Confusion by Occlusion: Inference, Graphics, and Representations of Depth

303034908

August 7, 2019

Department of Applied Mathematics and Theoretical Physics
University of Cambridge

A thesis presented for the degree of
Master of Philosophy in Computational Biology

Word count: 11966

Declaration of Authorship

I hereby declare that this dissertation entitled Confusion by Occlusion: Inference, Graphics, and Representations of Dept is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. I further state that no substantial part of this dissertation has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I confirm that I have read and understood the Faculty of Mathematics Guidelines on Plagiarism and the University-wide Statement on Plagiarism.

Acknowledgements

I am incredibly grateful to my supervisor, Ruben van Bergen, who with immense patience and creativity helped me in untangling this project. I am also thankful to Lucas, my friend, for his calm nature, his emotional support and companionship during the late nights. Finally, I am indebted to Niko Kriegeskorte for accepting me in his lab at the Zuckerman Institute, Columbia University, New York as well as his positivity at all times. It has been one of the best periods of my life.

Abstract

Visual perception refers to the ability to interpret the retinal activities to build an internal representation the local environment. At its core visual perception is an inference process: it tries to infer which representations best explain the visual input it receives. These representations can simply be a breakdown of the image into its objects and their properties. If this list is fully informative, we could invert the process and recreate the image. This is called graphics. When dealing with images of object that are occluded, and therefore ambiguous, inference becomes difficult. In this thesis we test the hypothesis that the brain contains an internal graphics engine that is used to refine and guide visual inference of occluded objects. We also hypothesise that this graphics engine is achieved in the form of feedback connections and recurrent dynamics in the visual stream. To test these hypotheses, we train, test and compare Artificial Neural Networks with different combinations of of feedforward and feedback connections on three tasks. First, to infer identity, location and depth from images of digits under occlusion. Second, to generate images of occluded digits from a pre-set representation. Finally, we train an autoencoder that combines the two tasks into one, from images to a compressed representation and then back, to investigate whether a better representation emerges and whether the autoencoder learns an insightful way to represent depth and compute with occlusions.

Contents

List of Figures

Chapter 1

Introduction

Visual perception as inverse graphics

Viewing the world seems so effortless that we forget that information from your local environment only reaches us by light that enters our eyes and excites neurons in our retina. It is the process of visual perception that takes these retinal activities and brings to your mind a faithful representation of the objects around you.

At its core visual perception is an inference process: it tries to infer which representations best explain the visual input it receives. As shown in figure 1.1, these representations can simply be a breakdown of the image into its properties. For example, we can describe the image of the purple circle in the box by a fully informative list: its shape, its colour, its location and its size. If we started with this representation, we could indeed create the image again - a process that is called graphics. This duality is why visual perception can be described as inverting the graphics process (G. Hinton 2013).

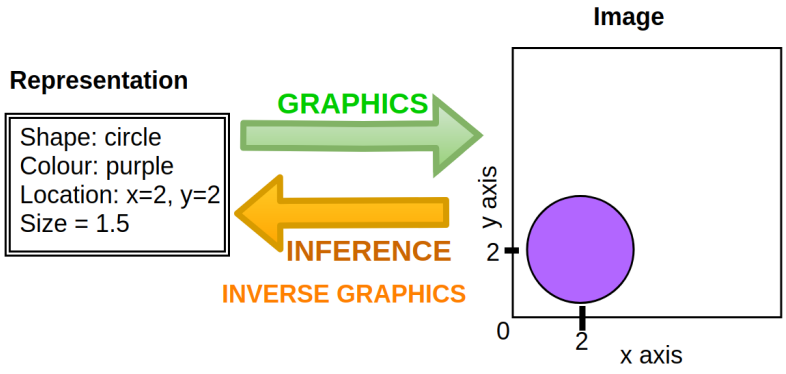

Figure 1.1: Graphics, inference and inverse graphics

The graphics model of the world must follow the optics of image formation. For example, when two objects are one behind the other, if the front one is opaque, light from the back object typically cannot reach us. The effect of only receiving light from the front object, as opposed to a sum of the back and the front, creates a non-linearity that is termed occlusion. With it, all information about the occluded part of the back object is lost. It is this non-linearity in the graphics model of the world that causes inference in visual perception to become ambiguous, as shown in figure 1.2.

