} \mathbf{h}_n^{k\top} * \mathtt{V}^k &= \sum_{k=1}^{K} \mathbf{h}_n^{k\top} * (\mathbf{v}_k * \mathtt{z}) \\ &= \sum_{k=1}^{K} \mathbf{h}_n^{k\top} * \sum_{c=1}^{C} \mathbf{v}_k^c * \mathtt{z}^c \\ &= \sum_{c=1}^{C} \left[\sum_{k=1}^{K} \mathbf{h}_n^{k\top} * \mathbf{v}_k^c \right] * \mathtt{z}^c \end{aligned} \tag{7.3}$$

which is the sum of separable filters $\mathbf{h}_n^{k\top} * \mathbf{v}_k^c$. The computational cost of this scheme is $O(KCdH'W)$ for the first vertical filters and $O(NKdH'W')$ for the second horizontal filter. Assuming that the image width $W \gg d$ is significantly larger than the filter size, the output image width $W \approx W'$ is about the same as the input image width W'. Hence the total cost can be simplified to $O(K(N+C)dH'W')$. Compared to the direct convolution cost of $O(NCd^2H'W')$, this scheme is therefore convenient provided that $K(N+C) \ll NCd$. For example, if K, N, and C are of the same order, the speedup is about d times.

In both schemes, we are assuming that the full rank original convolutional filter bank can be decomposed into a linear combination of a set of separable basis filters. The difference between the schemes is how/where they model the interaction between input and output channels, which amounts to how the low rank channel space approximation is modelled. In Scheme 1 it is done with the linear combination layer, whereas with Scheme 2 the channel interaction is modelled with 3D vertical and horizontal filters inducing a summation over channels as part of the convolution.

7.3 Optimisation

This section deals with the details on how to attain the optimal separable basis representation of a convolutional layer for the schemes. The first method (Section 7.3.1) aims to reconstruct the original filters directly by minimising filter reconstruction error. The second method (Section 7.3.2) approximates the convolutional layer indirectly, by minimising the empirical reconstruction error of the filter output.

7.3.1 Filter Reconstruction Optimisation

The first way that we can attain the separable basis representation is to aim to minimise the reconstruction error of the original filters with our new representation.

Scheme 1. The separable basis can be learnt simply by minimising the L_2 reconstruction error of the original filters, whilst penalising the nuclear norm $\|\mathtt{s}_m\|_*$ of the filters $\mathtt{s}_m$. In fact, the nuclear norm $\|\mathtt{s}_m\|_*$ is a proxy for the rank of $\mathtt{s}_m \in \mathbb{R}^{d\times d}$ and rank-1 filters are separable. This yields the formulation:

$$\min_{\{\mathtt{s}_m\},\{a_n^{cm}\}} \sum_{n=1}^{N}\sum_{c=1}^{C}\left\| \mathtt{W}_n^c - \sum_{m=1}^{M} a_n^{cm}\mathtt{s}_m \right\|_2^2 + \lambda \sum_{m=1}^{M} \|\mathtt{s}_m\|_*. \tag{7.4}$$

This minimisation is biconvex, so given $\{\mathtt{s}_m\}$ a unique $\{a_n^{cm}\}$ can be found, therefore a minimum is found by alternating between optimising $\{\mathtt{s}_m\}$ and $\{a_n^{cm}\}$. A suitably large λ ensures the resulting filters are rank-1. For full details of the implementation of this optimisation see Rigamonti et al., 2013.

Scheme 2. The set of horizontal and vertical filters can be learnt by explicitly minimising the L_2 reconstruction error of the original filters. From Equation 7.3 we can see that the original filter can be approximated by minimising the objective function

$$\min_{\{\mathbf{h}_n^{k\top}\},\{\mathbf{v}_k^c\}} \sum_{n=1}^{N}\sum_{c=1}^{C} \left\| \mathtt{W}_n^c - \sum_{k=1}^{K} \mathbf{h}_n^{k\top} * \mathbf{v}_k^c \right\|_2^2 . \tag{7.5}$$

This optimisation is simpler than for Scheme 1 due to the lack of nuclear norm constraints, which we are able to avoid by modelling the separability explicitly with two variables. We perform conjugate gradient descent, alternating between optimising the horizontal and vertical filter sets.

7.3.2 Data Reconstruction Optimisation

The problem with optimising the separable basis through minimising original filter reconstruction error is that this does not necessarily give the most optimised basis set for the end CNN prediction performance. As an alternative, one can optimise a scheme's separable basis by aiming to reconstruct the *outputs* of the original convolutional layer given training data. For example, for Scheme 2 this amounts to

$$\min_{\{\mathbf{h}_n^{k\top}\},\{\mathbf{v}_k^c\}} \sum_{i=1}^{|\mathcal{X}|}\sum_{n=1}^{N} \left\| \mathtt{W}_n * \Phi_{l-1}(\mathtt{x}_i) - \sum_{c=1}^{C}\sum_{k=1}^{K} \mathbf{h}_n^{k\top} * \mathbf{v}_k^c * \Phi_{l-1}(\mathtt{x}_i) \right\|_2^2 \tag{7.6}$$

where l is the index of the convolutional layer to be approximated and $\Phi_l(\mathtt{x}_i)$ is the evaluation of the CNN up to and including layer l on data sample $\mathtt{x}_i \in \mathcal{X}$ where $\mathcal{X}$ is the set of training examples. This optimisation can be done quite elegantly by simply mirroring the CNN with the un-optimised separable basis layers, and training only the approximation layer by back-propagating the L_2 error between the output of the original layer and the output of the approximation layer (see Figure 7.2). This is done layer by layer.

There are two main advantages of this method for optimisation of the approxima-

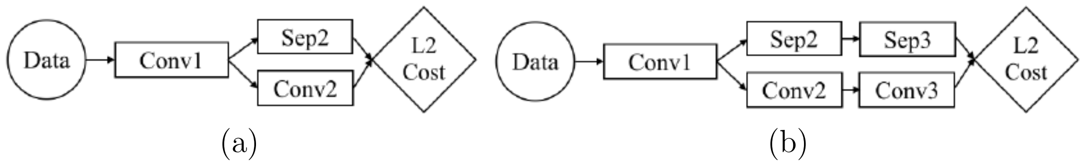


Figure 7.2: Example network schematics of how to optimise separable basis approximation layers in a data reconstruction setting. (a) Approximating Conv2 with Sep2. (b) Approximating Conv3 with Sep3, incorporating the approximation of Conv2 as well. Conv2 and Conv3 represent the original full-rank convolutional layers, whereas Sep2 and Sep3 represent the approximation structure that is learnt by back-propagating the L_2 error between the outputs.

tion schemes. The first is that the approximation is conditioned on the manifold of the training data – original filter dimensions that are not relevant or redundant in the context of the training data will by ignored by minimising data reconstruction error, but will still be penalised by minimising filter reconstruction error (Section 7.3.1) and therefore uselessly using up model capacity. Secondly, stacks of approximated layers can be learnt to incorporate the approximation error of previous layers by feeding the data through the approximated net up to layer l rather than the original net up to layer l (see Figure 7.2 (b)). This additionally means that all the approximation layers could be optimised jointly with back-propagation.

An obvious alternative optimisation strategy would be to replace the original convolutional layers with the un-optimised approximation layers and train just those layers by back-propagating the classification error of the approximated CNN. However, this does not actually result in better classification accuracy than doing L_2 data reconstruction optimisation – in practice, optimising the separable basis within the full network leads to overfitting of the training data, and attempts to minimise this overfitting through regularisation methods like dropout [Hinton et al., 2012] lead to under-fitting, most likely due to the fact that we are already trying to heavily approximate our original filters. However, this is an area that needs to be investigated in more detail.

7.4 Experiments

In this section we demonstrate the application of both proposed filter approximation schemes and show that we can achieve large speedups with a very small drop in accuracy. As a test model, we use the pre-trained CNN that performs case-insensitive character classification of scene text from Chapter 4.

We first give the details of the base CNN model used for character classification which will be subject to speedup approximations. The optimisation processes and how we attain the approximations of Scheme 1 & 2 to this model are given, and finally we discuss the results of the separable basis approximation methods on accuracy and inference time of the model.

7.4.1 Test Model

We use the case-insensitive character classifier model from Chapter 4 as a test model to apply our speedup framework. As a reminder, this is a four layer CNN with a softmax output which models a probability distribution $p(c|\mathtt{x})$ over an alphabet C which includes all 26 letters and 10 digits, as well as a noise/background (no-text) class. The input $\mathtt{x}$ is a grey input image patch of size 24×24 pixels, which has been zero-centred and normalised by subtracting the patch mean and dividing by the standard deviation. The non-linearity used between convolutional layers is maxout [Goodfellow et al., 2013a] as defined in Section 4.1. For reference, Table 7.1 gives the details of the layers for the model used, which is connected in the linear arrangement Conv1-Conv2-Conv3-Conv4-Softmax.

7.4.2 Datasets & Evaluation

We use the same training and test data as in Chapter 4. The training dataset consists of the 163k collected character samples from a number of datasets (see Section 4.3.1 for details) and the test set is a collection of 5k cropped characters

Layer name	Filter size	In channels	Out channels	Filters	Maxout groups	Time
Conv1	9×9	1	48	96	2	0.473ms (8.3%)
Conv2	9×9	48	64	128	2	3.008ms (52.9%)
Conv3	8×8	64	128	512	4	2.160ms (38.0%)
Conv4	1×1	128	37	148	4	0.041ms (0.7%)
Softmax	-	37	37	-	-	0.004ms (0.1%)

Table 7.1: The details of the layers in the CNN used with the forward pass timings of each layer.

from the ICDAR 2003 test dataset (Section 3.1.1). We evaluate the case-insensitive accuracy of the classifier, ignoring the background class. As shown in Chapter 4, the Test Model achieves state-of-the-art results of 91.3% accuracy compared to the next best result of 89.8% Alsharif and Pineau, 2014.

