University of Otago

Department of Computer Science

COSC490 Project Report

Stacked Hourglass Networks for Historical Document Analysis

Author:
Dr. Hannah
Clark-Younger
(749113)

Supervisor(s):
Dr. Steven Mills
Dr. Lech Szymanski

October 8, 2018

Abstract

The Marsden Collection is an important source of great interest to historians of New Zealand, but most of the pages are untranscribed and therefore automated search is impossible. Stacked hourglass networks are a new convolutional neural network architecture that has been shown to perform well on tasks that are relevantly similar to keyword search – they are capable of precisely locating semantic parts of objects (in the case of keyword search, the objects are words and the semantic parts are characters). I implemented a stacked hourglass network that I trained on simplified data generated out of individual characters drawn from the Marsden Collection. It learned to not just identify, but locate the characters in the image. Despite training on simplified data, the network can also correctly identify letters in unseen documents with 59% accuracy, which rises to 80% when considering the top 5 hypotheses.

1 Introduction

The aim of this project is to implement a stacked hourglass network, which is a particular kind of deep convolutional neural network, that can recognise and locate individual characters in images of handwritten historical documents. This is in pursuit of the ultimate goal of leveraging this information to build a tool that historians can use to search these handwritten documents for keywords of interest.

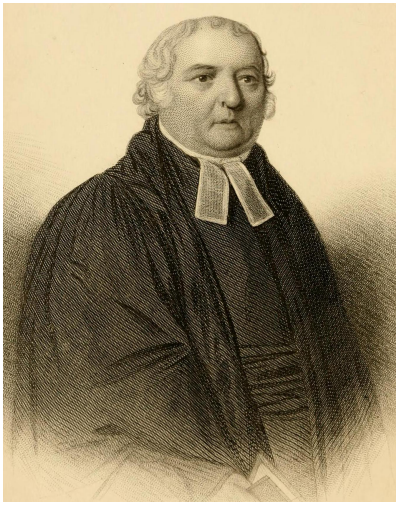

Figure 1: Samuel Marsden

1.1 Marsden Collection

The Hocken Collections of the University of Otago Library house the Marsden Collection: a large set of letters and journal entries written by Samuel Marsden (Fig. 1). Marsden (1765–1838) was an Anglican priest and one of the earliest Christian missionaries to travel to New Zealand. He is often credited as having introduced Christianity to New Zealand and is thus a very important figure in New Zealand's early European

history. The Marsden Collection, therefore, provides a rich and important historical source, as Marsden recorded in great detail his perspective on the events of that time. These, like many other primary sources of interest to historians, are handwritten and largely untranscribed, which means that researchers are unable to search them for keywords of interest (see Fig. 2 for some sample pages from this collection).

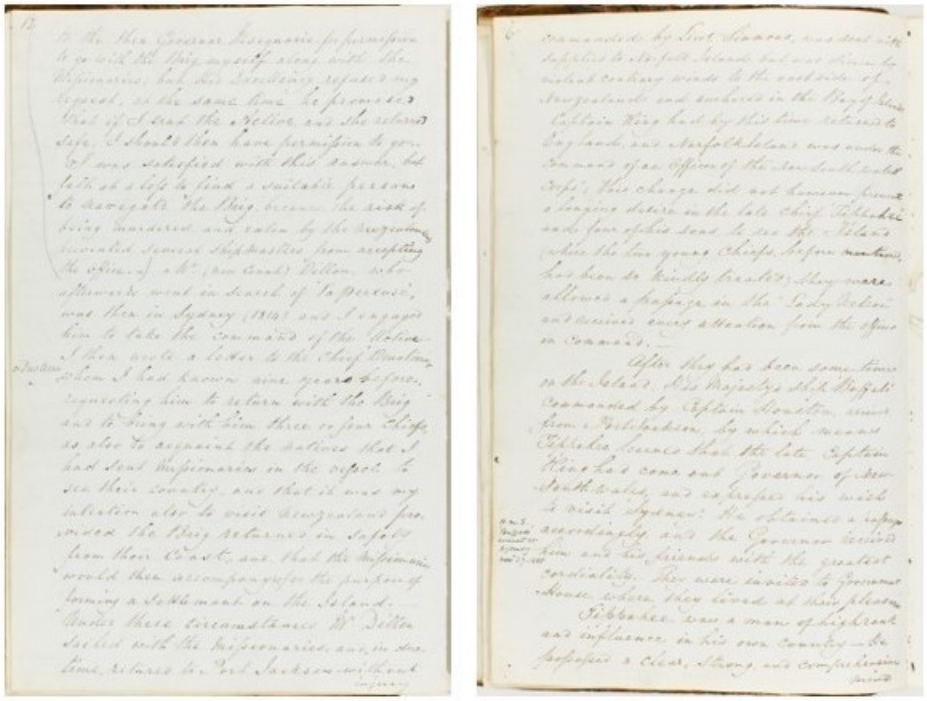

Figure 2: Sample pages from the Marsden Collection

One option would be to transcribe the documents, as then it would be possible to apply a standard search to the documents, using traditional algorithms. However, manual transcription incurs a large cost, and is often quite inaccurate as it is subject to human error. Automated transcribers, such as those using Optical Character Recognition (OCR) techniques, perform poorly on handwritten text like this (see [1] for a comparison of the performance of some state-of-the-art automated transcribers on the market).

Automated transcription requires that every character on the page (including word delimiters, such as spaces) be recognised to a high level of accuracy in order to produce a document that closely matches every word, in the correct order. However, this level of detail and accuracy is not required for keyword

search, which can be treated as a classification task: each pages either contains, or does not contain, the given keyword. This means that the problem may lend itself well to being solved by Machine Learning classification techniques – specifically, the use of deep Convolutional Neural Networks (CNNs), which have shown particularly impressive results in image classification tasks. A small set (1096 pages) of the Marsden documents have been transcribed; enough to be used as a training set for this task.

The Marsden documents form a nice dataset for initial investigations into handwriting recognition, because there is a decent sized corpus of pages, all with reasonably uniform handwriting. However, we have to be careful about how we measure how successful an automated search tool is. If it correctly identifies 90% of words in the documents, but that 90% includes the most common words, then it is probably useless as a search tool. Researchers are very unlikely to want to search for words like "the" or "and," and much more likely to want to search for terms like *convicts*, *parliament*, or *flora*. These words may only occur a handful of times, so the effect on the accuracy statistics of the (in)ability to identify them will be negligible, if we are not careful about how we measure accuracy. As we will see, this is not a problem for the technique I am focussing on.

1.2 Deep Convolutional Neural Networks

Artificial neural networks (see Fig. 3), inspired by the mammalian brain, have been shown to be very effective at various kinds of learning tasks, such as classifying images by their content or navigating a robot around an environment. After the initial excitement over them in the 1940s to 1960s (see [3] [4] [5]), there was a period referred to as the "dark ages" which was spurred by the discovery that, in their simple two-layer form, they are limited to learning tasks that are linearly separable [6]. However, since the conceptual introduction of the generalised delta rule in 1986, and thus the possibility of having a third (or in fact any number of) "hidden," layer(s) [7], the use and

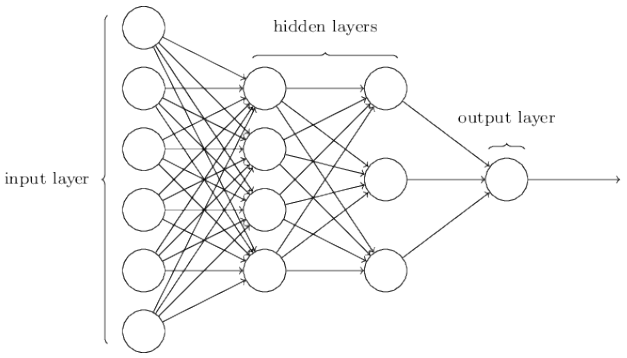

Figure 3: An Artificial Neural Network [2]

study of neural networks has revived, and indeed exploded.

Neural networks are implemented as directed graphs – nodes, or "neurons," (represented as circles in Fig. 3) with weighted connections (arrows in Fig. 3). Generally, they learn by updating the values of the weights of each connection, which produces different levels of activation in each neuron when it is triggered by some input. Depending on the type of artificial neural network, there may be more restrictions on the architecture: that is, the number and arrangement of these neurons and connections.

Research into and use of CNNs in particular has been booming, particularly on image classification tasks, since the first time a CNN outperformed all other methods on an image classification task in 1998 [8], and even more so since 2012, when a CNN outperformed the other methods on a historically difficult ImageNet dataset for the first time [9]. CNNs are neural networks with a particular kind of architecture (see Fig. 4): they have several layers, including convolutional layers and subsampling layers. The neurons in the convolutional layers respond to small patches of pixels, rather than to the entire image. This means that they can preserve the information contained in the relative location of the pixels – for example, a horizontal line consists of similar coloured pixels lined up next to one another, not randomly scattered around the image.