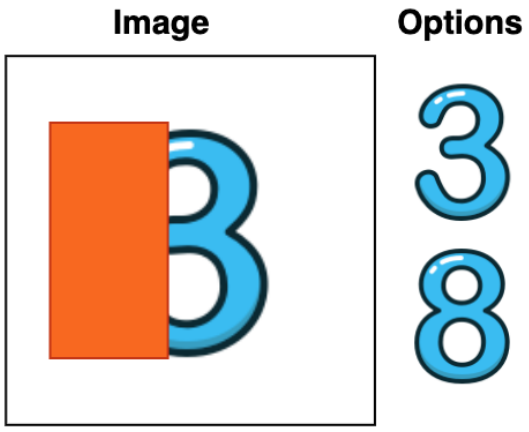

Figure 1.2: What is the identity of the back digit? The non-linearity of occlusion creates ambiguity in this inference.

The example in figure 1.2, is the simplest one can think of. Yet, in a world of much greater complexity, where an infinite number of possible object and their combinations can explain the light that enters your eyes, how the brain is able to do visual inference so efficiently, quickly and robustly, remain a mystery.

One way to understand the computational principles of visual perception is to build models, come up with hypotheses and test them in experimental settings. For a long time , computational models of visual perception were limited to early visual processing and were unable to build system-level models that were able to explain the entire visual hierarchy. But recently, new models based on research in artificial intelligence have emerged. These are more complex, more successful and more flexible than previous vision models, and they are called Artificial Neural Networks.

Artificial Neural Networks

Artificial Neural Networks (ANNs) are the most promising models of the visual perception for a variety of reasons.

Firstly, they can perform well on object detection tasks. This was shown in 2012 when Krizhevksy et al., used ANNs to classify 1.2 million natural images of the ImageNet competition into 1000 different categories, with a the right answer appearing in its top five guesses 75% of the time, around 15% higher than the previous state of the art (Krizhevsky, Sutskever, and G. E. Hinton 2012). Since then, the state of the art ANNs can perform with 98% accuracy on top-5 classification.

Secondly, they are connectionist, like the brain. Connectionist models cannot rely on a central computing unit, like in a computer, that can perform arbitrary operations. Instead, these models have to do everything in a distributed fashion, with localised computational units that perform rigidly defined operations, and can only communicate with a limited set of other computational units. See Box 1 for the basic theory of ANNs and how they related to the brain.

Box 1: Theory of ANNs and Deep Learning

The brain contains billions of cells with special electrochemical properties that connect to each other and are thought to drive the computations of the brain. These cells are called neurons. Neurons connect to other neurons at special interfaces called synapses. These synapses determine the strength of the connection between neurons, and can change over time in a process that we believe is crucial for learning.

The basic computational units in ANNs are a radical simplification of the neurons in the brain. The similarities that are maintained in this abstraction are described below.

Imagine a single unit, called , with a number I of inputs x_i from other units. Associated with each input unit is a weight w_i (i = 1 to I), which represents the strength of the connection between the unit x_i and the unit Z. Each unit also has associated with it a "bias" term, b, which represents how sensitive the unit Z is to its inputs.

From this we want to calculate the value unit Z, denoted as its output y. This value can loosely be interpreted as the firing rate of a biological neuron.

To calculate this output, we apply a specific activity rule, which has two steps.

1. Each input unit is multiplied by its weight. These values are then summed together to give a joint weighted input to the unit of interest and added to the bias term, which can be of the same or opposite sign as the combined input. This operation (see below) gives us the unit's activation a. Mathematically, it can described as:
$$a = \sum_i w_i x_i + b$$

2. A non-linear function to the activity of the unit, to finally give its output. This non-linear function makes sure that any negative activations are turned to zero, matching in abstraction the fact that firing rates cannot be negative. There are many possible activation functions, the most popular being the sigmoid, defined as $f(x) = \frac{1}{1+e^{-x}}$ and the rectified linear unit (ReLU), defined as $f(x) = max(x, 0)$