7.4.3 Implementation Details

The CNN framework we use is the CPU implementation of `Caffe` [Jia, 2013], where convolutions are performed by constructing a matrix of filter windows of the input, `im2col`, and using BLAS for the matrix-matrix multiplication between the filters and data windows. At time of writing, we found this to be the fastest CPU CNN implementation attainable. We use the network exactly as it is trained in Chapter 4.

For filter reconstruction optimisation, we optimise a separable basis until a stable minimum of reconstruction error is reached. For data reconstruction optimisation, we optimise each approximated layer in turn, and can incorporate a fine-tuning with joint optimisation.

For the CNN presented, we only approximate layers Conv2 and Conv3. This is because layer Conv4 has a 1×1 filter size and so would not benefit much from our speedup schemes. We also don't approximate Conv1 due to the fact that it acts on raw pixels from natural images – the filters in Conv1 are very different to those found in the rest of the network and experimentally we found that they cannot be approximated well by separable filters (also observed in Denton et al.,

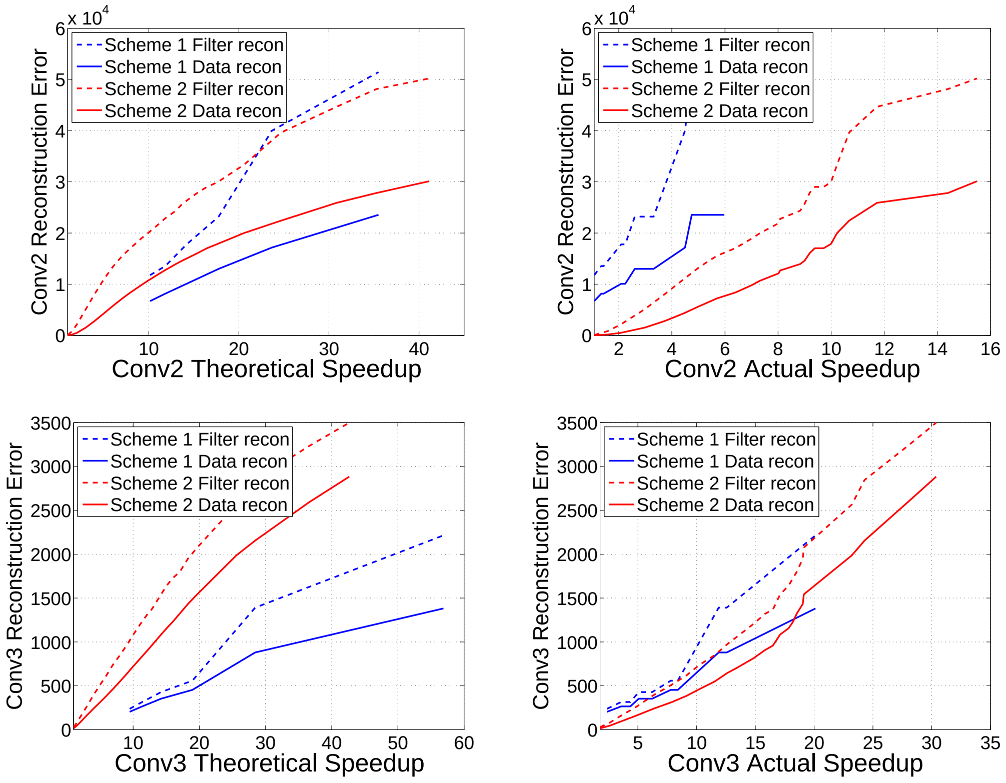
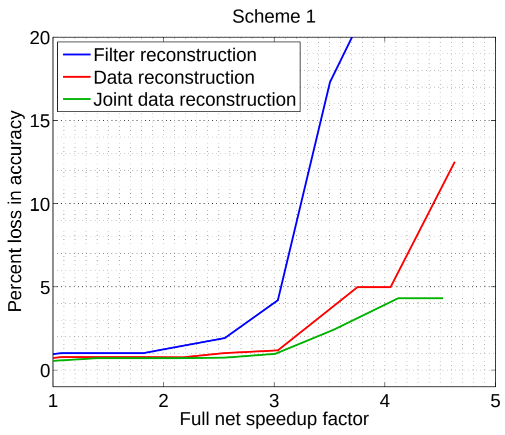


Figure 7.3: Reconstruction error for the theoretical and actual attained speedups on test data for Conv2 & Conv3. Theoretical speedups are obtained by considering the theoretical number of operations required for the different schemes with direct convolution. We do not go below 10× theoretical speedup for Scheme 1 as computation takes too long.

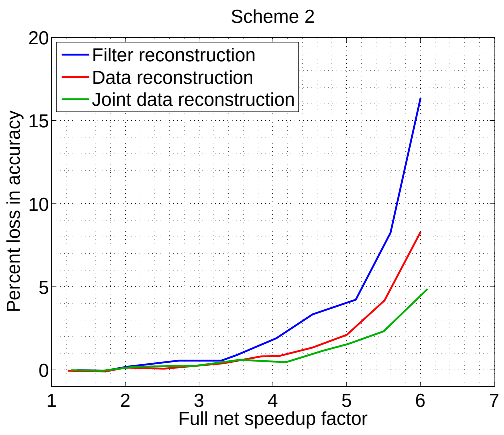
2014). Omitting layers Conv1 and Conv4 from the schemes does not change overall network speedup significantly, since Conv2 and Conv3 constitute 90% of the overall network processing time, as shown in Table 7.1.

7.4.4 Results

We now give the results of our speedup methods by assessing the layer-wise performance, as well as the speedups and accuracy that can be achieved when applying our approximations to the full classification network.



(a)



(b)

Figure 7.4: The percent loss in performance as a result of the speedups attained with Scheme 1 (a) and Scheme 2 (b).

7.4.4.1 Layer-wise Performance

Figure 7.3 shows the output reconstruction error of each approximated layer with the test data. It is clear that the reconstruction error worsens as the speedup achieved increases, both theoretically and practically. As the reconstruction error is that of the test data features fed through the approximated layers, as expected the data reconstruction optimization scheme gives lower errors for the same speedup compared to the filter reconstruction. This generally holds even when completely random Gaussian noise data is fed through the approximated layers – data from a completely different distribution to what the data optimization scheme has been trained on.

Looking at the theoretical speedups possible in Figure 7.3, Scheme 1 gives better reconstruction error to speedup ratio, suggesting that the Scheme 1 model is perhaps better suited for approximating convolutional layers. However, when the actual measured speedups are compared, Scheme 1 is actually slower than that of Scheme 2 for the same reconstruction error. This is due to the fact that the `Caffe` convolution routine is optimized for 3D convolution (summing over channels), so Scheme 2 requires only two `im2col` and BLAS calls. However, to implement Scheme 1 with

`Caffe` style convolution involving per-channel convolution without channel summation, means that there are many more costly `im2col` and BLAS calls, thus slowing down the layer evaluation and negating the model approximation speedups. It is possible that using a different convolution routine with Scheme 1 will bring the actual timings closer to the theoretically achievable timings.

7.4.4.2 Full Net Performance

Figure 7.4 (b) & (c) show the overall drop in accuracy as the speedup of the end-to-end network increases under different optimization strategies. Generally, joint data optimization of Conv2 and Conv3 improves final classification performance for a given speedup. Under Scheme 2 we can achieve a 2.5× speedup with no loss in accuracy, and a 4.5× speedup with only a drop of 1% in classification accuracy, giving 90.3% accuracy – still state-of-the-art for this benchmark. The 4.5× configuration is obtained by approximating the original 128 Conv2 filters with 31 horizontal filters followed by 128 vertical filters, and the original 512 Conv3 filters with 26 horizontal filters followed by 512 vertical filters.

This speedup is particularly useful for sliding window schemes, allowing fast generation of, for example, detection maps such as the character detection and classification maps shown in Figure 7.5 (a) and (b). There is very little difference with even a 3.5× speedup, and when incorporated in to a full application pipeline, the speedup can be tuned to give an acceptable end-pipeline result. Figure 7.5 (c) shows a selection of filter reconstructions obtained by the two schemes. While they perform similarly, visually it is clear the two presented schemes act very differently.

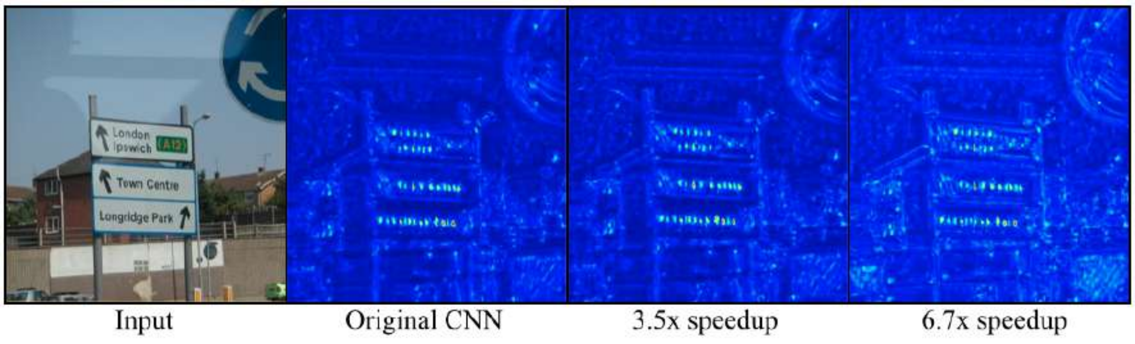
Comparing to an FFT based CNN [Mathieu et al., 2013], our method can result in greater speedups. With the same layer setup (5×5 kernel, $16 \times 16 \times 256$ input, 384 filters), Scheme 2 gives an actual 2.4× speedup with 256 basis filters (which should result in no performance drop), compared to 2.2× in Mathieu et al., 2013.