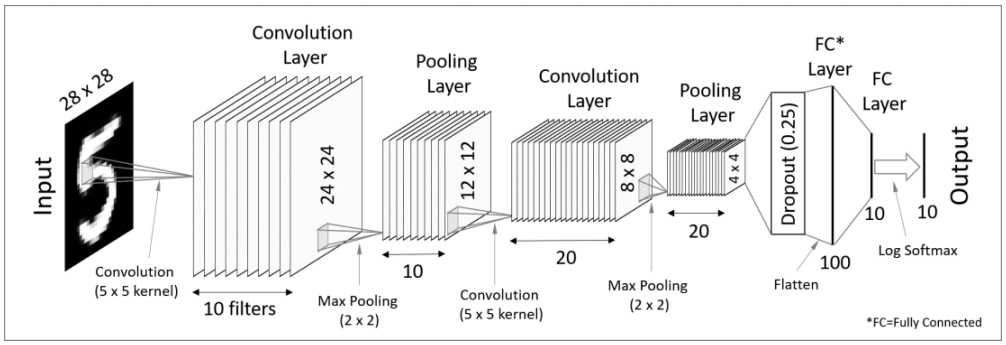

Figure 4: A Convolutional Neural Network (image from [10])

Each convolutional layer has a certain number of filters, each of which respond to a 'feature' (perhaps horizontal lines, or patches of blue pixels), and it can detect them wherever they occur in the image. Subsampling (max-pooling is commonly used) layers often follow convolutional layers, and compress or 'pool' the information extracted by the convolutional layers by keeping track of (in the case of max-pooling) only the strongest candidate for each feature in a given region, or (in the case of average-pooling) the average activation for that feature in the given region. In theory, as the information passes through the network,

the features that are detected become more complex. In the earliest layers, the features are basic lines and blobs, and the subsequent layers recombine these features to make increasingly complex features.

CNNs often use Rectified Linear Units (ReLU) as the activation function (this defines how the inputs to a given unit are combined to produce the activation of the unit). ReLU is a simple activation function, which takes the input (the sum of the units from the previous layer that are connected to the unit in question, weighted by the trainable weights associated with the respective connections, with the trainable bias of the unit added), and gives as the activation of the unit either this input, or 0, whichever is the highest. This helps to mitigate the vanishing gradient problem: in deep networks, early layers can't train quickly enough because the derivative gets significantly smaller as we backpropagate the updates, whereas when using ReLU, the slope is always 1 (when positive): so, the derivative remains significant enough to enable the weights in earlier layers to train faster than they otherwise would. All of this means that CNNs are very effective at learning useful features, detecting them, and combining them to form representations of types of objects so that they can identify them in images.

2 Background

Before I get into the details of the implementation and experimentation for this project, I present an overview of recent research, both related to the specific task of handwritten character recognition and related to the stacked hourglass architecture that I use to attempt this task.

2.1 Networks for Character Recognition

With the astounding success of CNNs in recent years, it is unsurprising that they have been applied to the task of Optical Character Recognition (OCR) [11], [12], [13]. Szymanski and Mills [1] compared some state-of-the-art OCR products on the market, but while they perform excellently on typeset text, they produce garbage when considering handwritten pages such as the Marsden documents. Some attempts at using CNNs for handwriting recognition in particular have achieved much higher accuracy than this, but are either not fully automated [14], or they rely on the words being pre-segmented, [15], or they work on very structured documents such as prescription orders written by medical doctors [16], or they use multiple separate networks for different aspects of the task, requiring large amounts of data to be annotated specifically for these individual tasks [17].

2.2 Prior work on using a CNN for searching Marsden

Preliminary work has been done on this research project; a CNN was designed, and the results of this were encouraging, but limited (see [1]). The propensity to overtrain strongly suggested it was not finding meaningful patterns that generalise to unseen pages. Over the summer (as part of a Summer Research Scholarship Project), I tried applying a Transfer Learning technique (similar to that introduced by [18]) to improve this CNN. The idea is that the CNN might perform better if it had been pre-trained on an easier task, which nonetheless was related, potentially giving it an advantage in training on the main task. I designed and implemented an auto-encoder which chopped some images up into small patches of 8×8 pixels (one handwritten letter in the data (when shrunk to the resolution the CNN is capable of managing) is approximately 8×8 – 16×16 pixels), then trained itself to reproduce the input. To do this task, the auto-encoder needs to learn some features – that is, patterns which occur often in the patches. Hopefully, the features will be those that are useful for the task of identifying words as part of a keyword search. Intuitively, in this case, these will be lines and curves in various orientations that occur most often in Marsden's handwriting. 8×8 pixels also corresponded to the size of the convolutions in the first layer of the original CNN, so this size makes sense for the second part of this technique.

I saved the weights and biases from the trained auto-encoder – that is, the model it was using to reproduce the patches. I then loaded these weights and biases into the first layer of the original CNN. The hypothesis was that it could give the CNN a head-start in learning to recognise words in the images, if the first layer could already recognise important features.

However, while the error seems to be lower (as shown in Fig. 5) for the pretrained CNN, showing some improvement over the performance of the original (un-pretrained) CNN, activation heatmaps (see Fig. 6) show no discernible improvement in the features the network was being attuned to.

I implemented a version of a 'deconvolutional' network that uses Layerwise Relevance Propagation (proposed by Bach et. al. [19]) to flow backwards from the predicted output of a network (on a particular input) back to the relative activations in the first layer that were most important in producing that output. It outputs a heatmap of the original image, with relative importance of the pixels for the formation of the classification decision. The idea is, if the image contains the word "establishment," for example, in the centre of the image, and the network predicted that the image contains "establishment," the pixels in the centre (those that actually depict the word) should be those that the network

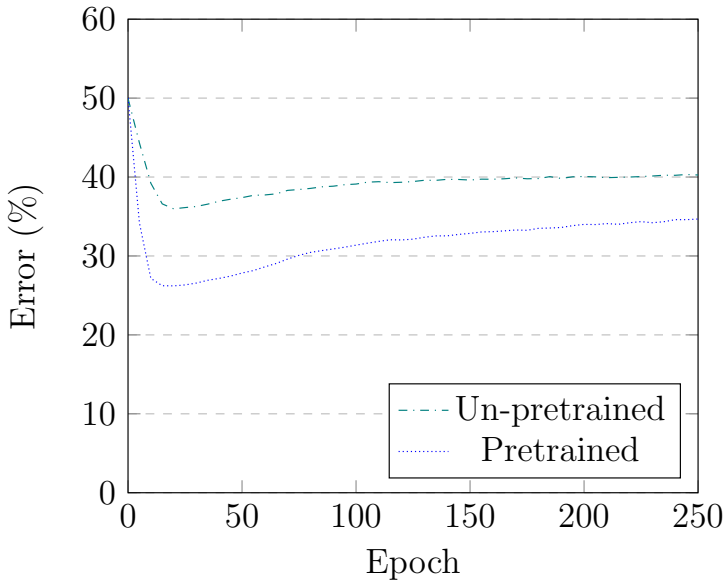

Figure 5: CNN on keyword search in Marsden documents

most heavily relied upon for its decision.

Fig. 6 shows the relative importance of each pixel in the correct classification made by the network that the page contains the word "establishment" (this word occurs at the start of the penultimate line in each image). If the networks really were recognising that word, we would expect a cluster of red pixels (i.e., these are the important ones) where that word is located, and mostly blue or green pixels everywhere else in the image. This expected heatmap is also shown in Fig. 6. Instead, we see a near-uniform distribution of yellow and green. So, whatever it is using to determine that this page contains "establishment," it is not the word itself.

Because these basic CNNs were promising (they perform significantly better than chance), but far from fully successful (as Fig. 6 demonstrates), it is worth exploring different CNN architectures. Importantly, for our problem of finding words in images, we know some of the relevant features that make up the word: individual letters. A natural thought is to try to exploit this fact, and try to train a network to recognise characters. That is what I did for this project.