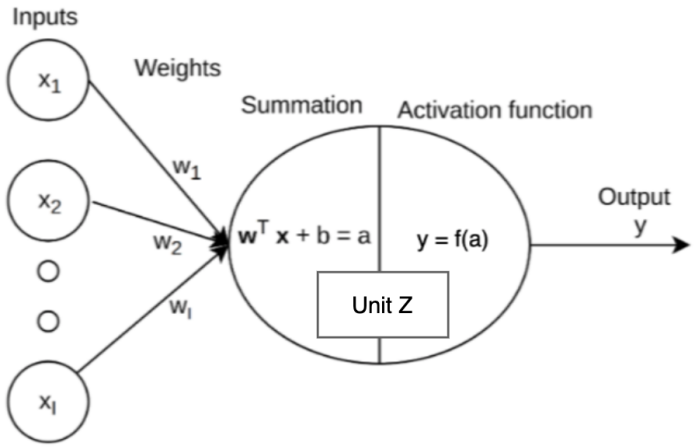

Figure 1.3: Example of a unit and its computations

By connecting many units together one can create a network. In deep learning, units are connected in a very specific architectures. First they are collecting into a 'layer' and then connected to other units in subsequent 'layers'. An example of this architecture is shown in figure 1.4.

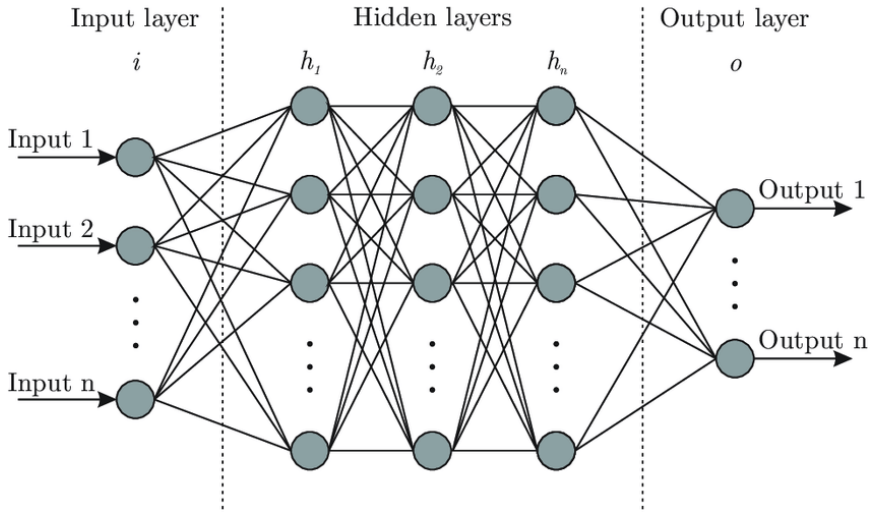

Figure 1.4: Example of a deep ANN, constructed by many units joined together.

Once the architecture and the initial weights are defined, we want to train the network to solve a task. The task is usually to approximate a complex mapping between an input and an output, where, for example, the input is an image and the output is a representation of some relevant properties of the image; for instance, a binary label that indicates whether or not a certain object (e.g. a cat) is present in the image.

To learn this function, the network modifies its weights to create new synaptic patterns.

To do this the output of the network (the activities of the units in the last layer), is compared to a target output. The difference between the network's output and the target output, called the error, is computed. Using an algorithm called backprogagation, the weights of the network are altered so that the when the next input is fed to the network, the computations created by the weights will more likely give the answer that matches the target. This type of learning is called supervised, as it requires human work to create the targets.

If instead, learning is done without any requirement of human interaction, it is called unsupervised.

Limitations and new directions

ANNs are very inspiring for brain research and have become a key element of an emerging field called Cognitive Computational Neuroscience, which aims to join together the behavioural approach from cognitive neuroscience and psychology, with the experimental approach of computational neuroscience and the algorithmic approach of artificial intelligence (Kriegeskorte and Douglas 2018).

However, feed-forward ANNs have shown to have vulnerabilities that we would not expect of a solved vision model. Firstly, they do not generalise well to object recognition

under occlusion Spoerer, McClure, and Kriegeskorte 2017, and secondly, they do not implement feedback connections despite ample evidence that shows the anatomical presence and functional use of feedback connections in the visual stream for object recognition (see Box 2) (O'Reilly et al. 2013).