Comparing with Denton et al., 2014, simply doing a filter reconstruction approximation with Scheme 2 of the second layer of OverFeat [Sermanet et al., 2013] gives a $2\times$ theoretical speedup with only 0.5% drop in top-5 classification accuracy on ImageNet, far better than the 1.2% drop in accuracy for the same theoretical speedup reported in Denton et al., 2014. However, these results are not directly comparable since the exact test data is different. This accuracy should be further improved if data optimisation is used.

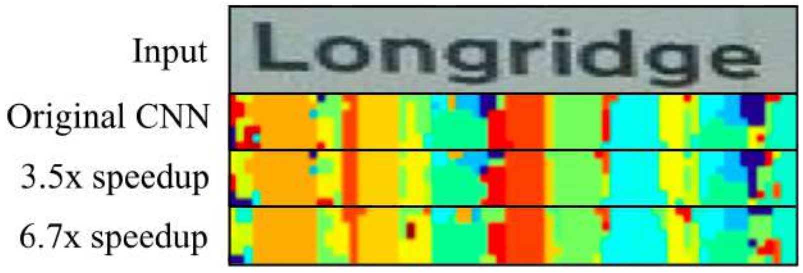
7.5 Conclusions

In this chapter we have shown that the redundancies in representation in CNN convolutional layers can be exploited by approximating a learnt full rank filter bank as combinations of a rank-1 filter basis. We presented two schemes to do this, with two optimization techniques for attaining the approximations. The resulting approximations require significantly less operations to compute, resulting in large speedups observed with a real CNN trained for scene text character recognition: a $4.5\times$ speedup, only a drop of 1% in classification accuracy.

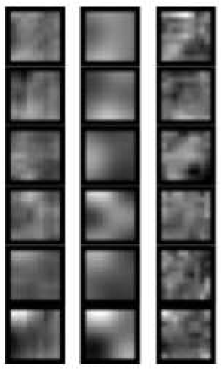
Since this work was first published [Jaderberg et al., 2014c], the following papers have extended our methods. Lebedev et al., 2014 perform full CP-decomposition on the entire 4D filter volumes to approximate the decomposition, and fine tune layer-wise as we did. Jin et al., 2014 also extend the method of Scheme 2 to include an extra final layer which consists of 1D filters in the channel domain.



(a)



(b)



(c)

Figure 7.5: Text spotting using the CNN character classifier. (a) The maximum response map over the character classes of the CNN output with Scheme 2 indicates the scene text positions. The approximations have sufficient quality to locate the text, even at 6.7× speedup. (b) The character classifier responses for a single word image. Each colour represents a different character class. (c) A selection of Conv2 filters from the original CNN (left), and the reconstructed versions under Scheme 1 (centre) and Scheme 2 (right), where both schemes have the same model capacity corresponding to 10× theoretical speedup. Visually the approximated filters look very different with Scheme 1 naturally smoothing the representation, but both still achieve good accuracy.

Chapter 8

Applications

Throughout this thesis we have focussed on the development of text spotting systems, as well as the deep learning algorithms to allow these powerful systems to be possible. In this chapter, we now contemplate the *applications* of text spotting.

As discussed in Chapter 1, there are a range of possible applications of text spotting. With fast and accurate text spotting, machines and automated processes would be better able to understand the visual world, be it for real time navigation and world exploration, or for offline media understanding, cataloguing, and retrieval. With the introduction of the exceptionally high accuracy text recognition methods based on deep learning described in Chapter 5 and Chapter 6, as well as the big increase in end-to-end text spotting and text retrieval performance also displayed in Chapter 5, there is suddenly scope for realising some of the practical applications of text spotting.

In this chapter, we show that the text spotting system proposed in Chapter 5 can be applied for visual media retrieval, allowing video to be searched instantly for text contained within thousands of hours of footage. The footage we process is provided in partnership with the British Broadcasting Corporation (BBC) and constitutes thousands of hours of BBC News footage. We describe this application

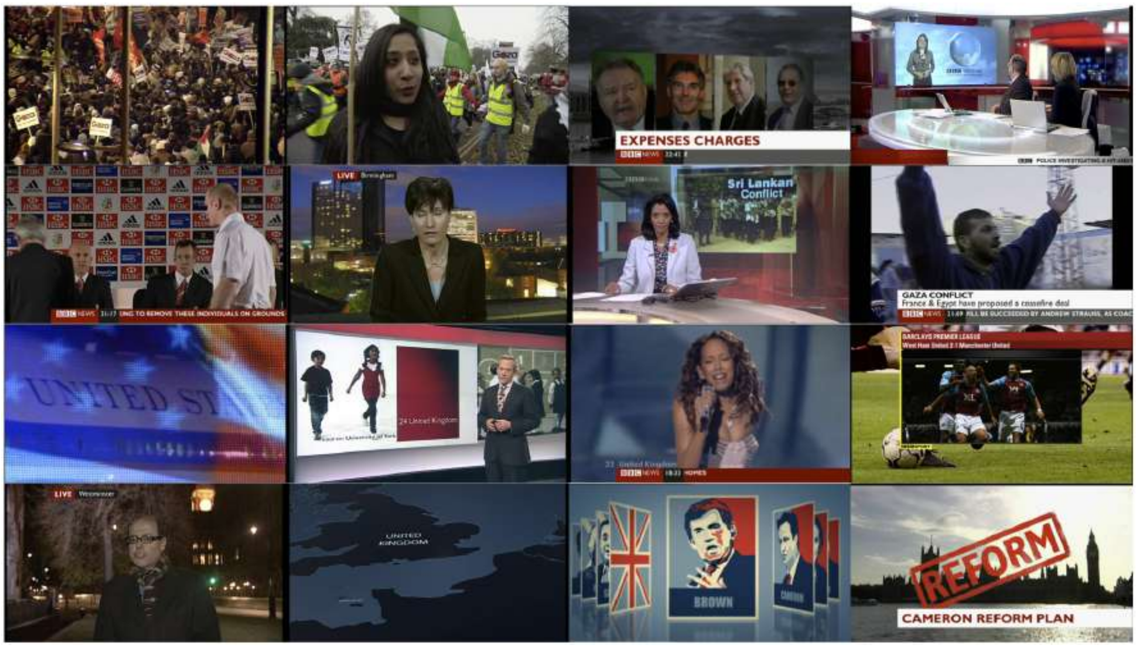


Figure 8.1: Some example frames from the BBC News dataset. There are a wide range of different types of text that occur within the frames: general scene text, artificially inserted labels, news ticker text, and 3D rendered graphics text.

in Section 8.1. To help enable future applications of this work, we have released software and data, outlined in Section 8.2.

8.1 Searching the News

In Chapter 5 we introduced a text spotting system based on word region proposals and a deep CNN for word recognition, exhibiting state-of-the-art performance on a wide range of datasets. In particular, what can be noted from Section 5.6.3 is the very high precision achieved on the text based image retrieval benchmarks. In addition, the algorithm is fast and easily parallelizable with multiple cores and multiple GPUs.

The high precision and speed of the pipeline in Chapter 5 allows us to process huge datasets for practical search applications. Here, we apply the text based image retrieval pipeline to TV broadcast search – allowing the frames, source videos, and

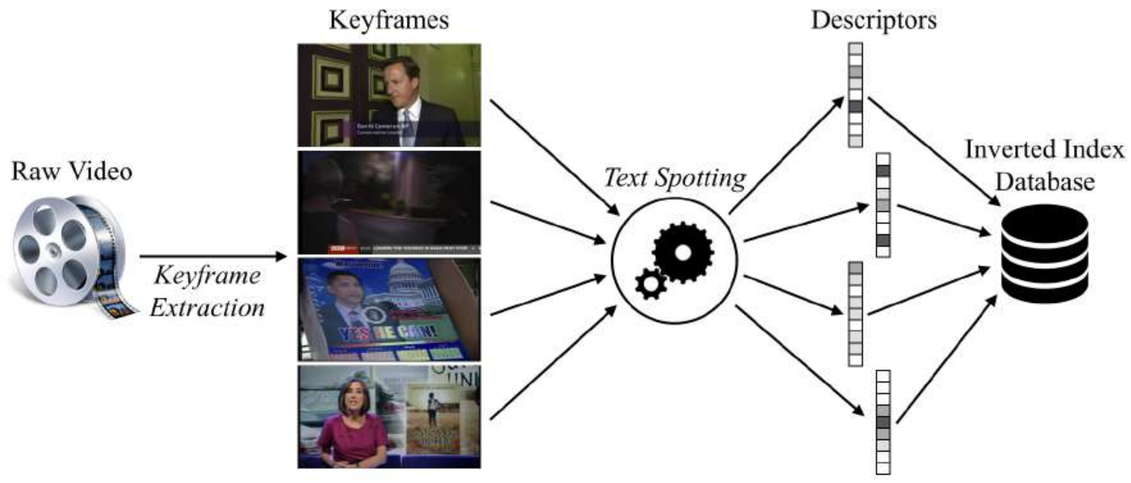


Figure 8.2: The processing pipeline for the video retrieval application. First, keyframes are extracted from the raw video file. Each of the keyframes are processed by the text spotting system of Chapter 5, producing per-image descriptors, encoding the text contained within the keyframe images. These descriptors are loaded in to an in-memory database with an inverted index for fast retrieval at query time.

metadata from the broadcasts which visually contain the search term to be instantly retrieved. This is a very useful function to have when vast archives of video footage need to be searched, as footage can be retrieved on the *visual content* in addition to the textual metadata which is easily searchable by traditional text indexing methods. Being able to search for text *within the video* would complement other search modalities, such as searching for objects [Chatfield and Zisserman, 2013].

We demonstrate this application on a large BBC News video dataset. We build a search engine and front-end web application around our text based image retrieval pipeline, allowing a user to instantly search for visual occurrences of text within the huge video dataset, and inspect the frames and broadcasts where the text occurs.

Data. The data consists of 5000 hours of footage broadcast on a number of BBC TV channels in the evening (after 6pm) between 2007 and 2012, and is limited to programmes with a genre label of "news", "factual", and "politics". The 5000 hours of video corresponds to approximately 12 million frames. The videos are taken from a range of different BBC programmes on news and current affairs, including the BBC's Evening News programme. Text is often present in the frames from artificially inserted labels, subtitles, news-ticker text, as well as from general scene text. No labels or annotations are provided for this dataset. Figure 8.1 shows some examples of frames from this dataset.