Another reason for the poor generalised performance may be the importance of rare terms for keyword search. The network often had only 1–2 instances of each of these rare words to train from, making it very difficult to generalise. This problem gets even worse when we consider the unknown number words that

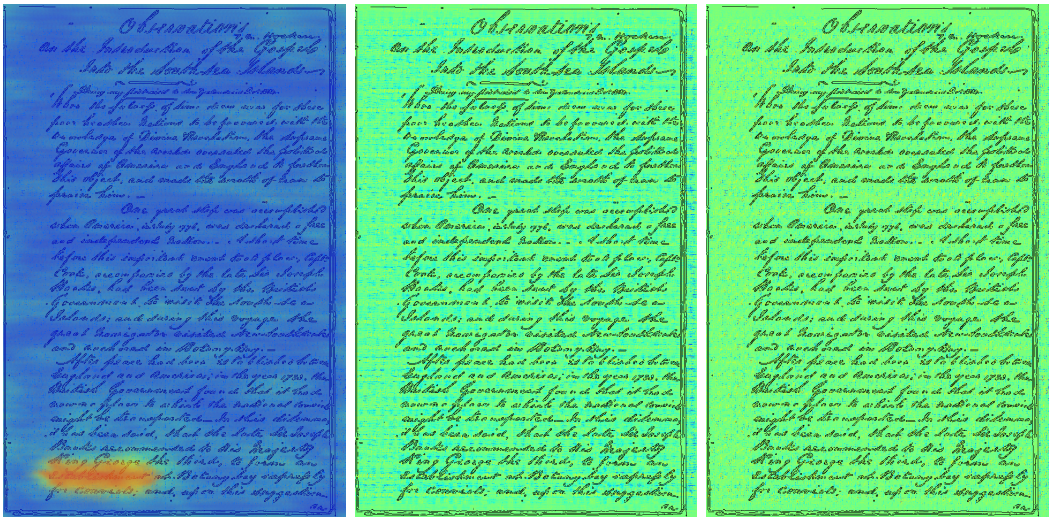

Figure 6: Expected heatmap for the word “establishment” (L), and heatmaps produced by un-pretrained (C) and pretrained (R) CNNs, which show that neither network responds to the actual word “establishment”

don’t occur in the training set at all. This is a problem that character recognition does not suffer from. Even the least common words are made up, statistically, of the most common characters. Also, although there may only be one example (or, indeed, no examples in the transcribed pages) to train from of a particularly uncommon word, there are many instances of the least common characters.

To make use of these features, however, the network will also need to predict the location of the letters. To accurately predict that a page contains the word “convicts”, for example, it isn’t enough to just find on the page any *c*, any *o*, any *n* and so on. It needs to find these letters *next to each other* and *in the right order*. Fig. 7 provides an illustration of the ultimate goal I have in mind. However, unlike OCR techniques, it may not need to have a very high level of accuracy on every character to do well at this task, as long as it is high

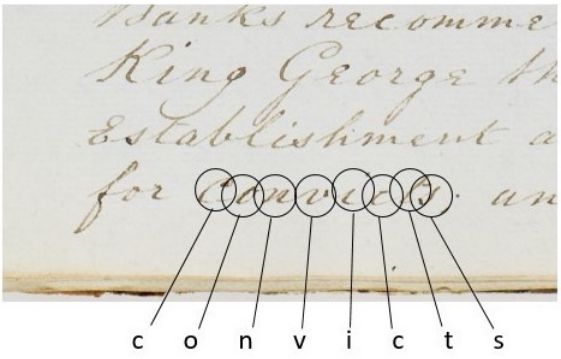

Figure 7: The goal: a network that can identify and locate individual characters

enough that a network can use statistical information about English to infer that it is likely that one of the patterns in the image is the keyword of interest.

2.3 Stacked Hourglass Networks

Stacked Hourglass networks are a recently developed kind of CNN with a particular architecture – they pool down to a very low resolution, then upsample and combine features using a symmetric topology, and then do it all again a number of times. They make heavy use of residual modules (as introduced in [20], see Fig. 8), or 'skip' layers, which preserve the information from early layers and provide it to the later layers directly, skipping the intermediary layers. This allows the output of the whole hourglass module to be informed by detail at several resolutions at once, which means that features of different scales can be detected, and the bigger picture information about the location of the feature/s is maintained even while the network continues to zoom in on the detail. This localisation information is especially important in this project, because we need precise and accurate predictions for the locations of the letters. Residual modules allow this precision and accuracy because the earliest layers keep track of this bigger picture localising information and feed it directly to the final layers, which use it to precisely locate the features it has found. The skip connections also allow for more efficient training of the early layers, as the error from the later layers can be backpropagated more directly to the earlier ones; thus minimising the vanishing gradient problem. One hourglass-shaped block of residual modules is shown in Fig. 9. Several blocks are then stacked, one after the other, making it a "stacked" hourglass network.

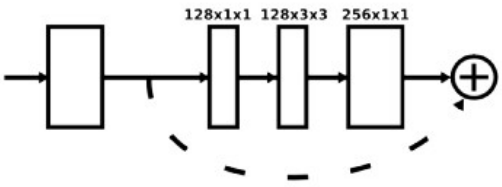

Figure 8: Residual module

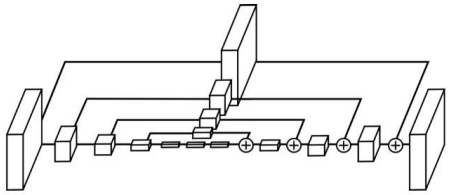

Figure 9: Hourglass shaped stack

This type of network has been demonstrated to be very effective at learning to locate semantic features of objects: for example, parts of the human body for pose estimation [21] and wheat spikelets for plant phenotyping [22].

The output of this kind of network is a heatmap which shows the location of each feature of the object, as predicted by the network. For wheat phenotyping,

the network can identify the location of each spikelet on each wheat plant in an image (see Fig. 10).

Figure 10: Wheat phenotyping [22]

For human pose estimation, the network can identify several kinds of feature on each human in an image, such as: pelvis, left elbow, right ankle (Fig. 11).

Figure 11: Pose estimation [21]

The supposition is that stacked hourglass networks are very good at identifying and locating semantic features of objects that have structure because of the way that they pool down and then upsample several times, using residual modules that allow detail from every resolution to inform the final prediction, giving them information about details at a number of levels, as well as precise and accurate information about location. This allows them to identify specific features that may require information about very fine details in conjunction with details at a more course-grained resolution, and to localise them in the image.

Many objects are made up of several semantic parts, each of which can be thought of as a feature. Humans have joints that the network can recognise and locate (and thus can determine, from the relative locations of the joints, the pose of the person). Wheat spikes are made up of several spikelets, each of which the network can precisely locate.

Handwritten text has similarly complex and important features: each word is made up of characters from an alphabet, arranged in a particular structure. In Marsden's case, as he wrote in English, the characters are those of the Latin

alphabet. Because of this small set of possible features, which are recombined in a large number of different configurations, the hypothesis is that stacked hourglass networks will be of use in identifying characters, and eventually words, in images.

The networks in both [21] and [22] have the same architecture, which is shown in Fig. 12, and is based on an encoding/decoding structure. First, convolutional and max-pooling layers form residual blocks which process features down to very low resolution, downsampling to a small, fixed-size (as small as 4×4 pixels) feature representation of the image. Each of these residual blocks has 256 filters in the final layer, so it can recognise 256 features. In theory, these are low-level features such as vertical or horizontal lines or patches of colour in the earlier layers, and in the later layers are complex features like eyes (for people) and blades of grass (for wheat stalks) – all of which are comprised of the lower-level features of the previous layers. It also branches off at each max-pooling layer, to preserve the spatial information at each resolution, and continues adding more convolutions to the pre-pooled layer. The information in these branches is then combined (by simple addition) as the network makes its way back up to the final image-like output.

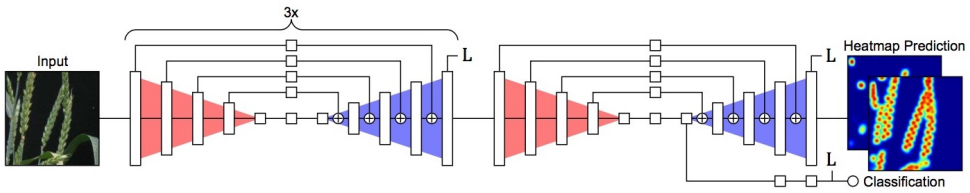

Figure 12: An overview of the CNN architecture in [22] (only two hourglass modules are shown here, but they used 4 and I used 8)

Once it has reached its lowest resolution, the network begins a top-down sequence of simple nearest neighbour unsampling and combination of features across scales. This process is symmetric: for every layer on the way down there is a layer on the way back up. Together, the encoding and decoding parts form an hourglass-like shape. This hourglass module is then duplicated several times (for [22], 4, and for [21], 4 or 8), and stacked one after the other so that the output of one hourglass module becomes the input for the next hourglass module.

This project, then, is exploring stacked hourglass networks and their application to recognition of handwritten characters (and words) from the Marsden collection. The goal of this project is to have implemented a stacked hourglass network that is able to recognise and locate individual characters in pages of handwritten text, with the idea that this information will be able to be applied

directly, or leveraged with the help of some other system, to perform keyword search to a high level of accuracy.