There is some recent evidence to show that these limitations may be related to each other. Whilst core object recognition (the ability of visual system to rapidly recognise objects despite variations in their appearance, e.g. position, scale, and rotation) has been shown to be possible in a single feed-forward sweep through the visual stream (DiCarlo, Zoccolan, and Rust 2012), object recognition under occlusion produces delayed behavioural and neural responses, and can be disrupted by masking, a magnetic stimulation of the cortex which which is thought to interfere with recurrent processing (Rajaei et al. 2019, Tang et al. 2014).

How feedback connections and recurrent dynamics help achieve object recognition under occlusion is still an open question. The fact that graphics is a much easier computation than inference, as it lacks any ambiguity about the generative process, has led to the rise of a hypothesis that is the key question in this thesis. The hypothesis is that to aid in the inference process, the brain is able to make use of a internal graphics model in the form of feedback connections that can impose a self-checking mechanism on the inferences, by testing whether the inferred representations would render into an image that matches the observed input.

In this thesis I first investigate the hypothesis that adding recurrent connections can improve performance in cases where graphics is easier than inference and so the former can help the latter. We choose to look at occlusions as a ubiquitous example of the non-linearity that adds ambiguity to visual inference. Secondly, I examine the trained models to gain an insight into the specific role of recurrency for visual object recognition under occlusion.

To do this I train different ANN to perform two different tasks:

1. To infer depth, object identity and location from images with occlusions. This is called the **inference task**.

2. To generate images with occlusions from pre-set representations of the objects in the images. This is called the **graphics task**.

In each case, I investigate the role of recurrent connections in performing these tasks. These tasks also both supervised.

Finally, I train a network that combines the two tasks into one, from images to a compressed representation and then back, to investigate whether a better representation emerges and whether the network learns an insightful way to represent depth and compute with occlusions. This is called the **unsupervised task**.

Box 2: Feedback, recurrency and analysis-by-synthesis

As shown in figure 1.5, as information flows from the retina to higher cortical areas, its abstraction increase. These connections however, flow also down from the higher cortical areas towards the retina. How these feedback connections aid in visual processing is still unknown. A theory, called analysis-by-synthesis, suggests that to perform visual perception the brain internalises the generative model of the world, and then able to use it to aid in the inference process performed.

One way analysis-by-synthesis could be achieved is by feedback connections and recurrent dynamics. One possible role of the feedback connections is to try to predict the inputs the their layer (Rao and Ballard 1999). When a new stimulus arrives, the difference between the top-down prediction and the bottom-up activities creates an error, that is then used to improve improve the performance of the higher levels, and therefore, improve its representation of the input, until equilibrium is reached. If it were to be unrolled over time it could described by the diagram in figure 1.6, where the feed back connections are they key aspect of the ability to both deal with competing hypotheses and ambiguity in inference as well as driving learning.