Offline pre-processing. This search application consists of two parts – a one-time, offline pre-processing stage to build a searchable representation of the video data, and an online search process to query this representation instantly and retrieve the results.

The offline pre-processing stage is illustrated in the schematic of Figure 8.2.

To reduce the number of frames that must be processed, we define a smaller number keyframes that are processed to represent the full set of frames. Keyframes are extracted from the video, at a rate of one per second and clustered based on similarity, resulting in 2.3 million keyframes.

Each keyframe is processed by the text spotting system of Chapter 5, acting in retrieval mode (Section 5.5.2), resulting in a vector per keyframe from Equation 5.6. As a reminder, these vectors represent the probability of a word occurring in the keyframe from the 90k words of 90kDict (Section 3.3), and can be viewed as a descriptor for the keyframe image.

The keyframes and descriptors are stored in a database, along with the bounding boxes for all the detections $\mathcal{B}_f$ used to generate the descriptor (this includes multiple detections of the same word within the image). An inverted index [Manning et al., 2008] is constructed on the descriptors.

Processing the 2.3 million keyframes took 2 weeks on a single machine with 16 physical CPU cores and 4 commodity GPUs.

Online search process. To query the system, a user provides a search term from the 90kDict (if the search term supplied is not present in the 90kDict, no results are returned) and a score for each keyframe in the database is computed (Equation 5.7). This can be computed very efficiently using the inverted index data structure. Keyframes with a score above a certain threshold are ranked in descending score order and displayed to the user. Due to the efficiency of the inverted index, this process can be completed in a matter of milliseconds.

The online system is wrapped in a web application and is accessible at time of writing at `http://zeus.robots.ox.ac.uk/textsearch/`. Figure 8.3 shows screenshots from the web application. When a user selects a particular result from the ranked list on the search results page (Figure 8.3 (b)), the result page is displayed which shows the keyframe, as well as the localisation of the query word that was stored and the metadata of the broadcast that the keyframe is from (Figure 8.3 (c)). It is worth noting that the online system does not require the large computational requirements of the offline pre-processing stage.

This application works exceptionally well, with Figure 8.4 showing some example retrieval results from our visual search engine. While we do not have groundtruth annotations to quantify the retrieval performance on this dataset, we measure the precision at 100 (P@100) for the test queries in Figure 8.4, showing a P@100 of 100% for the queries `hollywood` and `boris johnson`, and 93% for `vision`.

8.2 Releases

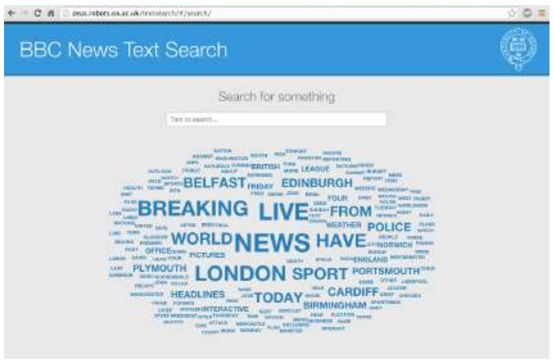
In this section we describe various software releases, aimed at helping enable future applications and development in text spotting.

Character-based Classifiers. We have released the convolutional neural network models from Chapter 4. The text/no-text classifier, case-insensitive character classifier, case-sensitive character classifier, and the bigram classifiers for the ICDAR

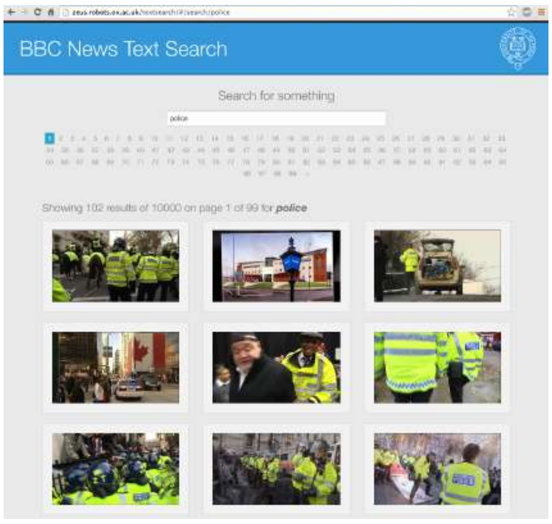
2003 and SVT test set bigrams are included in this release. The full details of these classifiers can be found in Section 4.1. The MATLAB[1] code allows the recreation of Table 4.1 as well as being able to run the classifiers in sliding window mode to create the detection and character class maps used by the pipeline. This software release also includes the character data used to train and test on, which includes the characters mined from Flickr (Section 3.2.1).

Word-based Models. We have also released the word-based recognition models used in Chapter 5 and Chapter 6. In particular, we package the dictionary classification model (Section 5.4.1), which gives state-of-the-art word recognition accuracy on all standard datasets, the character sequence model to allow unconstrained text recognition (Section 6.1.1), and the N-gram detection model (Section 6.1.2). The models are in the form of MATLAB code using the `MatConvNet` toolbox [Vedaldi and Lenc, 2014]. As mentioned in Section 3.3, we have also released the 9 million synthetic word images, Synth90k, used for training these models at `http://www.robots.ox.ac.uk/~vgg/data/text/`.

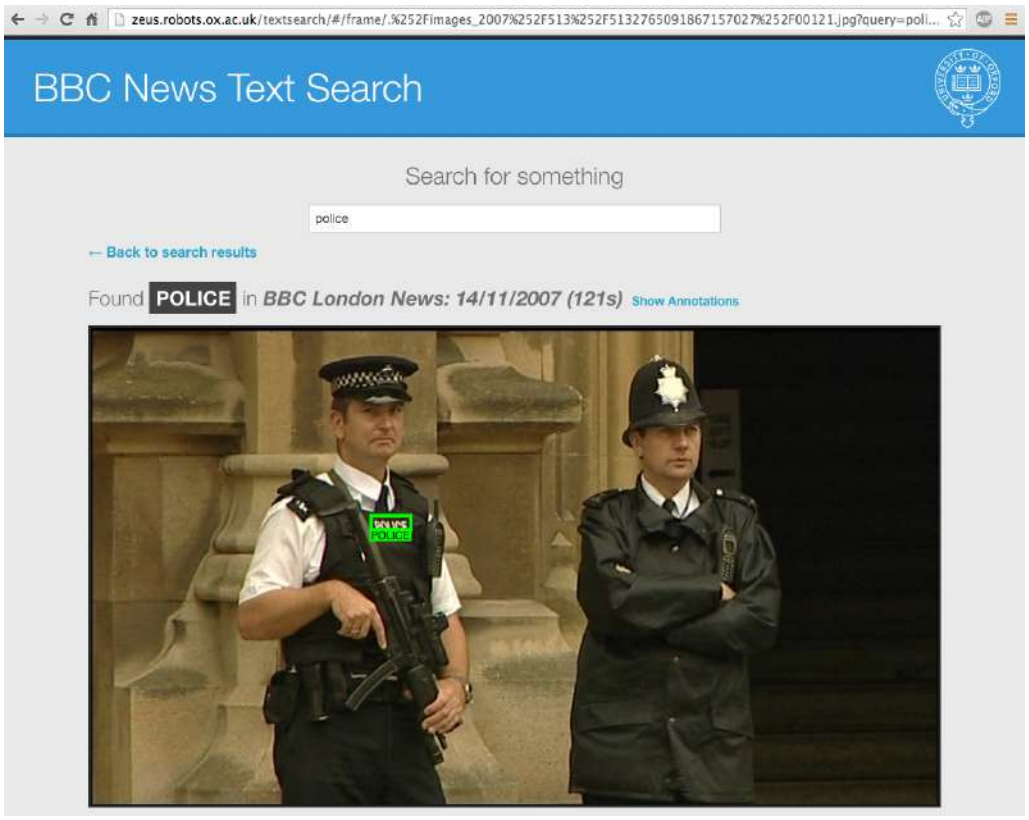
[1]`http://uk.mathworks.com/products/matlab/`


BBC News Text Search
Search for something
Text to search...
BREAKING LIVE FROM POLICE WORLD NEWS HAVE LONDON SPORT BELFAST EDINBURGH PLYMOUTH PORTSMOUTH TODAY CARDIFF HEADLINES BIRMINGHAM

(a)



(b)


zeus.robots.ox.ac.uk/textsearch/#/frame/.%252Fimages_2007%252F513%252F5132765091867157027%252F00121.jpg?query=poli...
BBC News Text Search
Search for something
police
← Back to search results
Found POLICE in BBC London News: 14/11/2007 (121s) Show Annotations
POLICE

(c)

Figure 8.3: Screenshots of the web application interface allowing the searching for text in BBC News footage. (a) Home page with search bar and a clickable word cloud, where the size of the word represents its frequency in the data. (b) The search results page for the search term `police`. (c) The results page for a particular retrieved frame showing the green bounding box localisation of the search term within the frame.