In order to explore stacked hourglass networks and how they perform on the task of interest here, I first tested the network on the Modified National Institute of Standards and Technology (MNIST) dataset, which is a standard dataset containing images of individual handwritten digits. This dataset is considered basic for CNNs now that they can perform much more complex classification tasks, but it is a good basic testing set for ensuring that my network is working minimally correctly, especially as the data is handwritten characters, similar to my actual data of interest. Once this was demonstrated to be working, I then tested on data of an apparently similar level of difficulty – individual characters taken from Marsden's documents, arranged in a simple 2×2 grid. I then scaled up the difficulty by training the network on more characters. After I demonstrated that the network was able to learn this basic task, I scaled up the difficulty further by introducing the possibility of multiple instances of each character in a single word, and then multiple words per image, and then arranged in less predictable locations. This gradual scaling up of difficulty allowed me to tweak the network as I went and build up what it could do incrementally.

3 Implementation

The architecture of my network is based on that proposed by Newell et. al. [21], and recently also implemented by Pound et. al. [22]. The main difference between these two networks is that in the first case, the network's goal is to detect and locate multiple instances of just two kinds of feature, while in the second case the goal is to detect and locate one instance of each of a set of features (each person usually has exactly one of each body part). In our case, we would ideally like to be able to detect and locate multiple instances of multiple features. Newell's implementation does, however, allow for multiple *people* in each image. So, I chose to use the implementation details from Newell. This allows many different features (in my case, characters from the Latin alphabet), as well as the option for multiple distinct clusters of those features (in my case, words).

I used Benbihi's [23] TensorFlow implementation of Newell et. al.'s stacked hourglass network. I made several modifications and updates to the basic implementation to get it to work with my data in the way that I wanted it to. In particular, I changed the sizes of the input and output layers, the number of features output by each residual block (I made this configurable), the way it selected training and validation data (no longer split randomly, for reasons given

in §3.1), and how it trained and measured accuracy. I also generalised the way the labels were turned into target output so that there could be any number of each kind of feature, rather than exactly one of each kind.

3.1 Data

Selecting and obtaining data for training is one of the main challenges when working with CNNs. It is important that there is a lot of it, first of all, but also that it is representative of the wider problem. It needs to be varied and as unbiased as possible, otherwise the CNN will just learn the specific details it needs to perform well on the task you gave it – biases and all. The data also needs to be accurately labelled, otherwise the network will learn the incorrect labels, and find spurious or irrelevant patterns that allow it to fit a model based on the incorrect class boundaries you have given it. If this is not possible, it may just fail to learn anything much at all.

In the case of this project, I need data to train the network to be able to classify and locate individual characters in images, but the transcribed Marsden documents are merely a text file with all the words (and thus, characters) in the image in order, but with no information about the location of the characters on the page (other than some that may be able to be inferred from the order they occur). Even then, we only have 1096 transcribed pages, and CNNs typically need much more data than that to train. So, I decided to generate my own data, using characters taken from the Marsden collection pasted together in random configurations. Generating data has the advantage that the labels can be accurately and precisely generated alongside the data, rather than requiring time-consuming and highly fallible hand-labelling of existing images. It also enables many thousands of random configurations to be generated, the idea being that we can produce enough data to be able to train the network to recognise and locate the characters in a way that is generalisable to unseen data (that is, without overtraining).

I wrote a tool that generates images from randomly selected smaller images. I first used images from the MNIST database (see Fig. 13), which is a publicly available database of handwritten digits originally sourced from

Figure 13: Data generated from MNIST

American high school students and Census Bureau employees. I chose this dataset to begin with because it is a standard dataset (now considered basic) for image classification, and because of the similarity with my real data, which is handwritten text. To use my tool, the user specifies the number of subimages to be incorporated (9 in each image in Fig. 13), then the tool randomly selects a digit type for each space, and randomly selects one of the instances of this number from the MNIST dataset. It simultaneously generates a file of labels for each image, formatted in the way the network expects (an extract from such a file is shown in Fig. 18).

Figure 14: Individual Marsden characters (*e, t, a, n*)

After I had shown that the network could learn this basic MNIST data (see results in §4.1), I then generated data from the Marsden documents. First, I hand cropped letters out of pages in the Marsden documents (see examples in Fig. 14). Each letter patch was 50×50 pixels, a fixed size to ensure that they're all at a uniform scale, chosen to balance keeping all of the detail of most of the letters, but not including a lot of detail from the neighbouring letters. I then shrunk each letter patch to 28×28 pixels to reduce the number of input channels (and thus complexity) the network would require. This resolution maintains enough detail for the letters to be distinguishable.

I cropped out 85 individual variations for each of the 26 letters (*a*–*z*). I tried to get a variety of styles when they existed – for example, when Marsden wrote a double *s* (as in, *ss*), the shape of the first *s* is quite starkly different to most of his other *s*'s (for example, see Fig. 15). I selected only from lowercase letters, except in the case of *z*, because it occurs so rarely that I couldn't find enough instances. So, I brought up the numbers using the uppercase *Z* in several instances of 'New Zealand'. I don't mind if the network isn't very good at recognising *z*'s, because they occur so infrequently, but as can be seen in the results (§4.3), it didn't

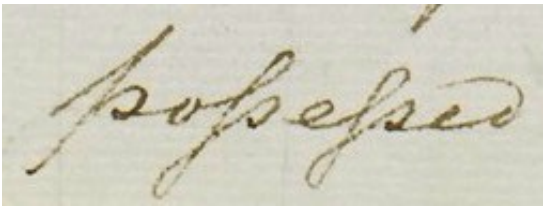

Figure 15: Marsden's distinctive double-*s*, shown twice in the word "possessed"

adversely affect the ability to learn to recognise z's.

I took samples from a variety of locations on the pages (top, bottom, middle, left, right), in case this introduced any systematic variety that wasn't immediately obvious to me. In particular, some letters very close to the edges have been warped by the scanning process, so made sure to include at least some of those in each case. I chose some from the start, the middle, and the end of words, and a selection of any other variations I noticed (for example, the vast majority of f's occur in 'of' or 'from,' so while I got many of the variations from those words, I gravitated toward f's in other words where possible, to increase variety).

Using this new dataset of individual characters, I generated random combinations to form my actual datasets. I used 70 of the 85 variations (of each letter) to generate the training set, 10 of the remaining variations to generate the validation set, and the last 5 variations to generate a small test set. The alternative would have been to generate one set of data from all 85 variations, and then randomly divide these into training, validation, and testing sets. However, this would mean that many (likely all) of the individual variations would occur somewhere in all three sets, so although the larger image with random collection of variations might be unique and occur in one set but not the others, it wouldn't provide a good check against overfitting, because the individual letters that make up each image in the test set would have been seen before. Therefore, I kept the three sets entirely separate by using distinct groups of letter variations for each.

Using the letter frequency order *e-t-a-o-i-n-s-r-h-l-d-c-u-m-f-p-g-w-y-b-v-k-x-j-q-z* (obtained from [24]), I generated first some 2×2 images using the first four most frequent letters (e, t, a, o), then I generated some 3×3 images using the first nine (up to h), 4×4 using the first 16 (up to p), and 5×5 using the first 25 (all letters but z). Some examples of this basic data are shown in Fig. 16.

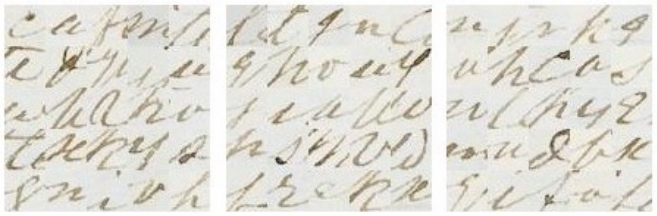

Figure 16: Examples of generated data

Once I had shown (see results in §4.2) that the network could learn this simple data, I generated another, significantly more difficult, dataset. First, I generated images of 10×10 characters at random, each letter occurring at the same frequency as it is found in English (see Table 1 for these frequencies, taken from [24]). This means that there are many more occurrences of e and t than of q and z, which I chose to do because a bias toward accurate prediction

the characters it is likely to see more frequently 'in the wild' is desirable.

Table 1: Relative letter frequences in English

e	12.49	*r*	6.28	*f*	2.40	*k*	0.54
t	9.28	*h*	5.05	*p*	2.14	*x*	0.23
a	8.04	*l*	4.07	*g*	1.87	*j*	0.16
o	7.64	*d*	3.82	*w*	1.68	*q*	0.12
i	7.57	*c*	3.34	*y*	1.66	*z*	0.09
n	7.23	*u*	2.73	*b*	1.48		
s	6.51	*m*	2.51	*v*	1.05		

This resulted in images of size 280×280, which had characters arranged in a regular grid. To combat this regularity, each image was then cropped to 256×256 at random. Any character which had 3/4 or more remaining is considered present in the image. The characters are still positioned in a grid pattern, but the grid is not so predictable. An image from this final training set is shown in Fig. 17.