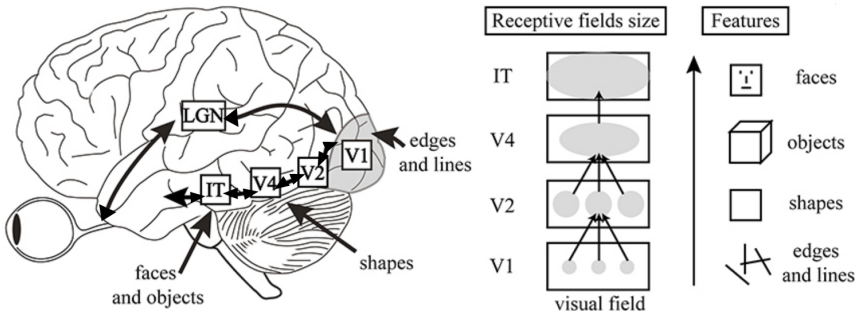

Figure 1.5: Example of hierarchy in the visual system

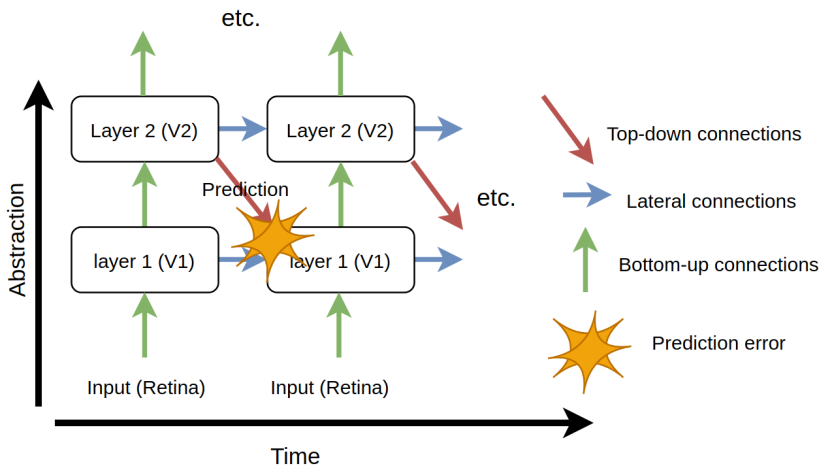

Figure 1.6: Example of feedback connections implementing predictive learning

A second way recurrent connections could help, is to perform an iterative process, similar to the expectation maximisation algorithm, where bottom-up connections act as the expectation step, and the top-down as the maximisation step (G. E. Hinton, Sabour, and Frosst 2018). In an ambiguous image of occluding digits, for example, two things need to be inferred: the digit identities and their segmentations in the image. The bottom-up could initially make a guess about the identities. This guess is then used by the top-down connections to refine the segmentation in the image. This provides a new, and hopefully better guess of the digit identities, and so on.

These types of computations an be constructed in artificial neural networks in the same way as shown in figure 1.6, with top-down connections, lateral and bottom-up connections between layers of increasing abstraction. The main difference, however, is that in deep learning, whilst recurrency can be implemented, it is not explicitly the

driver of learning. Instead, the error that drives learning is between the output of the network and its target.

Chapter 2

Methods

Data sets

In all three tasks, ANNs either receive or have to have to generate images of objects with occlusion. In this section I will describe the process of generating the images used to train these networks, and in the section after I will describe how these images are used in the three tasks.

Creating images

To test the ability of the networks to do to inference and graphics on objects under different levels of occlusion, we created three datasets of increasing difficulty, called **solid2**, **border2** and **border3**. The objects in each case were digits from the range 0 to 9.

Digit identities can be described by their shape, colour and size. In all the datasets, the font type and the size were fixed. The colours instead were varied. In the simplest case, in dataset solid2, only two digits were present in each image, one fully black and one fully white. In the more complex datasets border2 and border3, the digits all took the same form: black with a white outline, allowing either two digits to be present in each image, as in border2, or three, as in border3.

Each digit also have a specific vertical, horizontal and relative depth location in the image. For each digit the horizontal and vertical values were sampled from a uniform distribution. The relative depth instead was created by the order in which these digits were drawn onto the image. Occlusion is therefore created by the iterative drawing of digits on top of one another.

The images were first rendered at a resolution of 256x256 pixels, and then downsized to 32x32 for computational practicality. Figures 2.1, 2.2 and 2.3 show examples of this conversion. The pixels values were also normalised to between zero and one, with black being zero, 0.5 being grey (the background) and one being white. This was found to be a much more reliable way of creating high quality images than rendering directly in 32x32 pixel space.

Each dataset consisted of 100,000 images used to train the network (the training set) and a smaller 10,000 images used to evaluate the network's performance once trained (the test set). For a fair evaluation, the test set needs to include images that have never been seen by the network during training. This was achieved by pairing the ten digits with each other and never showing the specific pair of digits in the same image in the training set. For example, there were no training images in which a '0' appeared together with a '7',

but ’0’ could appear in combination with any other digit. The testing set, on the other hand, contained images of only these digit pairings, for example ’7’ and ’0’ or ’1’ and ’6’ etc. This approach allowed us to evaluate whether a network had actually learned the compositional nature of the images. A network that truly understands occlusions ought to be able to render or recognise an image with any combination of occluding digits. On the other hand, a network that has simply memorised the appearance of specific pairs of occluding digits should perform poorly on those combinations it was not specifically trained on.

The original code for these data sets was taken from Spoerer, McClure, and Kriegeskorte 2017 and adapted for these experiments. A summary and examples of the difference data sets are shown in table 2.1 and figures 2.1, 2.2 and 2.3.