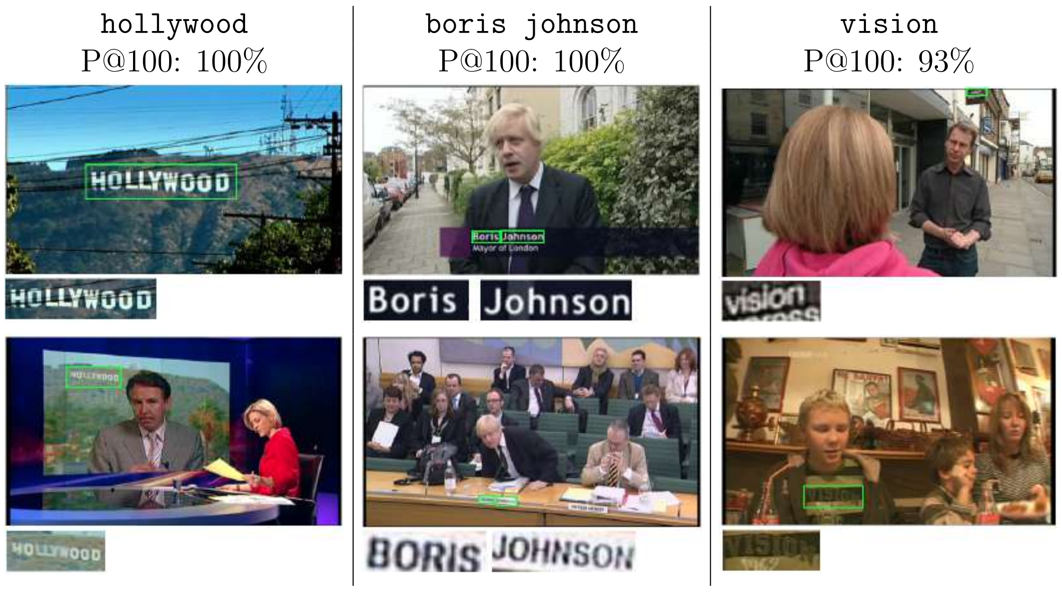


Figure 8.4: The top two retrieval results for three queries on our BBC News dataset – `hollywood`, `boris johnson`, and `vision`. The frames and associated videos are retrieved from 5k hours of BBC video. We give the precision at 100 (P@100) for these queries, equivalent to the first page of results of our web application.

Chapter 9

Summary and Future Work

In this chapter we summarise the achievements of the work presented in this thesis, and also discuss possible directions for future research.

9.1 Achievements

In this thesis we have proposed several methods for improving text spotting based on advancing the capabilities of deep learning frameworks, contributed to the public data available for this problem, as well as proposing a general method for speeding up CNNs and developing a practical application of text spotting.

In Chapter 3 we tackled the problem of data generation for text spotting. We approached this in two ways, first by automatic mining of real world data, and then by the synthetic generation of data. In Section 3.2 we showed that a simple weak baseline text spotting system can be used to crawl a photo sharing website and produce extra word and character level data samples automatically in a weakly supervised manner. We then made another contribution in Section 3.3 by presenting a synthetic data generation engine. This engine can generate word images with a huge variation in visual style, mimicking real-world text spotting data. The data produced by this process is realistic and variable enough to be used in place of real-world training data, to train text spotting systems that are applicable to real-world test data. We have made a major release of this data (9 million images) for the use of future work by others.

We proposed a full, end-to-end text spotting pipeline in Chapter 4. This builds upon the conventional character detection and classification based pipeline, however, used an unconventional CNN architecture including feature sharing between four classification tasks. The use of CNNs and feature sharing combined resulted in character classifiers that exceeded previous state-of-the-art [Alsharif and Pineau, 2014] in accuracy. The end-to-end text spotting system outperformed all existing methods and is only exceeded in F-score by our next contribution in this thesis.

Chapter 5 presented our second end-to-end text spotting pipeline, developed to explore the advantages of whole word detection and recognition. The fast region proposal based detection stage achieves very high word detection recall, with precision boosted by a filtering, regression, and non-maximal suppression stage which takes into account the text recognition results. The text recognition system is based on a deep CNN that takes the whole word image as input, and poses recognition as a classification problem in a huge dictionary. This text recognition system outperforms all previous methods (including the previous state of the art set by Chapter 4 and Bissacco et al., 2013a), and surprisingly is trained purely on our synthetic data. The end-to-end text spotting pipeline is fast and accurate, greatly outperforming all previous results in text spotting, and more than doubling the precision over Mishra et al., 2013 in large scale image retrieval by text. A final example of text spotting this system is shown in Figure 9.1.

We then focussed on a more neglected problem in natural scene text recognition, that of unconstrained text recognition, where a dictionary, lexicon, or constraining language model is not used. Chapter 6 described the system, comprised of two complementary character prediction CNNs, the first predicting the sequence of char-

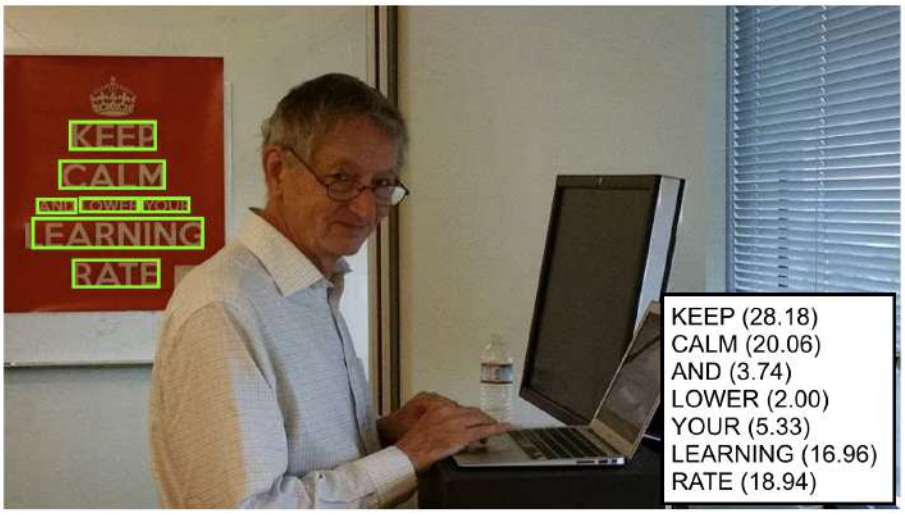


Figure 9.1: An example text spotting result using the pipeline of Chapter 5. The bounding boxes of each word found are shown in green, with the recognised words given in typographical layout order together with the score assigned by the pipeline in brackets.

acters directly, the second predicting the presence of character N-grams. The two architectures are combined in a single formulation and jointly optimised by defining and back-propagating a structured output loss. The resulting text recognition model performs very well and is shown to generalise to vocabulary and completely random text that has never been seen during training. There are no previous unconstrained scene text recognition algorithms to compare directly to, but once constrained by a dictionary or lexicon we match state of the art (set by Chapter 5).

Chapter 7 addressed the problem of speeding up CNNs, contributing to the efficient application of this inference method for a host of general computer vision problems beyond text spotting. We defined two frameworks for approximating CNNs, drawing on the work of Rigamonti et al., 2013 for creating separable filter bases. It is shown that by jointly optimising the approximations of the convolutional layers together, we can achieve a 4.5$\times$ speedup of the CNN from Chapter 4 with minimal drop in accuracy. This has inspired other work since its publication [Jaderberg et al., 2014c] including the successful work of Lebedev et al., 2014 who perform full tensor decomposition approximations for speedups.

Finally, in Chapter 8, we presented a practical application of text spotting. A web application front end was developed for searching through vast video archives for visual instances of user-given text, using the system of Chapter 5 to index the frames of thousands of hours of video. We also detailed the software releases made, and hope that through this software and data, new applications and advances can be discovered to move forward text spotting beyond this thesis.

9.2 Future Work

This section discusses some potentially promising directions for future work. These constitute both immediate through to long term visions for text spotting and deep learning.

Combined recognition models. In Chapter 5 we showed the power of performing text recognition by word classification, but this has the problem of the recognition result being constrained to the dictionary trained with. We can define models such as those from Chapter 6 to remove the dictionary constraint, but the accuracy is then much lower than language constrained model. It would therefore be interesting to pursue a combined approach, bringing together the powerful dictionary classification model of Section 5.4 and the unconstrained recognition model of Chapter 6.

A simple way to achieve this would be to add a "not in dictionary" class to the dictionary model. This would allow the model to flag when a word depicted in an image is likely to not be able to be classified by the model since it is not in the classification dictionary. When such a flag occurs, the governing system could then ignore the dictionary classification model and use the unconstrained recognition model to generate the final recognition result. A potentially more powerful way would be to incorporate the pre-trained dictionary model (Section 5.4), character sequence model (Section 6.1.1), and N-gram detection model (Section 6.1.2) in a single CRF or graph structure. This could then be trained jointly, presented with both dictionary and non-dictionary training samples to learn the optimal configuration of the three models. This more complex formulation would allow all three models to share evidence, potentially leading to more a accurate, and completely generalisable, recognition system.

Very deep architectures for text recognition. It would useful to explore further CNN architectures for the tasks in this thesis. Recently it was shown that big gains in object classification accuracy can be made by training very deep CNN architectures [Simonyan and Zisserman, 2015]. Initial experiments have shown that our own text recognition CNNs improve with depth, but this could be taken to the extreme with a similar architecture to [Simonyan and Zisserman, 2015], where only 3×3 convolution kernels and up to 19 weight layers are used.

Expanded synthetic data. There is potential to expand the synthetic data generation engine from Section 3.3. Currently the distributions driving the random process of data generation are hand set and arbitrarily defined. It would be interesting to perform parameter exploration to tune the distributions, potentially even by posing parameter exploration as a game played between the synthetic data engine and a binary classification CNN trained for real-world/synthetic data classification (similar in spirit to Goodfellow et al., 2014).

Continual self-supervised mining. Building on the work of Chapter 4, Chapter 5, and the data mining technique of Section 3.2, it would be interesting to continue classifier learning by self-supervision. A text spotting system could continually and automatically crawl and mine more data from the internet, using the weak supervision to generate real-world labelled data. This newly generated data could be used to continue training the classifiers and recognition models. This cycle of mining and training would allow the progressive updating of the learnt models, and could gradually mine enough real-world data to remove the reliance on synthetic data, potentially leading to more accurate recognition models.

Font classification, visual translation. Aside from text recognition, we could use our framework of synthetic data training CNN classifiers for a range of other text-related problems. For example, font identification [Chen et al., 2014] would be an area that could be improved upon – our synthetic data engine is capable of labelling the font used for each sample, potentially allowing the training of robust systems for font identification in the wild.