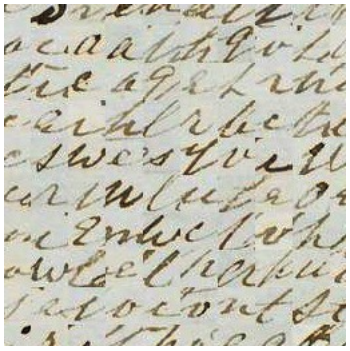

Figure 17: Training image

```
00010.jpg e 62 58 146 58 34 86 90 114 202 142 62 170 90 198 34 226
00010.jpg t 230 30 174 58 230 86 174 142 202 226
00010.jpg a 62 30 90 30 90 58
00010.jpg o 202 30 230 142 202 170 90 226 146 226
00010.jpg i 118 30 34 58 62 86 202 114 62 142 34 170 62 226
00010.jpg n 230 58 174 226
00010.jpg s 34 114 118 114 202 198 230 226
00010.jpg r 202 58 146 86 34 142
00010.jpg h 146 30 90 86 230 170
00010.jpg l 118 86 118 142 174 170 62 198 118 198
00010.jpg d 174 198
00010.jpg c 34 30 202 86 146 170 118 226
00010.jpg u 146 142 230 198
00010.jpg m 90 170
00010.jpg f -1 -1
00010.jpg p 146 198
00010.jpg g 174 30
00010.jpg w 62 114 90 142 118 170 34 198
00010.jpg y 146 114
00010.jpg b 174 86
00010.jpg v 174 114
00010.jpg k -1 -1
00010.jpg x -1 -1
00010.jpg j 230 114
00010.jpg q 118 58
00010.jpg z -1 -1
```

Figure 18: Labels for the image in Fig. 17

The labels for each image are given in a text file, in the form of pixel coordinates of the centre of each character, by type (see Fig. 18). Values of -1 indicate that there are none of this type in the image.

During the preprocessing of the data, these labels are used to generate a set of target 64×64 heatmaps for each image, to be used as the ground truth in supervised learning. Each image gets associated with it 26 target heatmaps (one for each character type). For each heatmap, a Gaussian with a standard deviation of one pixel is generated at each of the relative locations of the individual characters of that type. Preprocessing also flattens the colours into a greyscale image. Example target heatmaps for the image in Fig. 17, taken from the labels given in Fig. 18 (those for *e*, *t*, and *o*) are shown in Fig. 19.

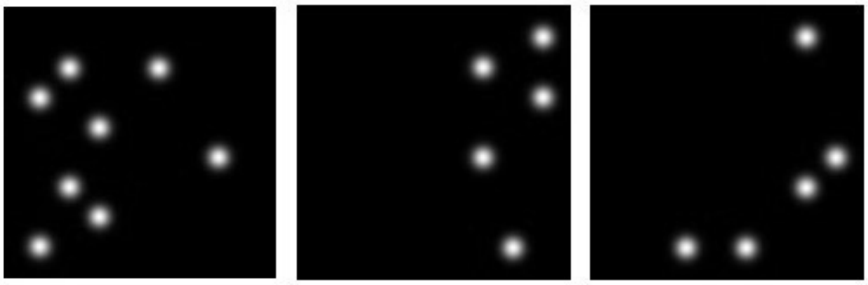

Figure 19: Target heatmaps for the image in Fig. 17 (e, t, and o shown)

3.2 Network

My network is based on that outlined in Newell et. al. ([21]). Each hourglass-shaped stack consists of 15 residual modules, each with 3 layers of 128, 128, and 256 filters (shown in blue in Fig. 20). The number of output features for each residual module of the network is configurable, so I also tried it with 128 and with 64. (see Fig. 8 and Fig. 9).

This hourglass-shaped network is then duplicated and stacked so that the output of the first becomes the input for the second, and so on. Following Newell et. al., I used 8 stacks. The full architecture of the network is shown in Fig. 20).

The network takes, as input, images (my data described in §3.1) of a configurable size. If they are any bigger than 256×256, it shrinks them down to this size. Otherwise, it keeps them the size they arrived. The final output is a set of prediction heatmaps: one for each character type, to match the target. So, for example, in the case with 4 characters, it outputs 4 heatmaps, one for each of e, t, a, and o. The heatmap gives the likelihood that the specified feature (character) is located at each pixel, as predicted by the network.

3.3 Training

I generated training and validation sets (of size 9000 and 1000 respectively) independently from distinct pools of letter variations, rather than generated from one pool and then split, to ensure the variations themselves are kept separate and only the training variations may be trained on, with the validation set being used to monitor progress. The testing set was generated from a third distinct set of letter variations.

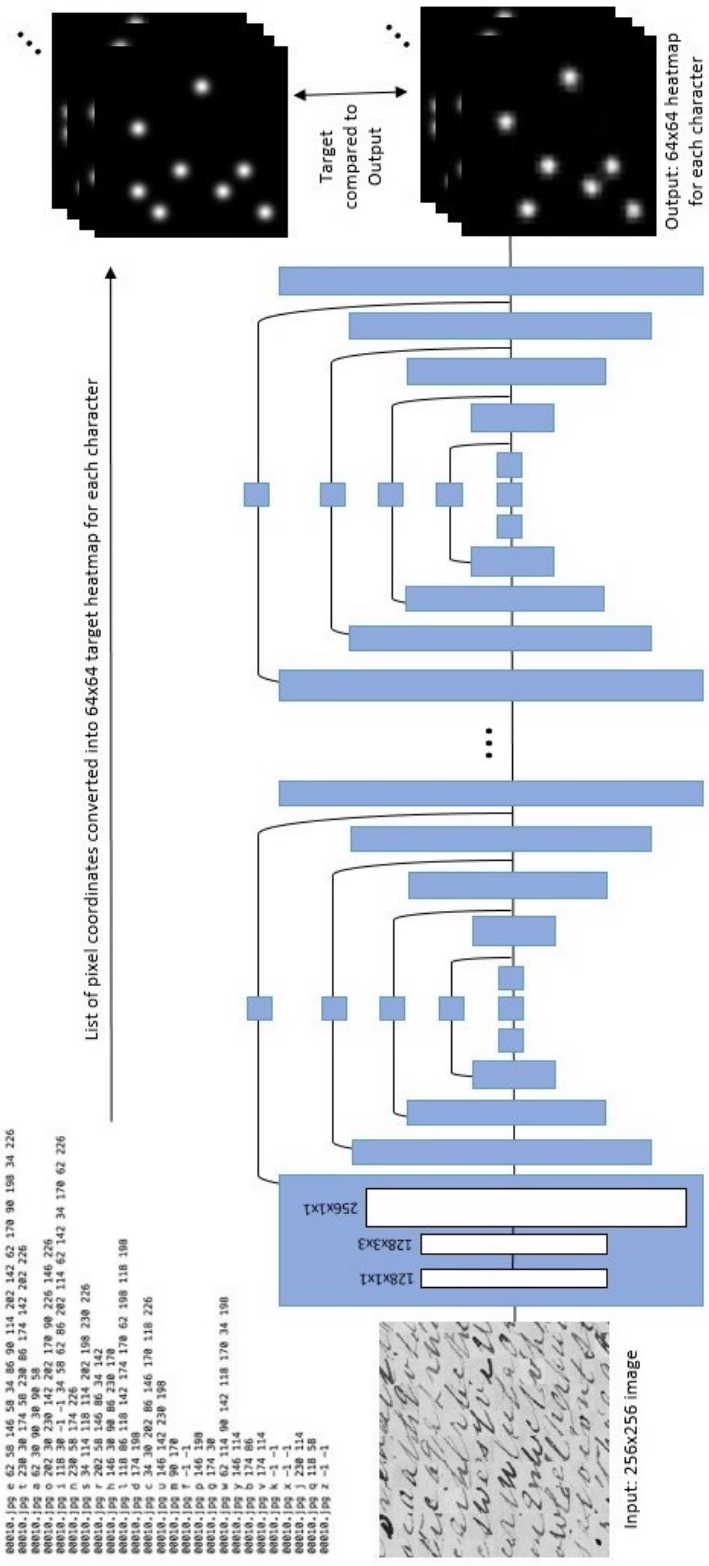

Figure 20: Network Architecture

I used supervised learning, which is standard for CNNs learning classification tasks. In supervised learning, the network is given an input, produces an output, and that output is compared to a ground-truth "correct" output, that the network is being trained to match. It calculates the difference between the output it gave and the target output, then updates its weights to make it likely to produce output more similar to the target next time it sees the same input. This 'difference' is measured in some way that fits the task: in this case, it was the amount of difference between each pixel in the output and target heatmaps.

I trained the network for 2000 epochs (passes through all of the data). The data was presented in mini-batches size of 4 (in order to fit into GPU memory), with training updates made at the end of each mini-batch. It was trained to minimise loss, which is calculated as mean squared error over every pixel in each heatmap. The network reached close to peak performance by 500 epochs, after which it didn't continue to improve, nor did it show any signs of overtraining.