Data set	number of digits	digit colour
solid2	2	one black one white
border2	2	both black with white outline
border3	3	all black with white outline

Table 2.1: Comparison of data sets

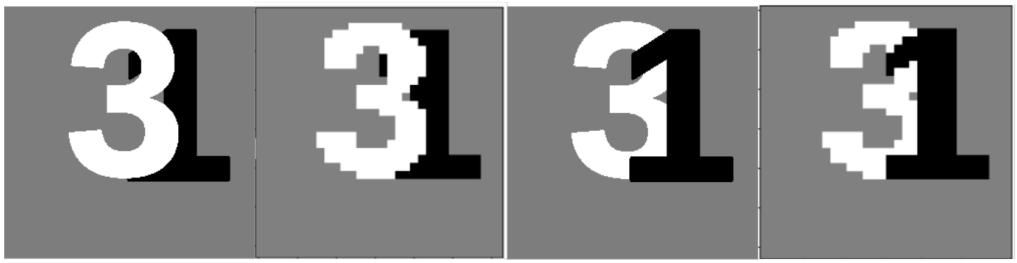

Figure 2.1: Example images from dataset A: initial resolution and at 32x32 pixels

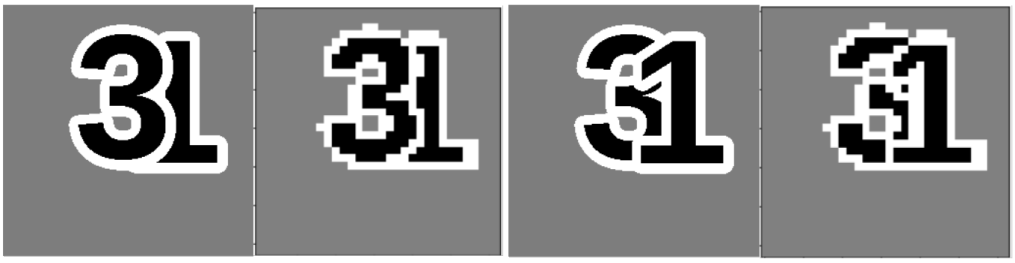

Figure 2.2: Example images from dataset B: initial resolution and at 32x32 pixels

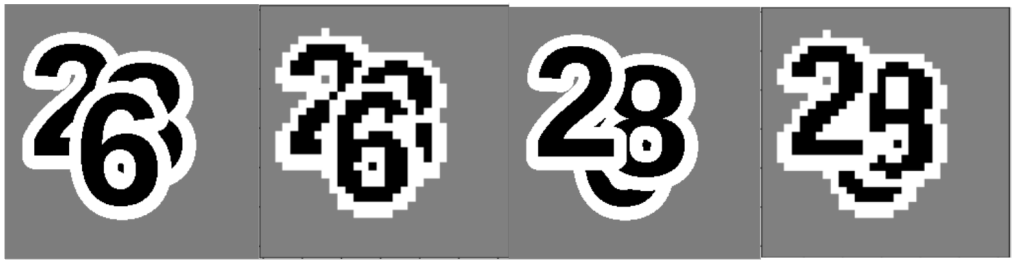

Figure 2.3: Example images from dataset C: initial resolution and at 32x32 pixels

Application of datasets for tasks

In the inference task, for every image, the network tries to match a pre-set representation of that image. For the solid2 dataset, we trained the networks on two different representations. These were called the 'colour-bound' and the 'depth-bound'. The colour-bound representation order the one-hot digit identities in the target vector by their colour, as is shown in figure . This creates a task that is purely of digit recognition. The depth-bound representation instead orders the digit identities by the relative depths in the image. With this representation, the networks have to both identify the correct digits, and their depths, and route the information correctly to the appropriate part of the target vector. Figure 2.5 shows an example of the depth-bound representation.

This depth-bound representation was also used for the border2 and the border3 datasets. Figure 2.6 shows an example of the depth-bound representation for the border3 dataset.

In the graphics task, we train the ANNs to take in a pre-set representation and output an image that matches that representation. To train the network on the solid2 dataset, the colour-bound representation was used. The networks trained on the border2 and border3 datasets instead used the depth-bound representation.