Another text-related problem that could be tackled with this framework is language identification. This would aim to identify the language that the word in the image belongs to. This problem could even be framed in the scenario of translating words back to English, and could incorporate research into the support for translating multiple languages into English using a single model, requiring a level of context switching conditioned on the detected language of the word image.

Adaptation to mobile systems. Due to the pervasiveness of smart phone usage, it becomes increasingly desirable to incorporate text spotting as a feature on mobile phones. The suitability and adaptation of our end-to-end text spotting pipelines for mobile phone deployment could be explored. This could be incorporated with the speedup (and memory reduction) method of Chapter 7, to allow the solution to fit in the limited memory and processing budget of these low-power mobile devices. Such a solution to mobile text spotting may exist where different parts of the processing pipeline are split between being computed locally on the phone, and remotely on a server, to achieve the fastest overall response time. This could also involve research into using a video stream rather than a single image to produce the text spotting result for the scene.

Holistic text spotting systems. Finally, as a longer term vision for the future of text spotting systems, a promising idea is to begin to move towards more holistic text spotting. By holistic we refer to merging the separate text detection and recognition stages together, and look to jointly optimise the end-to-end system, rather than each sub-system individually. This is starting to be explored with work such as Mnih et al., 2014, where attention models are integrated with recognition models, but there is much research to be done to get to the point of holistic text spotting systems.

Appendices

Appendix A

The Design of Convolutional Neural Network Architectures

Throughout this thesis, a number of convolutional neural networks (CNNs) [LeCun et al., 1998] are used successfully. While the exact details of the architectures are given, the design process and intuition behind the architectures are not. This appendix aims to summarise some general findings on best practices for designing CNNs for computer vision, and in the process to expose the design decisions and evolution of the two main architectures used in this thesis – the character recognition CNN architecture of Chapter 4, and the base architecture for the whole word recognition models used in Chapter 5 and Chapter 6.

The problem that arises when starting to design a CNN is that there are a large number of hyper parameters with respect to the architecture. These are namely: the number of convolutional layers, number of fully connected layers, number of filters in each layer, the kernel size of the filters, the type of non-linearity used, how much and what type of pooling is used, and input and featuremap normalisation schemes.

This high dimensional space of architectures cannot be fully explored for each new problem due to the long training time of a CNN. Instead, many of these design parameters must be set, often inherited from historically successful architectures, with iterations in design performed by varying the most important hyper parameters. The setting of hyper parameters generally reduces to a compromise between the number of weights, computational complexity, and model performance after training.

Evidence and general intuition for these hyper parameters will now be given, with reference to the design choices for the character recognition CNN from Chapter 4 and the word recognition CNNs from Chapter 5 and Chapter 6.

Network Structure. A CNN should start with convolutional layers. Convolutional layers exploit the fact that the same local structures can occur anywhere in an image, learning translationally equivariant responses. Convolutional layers are parameter efficient, as they integrate only local context and are good for extracting patch-level features. However, a CNN used for a task requiring global inference across the whole image (*e.g.* classification) should end with fully connected layers, which operate across the whole spatial resolution of the featuremap. In fact there is no distinction between a fully connected layer and a convolutional layer with a filter size the same as the spatial resolution of the featuremap it operates on, something we exploit for the CNN in Chapter 4. This global inference is something that might not be required for other tasks, such as semantic segmentation as shown in Long et al., 2014, and so no fully connected layers are needed.

Network Size. The size of a network is ultimately described by the the number of learnable parameters (weights and biases), which is a function of the number of layers, number of filters in each layer, and the size of the filters. Filter sizes should be large enough to capture the spatial size of the expected discriminative structures – in Chapter 4 we set the first layer convolutional filter size to 9×9 to be able to model large parts of the 24×24 character images. There is increasing evidence to show that deeper networks, with more layers, are more powerful [Simonyan and Zisserman, 2015; Goodfellow et al., 2013b], and that large filter sizes can be replaced by multiple layers of smaller filters (*e.g.* three stacked convolutional layers with 3×3 filters have the receptive field of a single convolutional layer with 7×7 filters). However, the increase in depth increases the number of parameters. For Chapter 4 we limit ourselves to relatively shallow four-layer networks in order to reduce parameters and avoid gross overfitting since we had limited training data. In Chapter 5 and Chapter 6 we had large amounts of data and could afford smaller filters with deeper networks resulting in larger numbers of parameters with minimal overfitting. In fact, initial experiments for the networks in Chapter 5 and Chapter 6 used shallower six-layer CNNs, but we found adding more layers gave higher accuracy models (with diminishing returns), producing the final eight-layer CNN architectures described. The number of filters in a layer is historically a multiple of 16 due to implementation details of `cuda-convnet` [Krizhevsky et al., 2012], and as a rule of thumb, for every $2\times$ reduction in spatial resolution from pooling, the number of filters should be doubled.

Non-linearity. A crucial part of the effectiveness of CNNs is their non-linear nature. This non-linear nature is created by inserting element-wise non-linearities after convolutional and fully connected layers. While classically a saturating non-linearity such as tanh is used, it was shown in Nair and Hinton, 2010; Krizhevsky et al., 2012; Jarrett et al., 2009 that rectified linear non-linearities are quicker to train with and produce higher accuracy models. Subsequently, the default non-linearity to use in computer vision models is a rectifier. However, alternatives such as maxout [Goodfellow et al., 2013a] can give better accuracy models than rectifiers. In Chapter 4, the use of maxout non-linearity gave around 1% improvement over rectifiers for case-insensitive accuracy, resulting in the final networks using maxout. However, maxout requires at least $2\times$ the number of filters per layer to achieve the same dimensional featuremaps – this was acceptable when training the small models of Chapter 4, but in Chapter 5 and Chapter 6 the increase in filters and computational complexity from using maxout was deemed too much compared to the minimal increase (if any) in accuracy that may have resulted by using maxout.

Pooling. Local pooling layers are used to reduce the spatial dimensionality of featuremaps, and provide spatial invariance in small neighbourhoods. Two popular pooling mechanisms are average pooling and max pooling. Max pooling most often gives higher accuracies in classification models [Jarrett et al., 2009], something which we confirmed when exploring the CNN design for Chapter 5 – max pooling consistently resulted in higher accuracy models than average pooling. In Chapter 4 we forgo the use of any pooling, as we want our CNN to be evaluated convolutionally across an entire image without any reduction in spatial resolution.

Normalisation. There are two places that normalisation can be employed – on the input images and local response normalisation (LRN) between convolutional layers. LRN has been successful in some historic architectures (*e.g.* Krizhevsky et al., 2012), however more recently it is shown that LRN does not give any improvements and can be ignored completely [Simonyan and Zisserman, 2015], so in this work we do not use LRN anywhere. However, input normalisation is an important addition to an architecture – whitening the inputs (sample mean subtraction and division by sample standard deviation) keeps the distribution of data values similar for each data sample. Input normalisation helps accuracy of models (we found a small 0.5% increase in accuracy when adding standard deviation division to the network of Chapter 5) and helps to map different datasets into the same dynamic range for generalisable models.

To summarise, while there are a large number of design choices when it comes to CNN architectures, generally *deep*, *rectifier* networks with *max pooling*, *small filters* (*e.g.* 3×3) and *input normalisation* give competitive results. While CNNs can be designed from scratch for a new problem, it is often best to start with a known successful architecture for a similar problem and iterate – the CNN from

Chapter 4 was based on that from Alsharif and Pineau, 2014, and the base CNN from Chapter 5 and Chapter 6 was based on that from Goodfellow et al., 2013b. Finally, one of the advantages of using CNNs is that from empirical experience, the trained performance of CNNs is not very sensitive to small changes in architecture (*e.g.* number of filters) if there is enough capacity to model the problem, making CNNs a robust machine learning tool.

Bibliography

Alexe, B., T. Deselaers, and V. Ferrari (2012). "Measuring the objectness of image windows". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 34.11, pp. 2189–2202.

Almazán, J., A. Gordo, A. Fornés, and E. Valveny (2014). "Word Spotting and Recognition with Embedded Attributes". In: *TPAMI*.

Alsharif, O. and J. Pineau (2014). "End-to-End Text Recognition with Hybrid HMM Maxout Models". In: *International Conference on Learning Representations*. arXiv:`1310.1811 [cs.CV]`.

Anthimopoulos, M., B. Gatos, and I. Pratikakis (2011). "Detection of artificial and scene text in images and video frames". In: *Pattern Analysis and Applications*, pp. 1–16.

Arandjelović, R. and A. Zisserman (2012). "Three things everyone should know to improve object retrieval". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Bengio, Y. (2009). "Learning deep architectures for AI". In: *Foundations and trends® in Machine Learning* 2.1, pp. 1–127.

Bissacco, A., M. Cummins, Y. Netzer, and H. Neven (2013). "PhotoOCR: Reading Text in Uncontrolled Conditions". In: *Proceedings of the International Conference on Computer Vision*.

Bissacco, A., M. Cummins, Y. Netzer, and H. Neven (2013). "PhotoOCR: Reading Text in Uncontrolled Conditions". In: *International Conference of Computer Vision*.

Boykov, Y. and M. P. Jolly (2001). "Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images". In: *Proceedings of the 8th International Conference on Computer Vision, Vancouver, Canada*. Vol. 2, pp. 105–112.

Breiman, L. (2001). "Random Forests". In: *Machine Learning* 45.1, pp. 5–32.

Campos, T. de, B. R. Babu, and M. Varma (2009). "Character recognition in natural images". In: *VISAPP*.

Chatfield, K. and A. Zisserman (2013). "VISOR: Towards on-the-fly large-scale object category retrieval". In: *Asian Conference of Computer Vision – ACCV 2012*. Springer Berlin Heidelberg, pp. 432–446.

Chen, G., J. Yang, H. Jin, J. Brandt, E. Shechtman, A. Agarwala, and T. Han (2014). "Large-Scale Visual Font Recognition". In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*.