3.4 Evaluation

For the initial stages, where there is exactly one of each kind of feature in each image, I measure accuracy by taking the maximum activation for each character's heatmap, and considering that to be the predicted location for that letter. This is then compared to the ground truth (target) location of the letter and Euclidean distance, as a proportion of the size of the heatmaps, is taken to be the error. Accuracy can be computed as $1 - error$, to obtain what is essentially how *close* the predicted location of the feature is to the actual location of the feature.

Later, when I move to the final version of the data, which can contain any number of each kind of feature (or none at all), this measure becomes inappropriate. The data can have any number of a particular feature, not always exactly one, and so it is no longer meaningful to compute *the* predicted (or target) location of a feature. There may be none, or many. So, we need a measure of accuracy that reflects this variation. For this final stage, then, I move to a measure of classification accuracy for each actual character in the image. I report top-1, top-3, and top-5 classification accuracy. This is computed as follows: the maximum activation over the local 5×5 pixel region of the output heatmap for each letter is considered 'the activation' for that location. A prediction is considered to be the maximum activation over all the 26 heatmaps. Because many of the character variations genuinely look ambiguous – for example, some e's look very much like c's or even q's (see Fig. 14), I consider top-3 and top-5 accuracy to also be appropriate measures.

It is also important to note what level, in each of these kinds of accuracy, counts as

good accuracy. This depends on what random activation patterns would score on the particular accuracy. In the first measure, Euclidean distance of two random pixels in an image is about 37% of the image size (I ran a simulation with n=1,000,000 pairs of random points uniformly distributed across the image). So, random activation should give us around 63% accuracy. In the second case, if the network outputs random activation patterns, then which feature causes the maximum activation is also random, so the chance of picking the correct letter is 1/26, which is 3.85%. For top-3 accuracy, random activation will give us 11.54%, and for top-5, 19.23%. If the network can learn the minimal information that e is the most common letter and thus predict e's everywhere, it could get top-1 accuracy of 12.49%, 37.47% for top-3 accuracy, and 62.45% for top-5 accuracy.

4 Results

4.1 MNIST

I first tested the network on MNIST, both with 4 digits and with 9 digits. As shown in Fig. 21, both learned to a high accuracy very quickly.

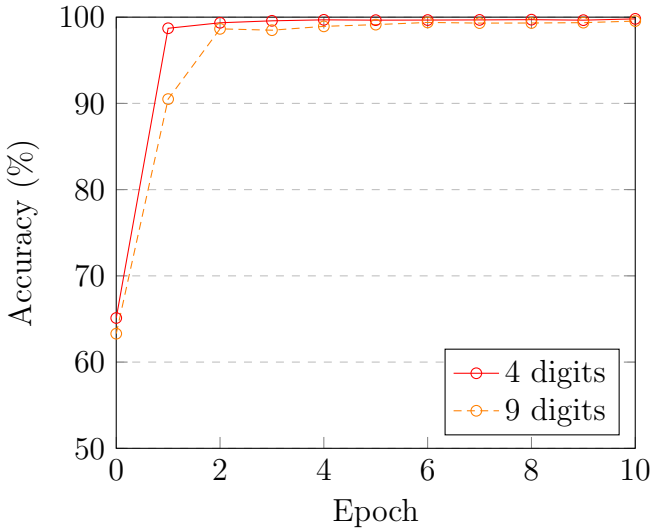

Figure 21: Training a Stacked Hourglass Network on MNIST digits

For simplicity, I've graphed the performance on validation data only – it is always very similar to the training data, and this is just to demonstrate that they quickly learn to a high level. As expected, it takes a bit longer to learn 9 than to learn just

4, but both are performing at over 99% accuracy within 4 epochs. Note that epoch 0 is before any training whatsoever, and the network performs at about 63% accuracy, which is what we expect a random activation pattern to give us.

4.2 Simple data

After demonstrating that the network was working as expected: learning to recognise and locate handwritten digits from the MNIST dataset, I generated simple data from the Marsden characters, and tested the network on this. I tried it with 4, 9, 16, and 25 characters. The results for the first 50 epochs are shown in Fig. 22.

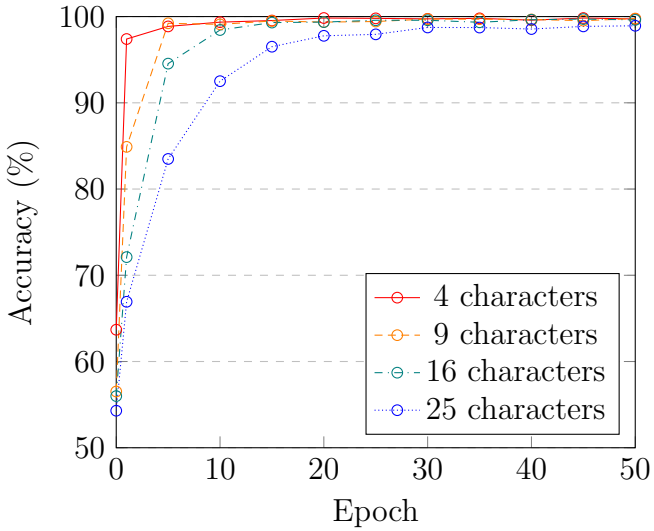

Figure 22: Training a S-H Network on Marsden characters

Again, I have shown only the performance on the validation data as it is indicative of performance on the training data. In the case of 4, 9, and 16 characters, within 10–15 epochs the network performs consistently at over 99% accuracy on validation data. As expected, it learns 4 characters the most quickly, then 9, then 16, and 25 the least quickly. It takes about 75 epochs to learn the 25 character data to over 99% validation accuracy. In all cases, once it has reached that high level of accuracy, it maintains it for both training and validation data for as long as I have tested it: 500 epochs at least. In particular, this is notable because the validation accuracy doesn't start to decrease. When training CNNs (or any other statistical modelling based machine learning technique), we expect the training accuracy to continue to increase (or to flatten off at a very high level), but the validation accuracy to increase for some

time, and then start to decrease. This is because if the network trains for too long, it starts to "overfit" to the training data. Overfitting is when the network starts to pay too much attention to the small variations in the training data, memorising every variation, which causes it to lose the smooth shape of the nice, general, function. This means its performance will decline on validation (and test) data because it is merely memorising all the details of the training data and so failing to generalise on the data that it doesn't train on.

For example, the performance of a network where each module outputs 64 features on 25 Marsden characters is given in Fig. 23, showing both training and validation accuracy for 300 epochs (it continues like this for at least 500 epochs). The accuracy on both training and validation data reach at least 98% and then remain constantly at this level. While Fig. 23 shows the performance of a network with the capacity to learn 64 features at each module, it is similar for 128 and 256 features.

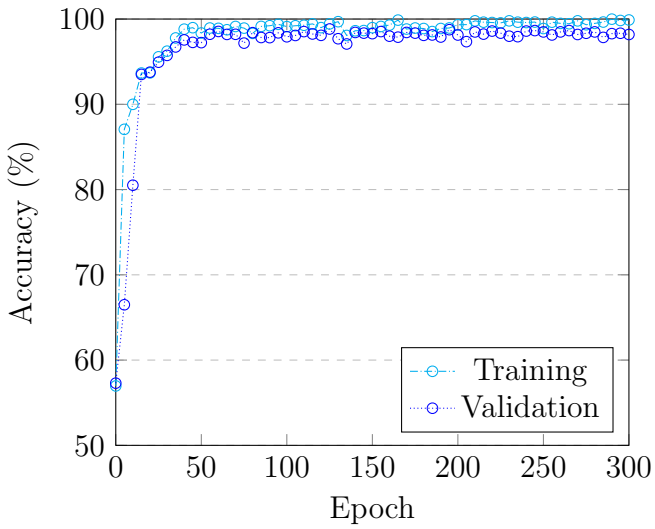

Figure 23: Training a S-H Network on 25 Marsden characters

It is puzzling that the network doesn't begin to overfit to the irrelevant variations (noise) in the training data. Perhaps this indicates that the network doesn't have the capacity to overtrain – to overtrain, the network must be powerful enough to be able to memorise all of the training data. To test this hypothesis, I went back to the 4 character dataset, and increased the number of features in the network to 280. Given that for each of the four characters, there are 70 variations in the training set, 280 features is enough that the network should be able to memorise each of the 280 character variations as a distinct feature, and thus we should observe overtraining if the network behaves as other CNNs behave. The results are shown in Fig. 24.