Chen, H., S. Tsai, G. Schroth, D. Chen, R. Grzeszczuk, and B. Girod (2011). "Robust text detection in natural images with edge-enhanced Maximally Stable Extremal Regions". In: *Proc. International Conference on Image Processing (ICIP)*, pp. 2609–2612.

Chen, P. and D. Suter (2004). "Recovering the Missing Components in a Large Noisy Low-Rank Matrix: Application to SFM". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.8, pp. 1051–1063.

Chen, X. and A. L. Yuille (2004). "Detecting and reading text in natural scenes". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Vol. 2, pp. II–366.

Cheng, M.-M., Z. Zhang, W.-Y. Lin, and P. Torr (2014). “BING: Binarized normed gradients for objectness estimation at 300fps”. In: *IEEE CVPR.*

Chetlur, S., C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer (2014). “cudnn: Efficient primitives for deep learning”. In: *arXiv preprint arXiv:1410.0759.*

Ciresan, D. C., U. Meier, and J. Schmidhuber (2012). “Multi-column deep neural networks for image classification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3642–3649.

Coates, A., A. Y. Ng, and H. Lee (2011). “An Analysis of Single-Layer Networks in Unsupervised Feature Learning”. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics.*

Coltheart, M., K. Rastle, C. Perry, R. Langdon, and J. Ziegler (2001). “DRC: a dual route cascaded model of visual word recognition and reading aloud”. In: *Psychological review* 108.1, pp. 204–256.

Cristianini, N. and J. Shawe-Taylor (2000). *An Introduction to Support Vector Machines.* Cambridge University Press.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). “ImageNet: A Large-Scale Hierarchical Image Database”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

Denil, M., B. Shakibi, L. Dinh, and N. de Freitas (2013). “Predicting Parameters in Deep Learning”. In: *Advances in Neural Information Processing Systems*, pp. 2148–2156.

Denton, E., W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus (2014). “Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation”. In: *arXiv preprint arXiv:1404.0736.*

Dollár, P. and C. L. Zitnick (2013). “Structured forests for fast edge detection”. In: *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE, pp. 1841–1848.

Dollár, P. and C. L. Zitnick (2014). “Fast edge detection using structured forests”. In: *arXiv preprint arXiv:1406.5549*.

Dollár, P., S. Belongie, and P. Perona (2010). “The Fastest Pedestrian Detector in the West.” In: *BMVC*. Vol. 2. 3, p. 7.

Dollár, P., R. Appel, S. Belongie, and P. Perona (2014). “Fast feature pyramids for object detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Dutta, S., N. Sankaran, K. Sankar, and C. Jawahar (2012). “Robust recognition of degraded documents using character n-grams”. In: *Document Analysis Systems (DAS), International Workshop on*. IEEE, pp. 130–134.

Epshtein, B., E. Ofek, and Y. Wexler (2010). “Detecting text in natural scenes with stroke width transform”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 2963–2970.

Erhan, D., Y. Bengio, A. Courville, and P. Vincent (2009). *Visualizing Higher-Layer Features of a Deep Network*. Tech. rep. 1341. University of Montreal.

Everingham, M., L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman (2010). “The PASCAL Visual Object Classes (VOC) Challenge”. In: *International Journal of Computer Vision* 88.2, pp. 303–338.

Farabet, C., Y. LeCun, K. Kavukcuoglu, E. Culurciello, B. Martini, P. Akselrod, and S. Talay (2011). “Large-scale FPGA-based convolutional networks”. In: *Machine Learning on Very Large Data Sets*.

Farabet, C., C. Couprie, L. Najman, and Y. LeCun (2012). “Scene parsing with multiscale feature learning, purity trees, and optimal covers”. In: *arXiv preprint arXiv:1202.2160*.

Felzenszwalb, P. and D. Huttenlocher (2005). "Pictorial Structures for Object Recognition". In: *International Journal of Computer Vision* 61.1.

Felzenszwalb, P. F., R. B. Grishick, D. McAllester, and D. Ramanan (2010). "Object Detection with Discriminatively Trained Part Based Models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

Fischer, A., A. Keller, V. Frinken, and H. Bunke (2010). "HMM-based word spotting in handwritten documents using subword models". In: *Pattern recognition (icpr), 2010 20th international conference on.* IEEE, pp. 3416–3419.

Freund, Y. and R. Schapire (1997). "A decision theoretic generalisation of online learning". In: *Computer and System Sciences* 55.1, pp. 119–139.

Friedman, J., T. Hastie, and R. Tibshirani (2000). "Additive Logistic Regression: a Statistical View of Boosting". In: *Annals of Statistics* 28.2, pp. 337–407.

Frinken, V., A. Fischer, R Manmatha, and H. Bunke (2012). "A novel word spotting method based on recurrent neural networks". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 34.2, pp. 211–224.

Fukushima, K. (1988). "Neocognitron: A hierarchical neural network capable of visual pattern recognition". In: *Neural networks* 1.2, pp. 119–130.

Girshick, R. B., J. Donahue, T. Darrell, and J. Malik (2014). "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

Goel, V., A. Mishra, K. Alahari, and C. V. Jawahar (2013). "Whole is greater than sum of parts: Recognizing scene text words". In: *ICDAR*, pp. 398–402.

Gomez, L. and D. Karatzas (2013). "Multi-script text extraction from natural scenes". In: *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on.* IEEE, pp. 467–471.

Goodfellow, I. J., D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio (2013). "Maxout networks". In: *arXiv preprint arXiv:1302.4389.*

Goodfellow, I. J., Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet (2013). "Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks". In: *arXiv:1312.6082*.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). "Generative adversarial nets". In: *Advances in Neural Information Processing Systems*, pp. 2672–2680.

Gordo, A. (2014). "Supervised mid-level features for word image representation". In: *ArXiv e-prints*. arXiv:`1410.5224 [cs.CV]`.

Graves, A. and J. Schmidhuber (2005). "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". In: *Neural Networks* 18.5, pp. 602–610.

Graves, A., M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber (2009). "A novel connectionist system for unconstrained handwriting recognition". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 31.5, pp. 855–868.

He, K., X. Zhang, S. Ren, and J. Sun (2014). "Spatial pyramid pooling in deep convolutional networks for visual recognition". In: *Computer Vision–ECCV 2014*. Springer, pp. 346–361.

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2012). "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580.

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Huang, W., Y. Qiao, and X. Tang (2014). "Robust Scene Text Detection with Convolution Neural Network Induced MSER Trees". In: *Computer Vision–ECCV 2014*. Springer, pp. 497–511.

Hubel, D. H. and T. N. Wiesel (1962). “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. In: *The Journal of Physiology* 160, pp. 106–154.

Iandola, F., M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer (2014). “DenseNet: Implementing Efficient ConvNet Descriptor Pyramids”. In: *arXiv preprint arXiv:1404.1869*.

Jaderberg, M., A Vedaldi, and A. Zisserman (2014). “Deep Features for Text Spotting”. In: *European Conference on Computer Vision*.

Jaderberg, M., K. Simonyan, A. Vedaldi, and A. Zisserman (2014). “Synthetic Data and Artificial Neural Networks for Natural Scene Text Recognition”. In: *NIPS Deep Learning Workshop 2014*.

Jaderberg, M., K. Simonyan, A. Vedaldi, and A. Zisserman (2015). “Reading Text in the Wild with Convolutional Neural Networks”. In: *IJCV*.

Jaderberg, M., A. Vedaldi, and A. Zisserman (2014). “Speeding up Convolutional Neural Networks with Low Rank Expansions”. In: *BMVC 2014*.

Jaderberg, M., K. Simonyan, A. Vedaldi, and A. Zisserman (2015). “Deep Structured Output Learning for Unconstrained Text Recognition”. In: *ICLR 2015*.

Jarrett, K., K. Kavukcuoglu, M Ranzato, and Y. LeCun (2009). “What is the best multi-stage architecture for object recognition?” In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, pp. 2146–2153.

Jia, Y. (2013). *Caffe: An Open Source Convolutional Architecture for Fast Feature Embedding*. `http://caffe.berkeleyvision.org/`.

Jin, J., A. Dundar, and E. Culurciello (2014). “Flattened Convolutional Neural Networks for Feedforward Acceleration”. In: *arXiv preprint arXiv:1412.5474*.

Karatzas, D., F. Shafait, S. Uchida, M. Iwamura, S. R. Mestre, J. Mas, D. F. Mota, J. Almazan, L. P. de las Heras, et al. (2013). “ICDAR 2013 Robust Reading Competition”. In: *ICDAR*. IEEE, pp. 1484–1493.

Kavukcuoglu, K., P. Sermanet, Y. Boureau, K. Gregor, M. Mathieu, and Y. LeCun (2010). “Learning Convolutional Feature Hierarchies for Visual Recognition.” In: *NIPS*. Vol. 1. 2, p. 5.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*, pp. 1106–1114.

Lebedev, V., Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky (2014). “Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition”. In: *arXiv preprint arXiv:1412.6553*.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel (1989). “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4, pp. 541–551.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

LeCun, Y. and C. Cortes (1998). *The MNIST database of handwritten digits.*

Lee, D. C., Q. Ke, and M. Isard (2010). “Partition Min-Hash for Partial Duplicate Image Discovery”. In: *Proceedings of the European Conference on Computer Vision.*

Lee, H., R. Grosse, R. Ranganath, and A. Y. Ng (2009). “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations”. In: *Proceedings of the 26th Annual International Conference on Machine Learning.* ACM, pp. 609–616.

Li, J. and J. Z. Wang (2003). “Automatic linguistic indexing of pictures by a statistical modeling approach”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence.*

Long, J., E. Shelhamer, and T. Darrell (2014). “Fully convolutional networks for semantic segmentation”. In: *arXiv preprint arXiv:1411.4038*.

Lucas, S. (2005). “ICDAR 2005 text locating competition results”. In: *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on.* IEEE, pp. 80–84.