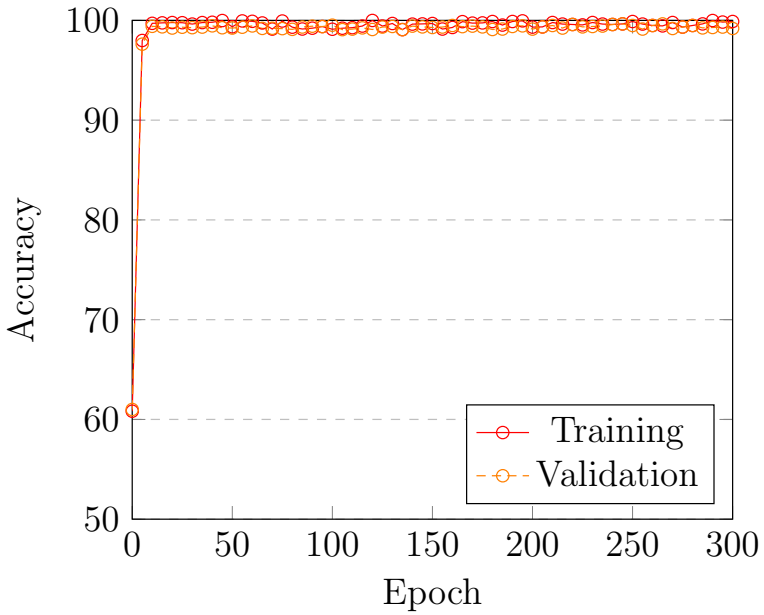

Figure 24: 4 Marsden characters, 280 features

These results indicate that that even when the network has enough features to memorise one variation of one character with each feature, it still does not overtrain. It could be that the distinctive architecture of the stacked hourglass network prevents overtraining because it forces the features to be compressed down into something useful even at low resolutions, thus forcing more general, or average, features for each type of character. This is an interesting area for further research.

These preliminary results appear to show that the network is competently learning the Marsden characters and is able to locate them in the images. So, to verify these results I wrote a script that loaded the trained network and fed images through it, outputting a series of 25 pixel locations – the hottest spot from the output heatmaps for each of the 25 features. Some results on testing images are shown in Fig. 25.

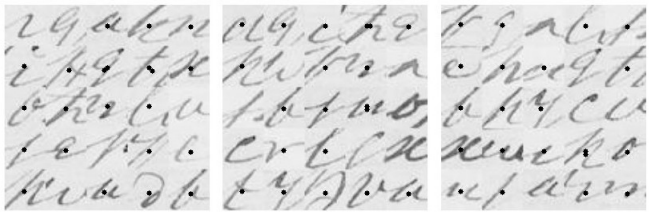

Figure 25: Hotspots

4.3 Final data

After demonstrating that the network can learn the locations of the individual letters in this restricted setting with simple data, I modified the pre-processing of the data to be able to cope with any number of each type of feature (0 to arbitrarily many). This was to accommodate the form of my final set of training data, which was images generated out of character variations where each type of letter occurred at the frequency which matches its occurrence in English, and cropped at random to ensure it couldn't rely on the letters occurring in one of the only 25 possible positions. This kind of data is shown in Fig. 17. After training, the performance (on the same training image given in Fig. 17) is demonstrated qualitatively in Fig. 26 (quantitative results are given in Table 2). The images are each the greyscale version of the original image superimposed with the heatmap for a given character. For comparison with the target heatmaps given in Fig. 19, *e*, *t*, and *o* are shown. In addition, the heatmaps for *c* and *l* are shown.

Figure 26: Performance on the image in Fig. 17 (top: *e*, *t*; bottom: *o*, *c*, *l*)

By comparison with the target heatmaps in Fig. 19, we can see that the network has found six of the eight *e*'s, and the other two it has misclassified as *c*'s, which seems reasonable when we look at the specific *e*'s in question. It has only classified two of the five *t*'s correctly, but the other three have all been misclassified as *l*'s,

which is reasonable at least in two cases, as they are missing their crosses entirely, thus being essentially indistinguishable from *l*'s, especially out of context as these are being presented. It has also picked up all but one of the *o*'s, and for the last one it has some level of activation but it does not appear to be confident (and as we can see, that one indeed looks very un-*o*-like).

In addition to measuring the performance of the network on training and testing data, I evaluated how well the trained network generalises to real data. 20 pages were selected at random from the original Marsden documents, and a 256×256 patch was cropped at random from each page. I then hand-labelled the locations of each letter in that patch to give the ground-truth locations of the letters. The network was then tested on these real images. Accuracy for training, testing and real data is presented in Table 2.

Table 2: Accuracy on Training, Test and Real Data

	Training Data	**Test Data**	**Real Data**
Top-1	88.75%	63.69%	59.00%
Top-3	93.30%	75.15%	72.60%
Top-5	95.40%	81.02%	79.80%

This performance, even on randomly selected patches of real data, is significantly better than both the random activation (3.85% for top-1 accuracy) and the 'guess everything is an *e*' minimal learner (12.49% for top-1 accuracy). The example heatmaps in Figs. 26 and 27, and the details of accuracy broken down by character along with the most common errors as shown in Table 3, also show that this does not appear to be what it is doing.

Some examples of the network's predictions for characters on patches of real data are shown in Fig. 27. Again, the heatmaps were superimposed on the greyscale versions of the original inputs. Note that it correctly predicts even in some surprising cases: for *s*, it correctly finds a large variety of different shapes and sizes, including the same instance we saw in Fig. 15 of "possessed," and even on a strange angle in the second image. It also correctly finds the *z* (the word on the second line of the second image is "zealanders") even though this only occurred in the training data at a frequency of 0.1%. The images also suggest that the kinds of false positives it makes are sensible ones: when predicting *n*'s, for example, it also finds *n*-like *u*'s and *m*'s (both shown in the second image). When predicting *l*'s, it also finds the tops of *h*'s, the tail of *y*'s, and other such *l*-like shapes.

To revisit our original goal example, consider the (actual) top-3 hypotheses for the characters in "convicts," shown in Fig. 28. Of the eight letters, the network

Figure 27: Performance on patches of real data (*n, s, d, z, and l* shown)

correctly predicted five of them as its top hypothesis, and another one (v) as its second hypothesis. This example is representative of the overall results on unseen data: top-1 and top-3 performance on this example are very close to that reported in Table 2: this example shows *typical* performance of the network. The mistakes it makes are, I think, reasonable ones, particularly considering that the network has just learned the letters in isolation. For English speakers, it is much easier to discern letters in context because we can make use of our implicit (or explicit) knowledge about which letters are likely to be found adjacent to one another, up to and including, of course, our mental dictionary of English words.

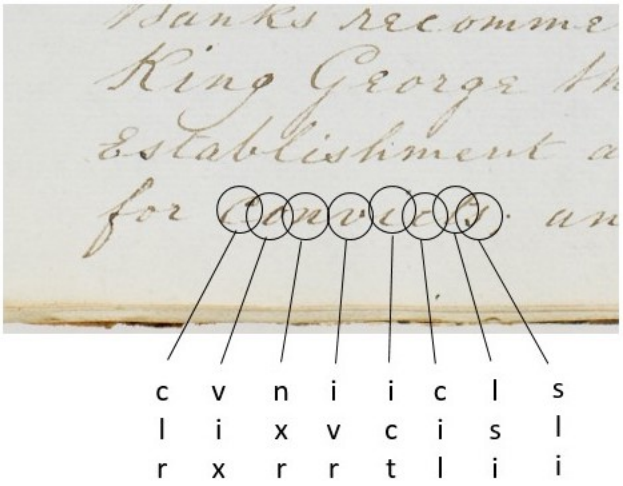

Figure 28: Top-3 hypotheses for each letter of "convicts"

The network seems to be making sensible classification mistakes when it misclassifies characters. However, merely *seeming* is not a reliable result. So, I measured the most common classification errors for each type of character. That is, when it does not identify the correct letter, I identified which letter it is most likely to classify it as. The character most commonly identified incorrectly for each character type is given in Table 3. I have presented the results on testing data because it gives an indication of performance on unseen instances of the characters, so an idea of how the network's function generalises. However, the sample size of the testing data is only 5 distinct instances of each type of character, so the results could easily be skewed by the existence of one unusual instance. Misclassifying just one outlier could misleadingly inflate the proportions for that character type. So, I have also presented the results of training data, which provides more variation for each letter (70 distinct instances of each type of character).

As shown in the table, these misclassification errors are generally not uniformly

Table 3: Most Common Errors (MCE) for each Character

	Training Data		**Testing Data**	
Character	Top-1 Accuracy	MCE	Top-1 Accuracy	MCE
e	72.81%	c (20.05%)	43.34%	c (14.52%)
t	50.02%	l (31.62%)	24.70%	l (24.40%)
a	98.00%	r (1.27%)	48.14%	u (24.16%)
o	91.21%	a (4.89%)	46.89%	v (35.12%)
i	96.45%	c (1.02%)	98.40%	j (0.89%)
n	99.51%	p (0.19%)	98.37%	m (1.63%)
s	97.67%	j (1.43%)	59.15%	l (29.27%)
r	96.77%	i (1.13%)	88.64%	x (3.18%)
h	89.53%	p (8.08%)	35.39%	p (54.69%)
l	98.52%	i (0.37%)	100.00%	NA
d	98.32%	w (0.53%)	75.36%	w (9.78%)
c	98.64%	l (1.22%)	76.68%	i (20.95%)
u	89.28%	n (10.38%)	48.67%	w (25.66%)
m	99.59%	n (0.38%)	87.71%	n (12.29%)
f	94.66%	j (3.37%)	52.17%	j (24.64%)
p	98.24%	m (1.07%)	81.94%	w (5.16%)
g	94.51%	q (2.10%)	54.86%	l (30.56%)
w	95.95%	m (1.80%)	84.43%	m (13.11%)
y	98.38%	j (0.65%)	71.20%	q (7.20%)
b	94.76%	k (2.10%)	99.15%	k (0.85%)
v	90.38%	u (6.13%)	73.97%	u (16.44%)
k	93.77%	p (4.09%)	79.63%	p (18.52%)
x	97.90%	n (1.05%)	92.31%	c (7.69%)
j	87.55%	l (6.97%)	83.33%	l (16.67%)
q	91.50%	g (5.17%)	33.33%	g (66.67%)
z	93.46%	j (3.62%)	75.00%	j (25.00%)

distributed across the incorrect letters, but it is in fact making regular, quite sensible, mistakes. That is, mistakes that I believe a human could conceivably make when considering the letters in isolation. For example, e is most often misclassified as c, t is most often misclassified as l, and (for testing data) n & m are most often misclassified as each other. Referring back to the example characters in Fig. 14, we can see that, indeed, many of the e's could plausibly be mistaken for c's, t's for l's, and so on. Similarly, w is most commonly misclassified as m, m as n, q as g, v as u, b as k, f as j, and j as l. In all of these cases, the characters do have similar shapes.

Nothing I have done demonstrates that this particular architecture is *necessary* for this task, or even that it has an edge over any other architecture on this task. I didn't investigate too deeply into this question, though I tried it on the same architecture but with 1 stack and with 4 stacks, as well as with 8 stacks but only 64 features in each module, and it had trouble learning to the same level of accuracy in each of these cases. Some preliminary results (on training and test data after 500 epochs) are shown in Table 4. Although this data shows poor results compared to the version with 8 stacks and 256 filters, it is possible that with some further tweaks (such as significantly longer training times, or different batch sizes, or more filters), the performance could perhaps be lifted.

Table 4: Performance on alternative architectures

	4 stacks, 256 filters		**1 stack, 256 filters**		**8 stacks, 64 filters**	
	Training	**Test**	**Training**	**Test**	**Training**	**Test**
Top-1	58.21%	37.40%	21.87%	11.28%	42.61%	29.89%
Top-3	66.64%	42.03%	32.20%	22.13%	52.19%	40.52%
Top-5	79.40%	59.63%	45.16%	30.49%	62.01%	49.10%

5 Discussion

While the performance of my stacked hourglass network is not yet at a level where it can be directly applied to documents as a search algorithm, it is certainly promising and deserves further investigation and development. The results presented here have been submitted to IVCNZ 2018. The next step would be to leverage the information it has learned and feed it into some other system to obtain a keyword search tool. For example, the output of this network could be fed into another CNN which would use it to learn how to identify and locate whole words based on the features it has learned already, using a form of transfer learning [18]. Alternatively, the output could be fed into a hidden Markov model that combines the characters' locations with known statistical properties of English to improve the accuracy of the predicted character [25] [26]. For example, as in the case shown in Fig. 28 if the network sees what looks like a v, but it is preceded by what it is pretty sure is a c, it can adjust its prediction for the apparent v. This could be supplemented by an understanding of what kinds of mistakes it commonly makes (t's do in fact look very similar to l's, so predicting l for t is a more common mistake than predicting x for t). Perhaps, instead, the output of this network could be fed into a recurrent neural network,

which can keep track of patterns in ordered input and essentially learn the statistical properties of English itself (see [27] for an example of this kind of approach). Or, perhaps it will be enough to simply feed the top-3 or top-5 predictions (with their relative locations) into a brute-force search of a look-up dictionary, to find the word that pattern of letters is most likely to be, from a set list of words: approximately 200,000 in English [28].

An interesting observation is that the network didn't appear to be subject to overtraining (even when allowed to continue training for several thousand epochs). Perhaps this indicates that the distinctive architecture of the stacked hourglass network prevents overtraining because it forces the features to be compressed down into something useful even at low resolutions, thus forcing more general, or average, features for each type of character. This is an interesting area for further research.

Acknowledgments

First of all, I would like to thank Steve Mills and Lech Szymanski for all their help and support. Without their suggestions, discussion, help with debugging, and thorough comments on drafts, this project would be in far worse shape.

I would also like to thank Hocken Collections, and in particular Anna Blackman, for providing the document images used in this collection. I gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X GPU used for this research.

References

[1] Lech Szymanski and Steven Mills. *CNN for Historic Handwritten Document Search*. IVCNZ, 2017.

[2] gk_ "How Neural Networks Work," https://chatbotslife.com/how-neural-networks-work-ff4c7ad371f7

[3] W. S. McCulloch and W. Pitts. *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Bulletin of Mathematical Biophysics, 5: 115–133, 1943.

[4] D. O. Hebb. *The Organisation of Behaviour: a Neuropsychological Theory*. John Wiley & Sons, NY, 1949.

[5] F. Rosenblatt. *Principles of Neurodynamics*. Spartan, NY, 1962.

[6] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*. In D. E. Rumelhart, J. L. McClelland, The PDP Research Group (eds.) "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations." MIT Press, Cambridge, MA, 1986.

[8] Yann LeCun, Yoshua Bengio, and Geoffery Hinton. *Deep Learning*. Nature, 521(7553): 436–444, 2015.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*. Proceedings of the 25th International Conference on Neural Information Processing Systems,Vol. 1: 1097–1105, 2012.

[10] Rensu Theart. *Getting started with PyTorch for Deep Learning (Part 3: Neural Network basics)*. https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics, 2017.

[11] P. M. Manwatkar and K. R. Singh, "A technical review on text recognition from images," International Conference on Intelligent Systems and Control (ISCO), 2015.

[12] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Reading text in the wild with convolutional neural networks," International Journal of Computer Vision, vol. 116, no. 1, pp. 1–20, 2016.

[13] A. Poznanski and L. Wolf, "CNN-n-gram for handwriting word recognition," in Proc. Computer Vision and Pettern Recognition, pp. 2305–2314, 2016.

[14] L. Schomaker et. al. http://monk.hpc.rug.nl/monk/demo.html.

[15] S. Sudholt, G. A. Fink, "PHOCNet: A deep convolutional neural network for word spotting in handwritten documents," International Conference on Frontiers in Handwriting Recognition, 2016.

[16] P. P. Roya, A. K. Bhuniab, A. Dasb, P. Dharc, and U. Pald, "Keyword spotting in doctor's handwriting on medical prescriptions," Expert Systems With Applications 76, pp. 113—128, 2017.

[17] C. Wigington, C. Tensmeyer, B. Davis, W. Barrett, B. Price, and S. Cohen, "Start, Follow, Read: End-to-End Full-Page Handwriting Recognition," European Conference on Computer Vision (ECCV), 2018.

[18] L. Y. Pratt, "Discriminability-based transfer between neural networks," NIPS Conference: Advances in Neural Information Processing Systems. Morgan Kaufmann Publishers, 204–211, 1993.

[19] S. Bach, A. Binder, G. Montavon, F. Klauschen, K. Müller, W. Samek, "On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation," PLoS ONE 10:7, 2015.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[21] Alejandro Newell, Kaiju Yang, and Jia Deng. *Stacked Hourglass Networks for Human Pose Estimation*. CoRR, 2016.

[22] Michael Pound, Jonathan Atkinson, Darren Wells, Tony Pridmore, and Andrew French. *Deep Learning for Multi-task Plant Phenotyping*. Cold Spring Harbor Laboratory, 2017.

[23] Walid Benbihi. *Tensorflow implementation of Stacked Hourglass Networks for Human Pose Estimation*. https://github.com/wbenbihi/hourglasstensorlfow, 2017.

[24] Peter Norvig. *English Letter Frequency Counts: Mayzner Revisited, or ETAOIN SRHLDCU*. http://norvig.com/mayzner.html, 2013.

[25] C. E. Shannon, "A Mathematical Theory of Communication," The Bell System Technical Journal, Vol. 27, pp. 379–423, 623–656, July, October, 1948.

[26] F. Wang, Q. Guo, J. Lei, and J. Zhang, "Convolutional recurrent neural networks with hidden Markov model bootstrap for scene text recognition," IET Computer Vision, vol 6, no. 11, pp. 497–504, 2017.

[27] S. Lai, L. Xu, K. Liu, J. Zhao, "Recurrent Convolutional Neural Networks for Text Classification," Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.

[28] Oxford English Dictionary, "How many words are there in the English language?" https://en.oxforddictionaries.com/explore/how-many-words-are-there-in-the-english-language/