Lucas, S. M., A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young (2003). “ICDAR 2003 robust reading competitions”. In: *2013 12th International Conference on Document Analysis and Recognition.* Vol. 2. IEEE Computer Society, pp. 682–682.

Mamalet, F. and C. Garcia (2012). “Simplifying convnets for fast learning”. In: *Artificial Neural Networks and Machine Learning–ICANN 2012.* Springer, pp. 58–65.

Manmatha, R, C. Han, and E. M. Riseman (1996). “Word spotting: A new approach to indexing handwriting”. In: *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on.* IEEE, pp. 631–637.

Manning, C. D., P. Raghavan, and H. Schütze (2008). *Introduction to Information Retrieval.* Cambridge University Press.

Markoff, J. (2013). *Device From Israeli Start-Up Gives the Visually Impaired a Way to Read.* Ed. by N. Y. Times. URL: `http://www.nytimes.com/2013/06/04/science/israeli-start-up-gives-visually-impaired-a-way-to-read.html`.

Matas, J., O. Chum, M. Urban, and T. Pajdla (2002). “Robust Wide Baseline Stereo from Maximally Stable Extremal Regions”. In: *Proceedings of the British Machine Vision Conference*, pp. 384–393.

Mathieu, M., M. Henaff, and Y. LeCun (2013). “Fast Training of Convolutional Networks through FFTs”. In: *CoRR* abs/1312.5851.

Mishra, A., K. Alahari, C. Jawahar, et al. (2012). "Scene text recognition using higher order language priors". In: *BMVC 2012-23rd British Machine Vision Conference*.

Mishra, A., K. Alahari, and C. Jawahar (2013). "Image Retrieval using Textual Cues". In: *Computer Vision (ICCV), 2013 IEEE International Conference on*. IEEE, pp. 3040–3047.

Mnih, V., N. Heess, A. Graves, et al. (2014). "Recurrent models of visual attention". In: *Advances in Neural Information Processing Systems*, pp. 2204–2212.

Nair, V. and G. E. Hinton (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the International Conference on Machine Learning*.

Neumann, L. and J. Matas (2010). "A method for text localization and recognition in real-world images". In: *Proceedings of the Asian Conference on Computer Vision*. Springer, pp. 770–783.

Neumann, L. and J. Matas (2011). "Text Localization in Real-World Images Using Efficiently Pruned Exhaustive Search". In: *Proc. ICDAR*. IEEE, pp. 687–691.

Neumann, L. and J. Matas (2012). "Real-Time Scene Text Localization and Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Neumann, L. and J. Matas (2013). "Scene Text Localization and Recognition with Oriented Stroke Detection". In: *Proceedings of the International Conference on Computer Vision*, pp. 97–104.

Novikova, T., O. Barinova, P. Kohli, and V. Lempitsky (2012). "Large-lexicon attribute-consistent text recognition in natural images". In: *Proceedings of the European Conference on Computer Vision*. Springer, pp. 752–765.

Oquab, M., L. Bottou, I. Laptev, and J. Sivic (2014). “Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks”. In: *Computer Vision and Pattern Recognition (CVPR).*

Otsu, N. (1979). “A threshold selection method from gray-level histograms”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1, pp. 62–66.

Ozuysal, M., P. Fua, and V. Lepetit (2007). “Fast Keypoint Recognition in Ten Lines of Code”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

Perronnin, F., Y. Liu, J. Sánchez, and H. Poirier (2010). “Large-Scale Image Retrieval with Compressed Fisher Vectors”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

Pfister, T., K. Simonyan, J. Charles, and A. Zisserman (2014). “Deep Convolutional Neural Networks for Efficient Pose Estimation in Gesture Videos”. In: *Asian Conference on Computer Vision (ACCV).*

Posner, I., P. Corke, and P. Newman (2010). “Using Text-Spotting to Query the World”. In: *IROS.*

Quack, T. (2009). “Large scale mining and retrieval of visual data in a multimodal context.” PhD thesis. ETH Zurich, pp. 1–223.

Quack, T. (2009). “Large scale mining and retrieval of visual data in a multimodal context.” PhD thesis. ETH Zurich, pp. 1–223. ISBN: 978-3-86628-253-7.

Rath, T. and R. Manmatha (2007). “Word spotting for historical documents”. In: *IJDAR* 9.2-4, pp. 139–152.

Rigamonti, R., M. A. Brown, and V. Lepetit (2011). “Are sparse representations really relevant for image classification?” In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on.* IEEE, pp. 1545–1552.

Rigamonti, R., A. Sironi, V. Lepetit, and P. Fua (2013). “Learning separable filters”. In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE, pp. 2754–2761.

Rodriguez-Serrano, J. A., F. Perronnin, and F. Meylan (2013). “Label embedding for text recognition”. In: *Proceedings of the British Machine Vision Conference*.

Rosenblatt, F. (1958). “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain”. In: *Psychological Review* 65.6, pp. 386–408.

Roth, S. and M. J. Black (2005). “Fields of Experts: A Framework for Learning Image Priors”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, San Diego*. Vol. 2, pp. 860–867.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). “Learning representations by back-propagating errors”. In: *Nature* 323.6088, pp. 533–536.

Russel, S., P. Norvig, et al. (1994). “Artificial Intelligence: A Modern Approach, 1995”. In: *Cited on*, p. 20.

Salakhutdinov, R. and G. E. Hinton (2009). “Deep boltzmann machines”. In: *International Conference on Artificial Intelligence and Statistics*, pp. 448–455.

Sermanet, P., D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun (2013). “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks”. In: *arXiv preprint arXiv:1312.6229*.

Shahab, A., F. Shafait, and A. Dengel (2011). “ICDAR 2011 robust reading competition challenge 2: Reading text in scene images”. In: *Proc. ICDAR*. IEEE, pp. 1491–1496.

Simard, P. Y., D. Steinkraus, and J. C. Platt (2003). “Best practices for convolutional neural networks applied to visual document analysis”. In: *2013 12th International Conference on Document Analysis and Recognition*. Vol. 2. IEEE Computer Society, pp. 958–958.

Simonyan, K., A. Vedaldi, and A. Zisserman (2013). “Deep Fisher Networks and Class Saliency Maps for Object Classification and Localisation”. In: *ILSVRC workshop.* URL: `http://image-net.org/challenges/LSVRC/2013/slides/ILSVRC_az.pdf`.

Simonyan, K. and A. Zisserman (2014). “Two-stream convolutional networks for action recognition in videos”. In: *Advances in Neural Information Processing Systems*, pp. 568–576.

Simonyan, K. and A. Zisserman (2015). “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *ICLR 2015.*

Song, H. O., S. Zickler, T. Althoff, R. Girshick, M. Fritz, C. Geyer, P. Felzenszwalb, and T. Darrell (2012). “Sparselet models for efficient multiclass object detection”. In: *Computer Vision–ECCV 2012.* Springer, pp. 802–815.

Song, H. O., T. Darrell, and R. B. Girshick (2013). “Discriminatively activated sparselets”. In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 196–204.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2014). “Going deeper with convolutions”. In: *arXiv preprint arXiv:1409.4842.*

Taigman, Y., M. Yang, M. Ranzato, and L. Wolf (2014). “Deep-Face: Closing the Gap to Human-Level Performance in Face Verification”. In: *IEEE CVPR.*

Tompson, J. J., A. Jain, Y. LeCun, and C. Bregler (2014). “Joint training of a convolutional network and a graphical model for human pose estimation”. In: *Advances in Neural Information Processing Systems*, pp. 1799–1807.

Torralba, A., K. P. Murphy, and W. T. Freeman (2004). “Sharing features: efficient boosting procedures for multiclass object detection”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Washington, DC*, pp. 762–769.

Toshev, A. and C. Szegedy (2013). “DeepPose: Human Pose Estimation via Deep Neural Networks”. In: *arXiv preprint arXiv:1312.4659.*

Uijlings, J. R., K. E. van de Sande, T. Gevers, and A. W. Smeulders (2013). “Selective search for object recognition”. In: *International journal of computer vision* 104.2, pp. 154–171.

Vanhoucke, V., A. Senior, and M. Z. Mao (2011). “Improving the speed of neural networks on CPUs”. In: *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop.*

Vedaldi, A. and K. Lenc (2014). “MatConvNet – Convolutional Neural Networks for MATLAB”. In: *CoRR* abs/1412.4564.

Viola, P. and M. Jones (2001). “Rapid Object Detection using a Boosted Cascade of Simple Features”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 511–518.

Wang, J., J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong (2010). “Locality-constrained Linear Coding for Image Classification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

Wang, K., B. Babenko, and S. Belongie (2011). “End-to-end scene text recognition”. In: *Proceedings of the International Conference on Computer Vision.* IEEE, pp. 1457–1464.

Wang, T., D. J Wu, A. Coates, and A. Y. Ng (2012). “End-to-end text recognition with convolutional neural networks”. In: *ICPR.* IEEE, pp. 3304–3308.

Weinman, J. J., Z. Butler, D. Knoll, and J. Feild (2014). “Toward Integrated Scene Text Reading”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 36.2, pp. 375–387. ISSN: 0162-8828. DOI: `10.1109/TPAMI.2013.126`.

Wickelgran, W. A. (1969). “Context-sensitive coding, associative memory, and serial order in (speech) behavior.” In: *Psychological Review* 76.1, p. 1.

Yao, C., X. Bai, B. Shi, and W. Liu (2014). “Strokelets: A learned multi-scale representation for scene text recognition”. In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, pp. 4042–4049.

Yi, C. and Y. Tian (2011). “Text string detection from natural scenes by structure-based partition and grouping”. In: *Image Processing, IEEE Transactions on* 20.9, pp. 2594–2605.

Yin, X.-C., X. Yin, and K. Huang (2013). “Robust Text Detection in Natural Scene Images”. In: *CoRR* abs/1301.2628.

Zitnick, C. L. and P. Dollár (2014). “Edge boxes: Locating object proposals from edges”. In: *Computer Vision–ECCV 2014*. Springer, pp. 391–405.