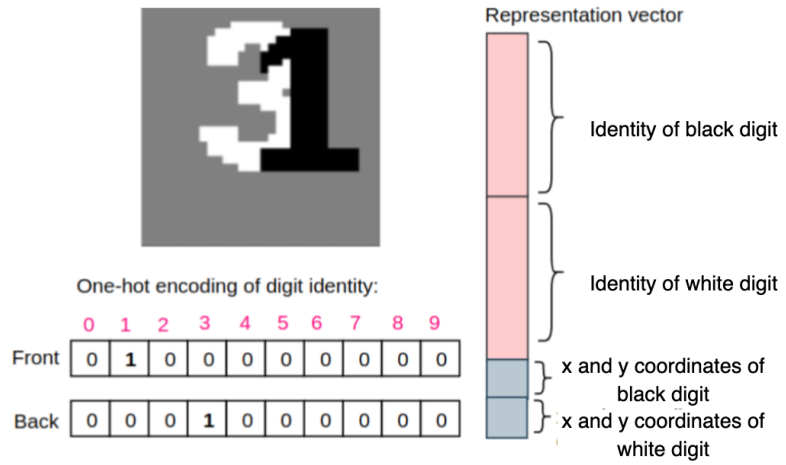

Figure 2.4: Example of the colour-bound representation used with the solid2 dataset

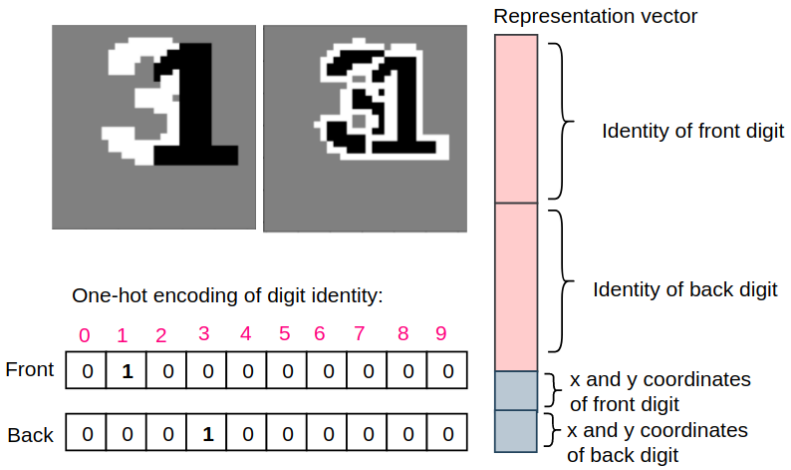

Figure 2.5: Example of the depth-bound representation used with the border2 dataset

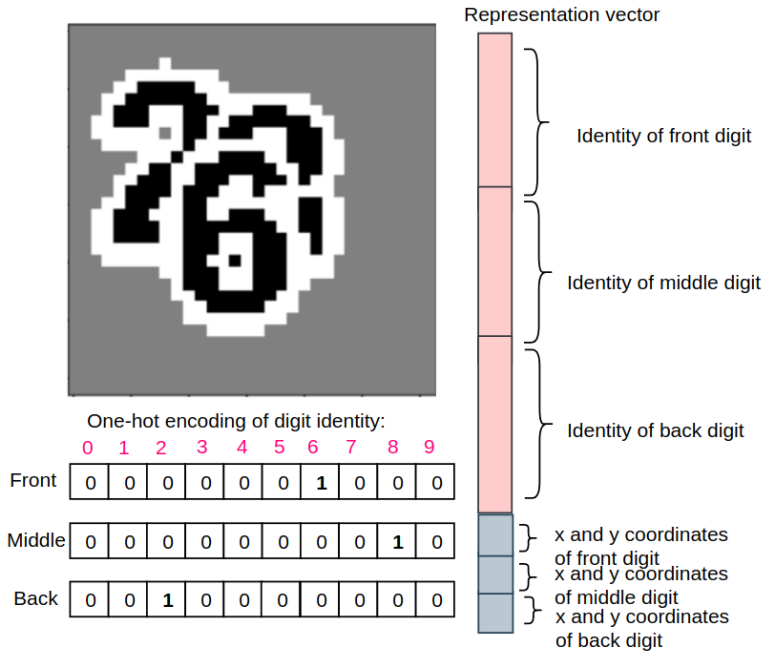

Figure 2.6: Example of the depth-bound representation used with the border3 dataset

Models

Convolutional and transposed convolutional layers

All the ANNs trained for the different tasks consist of a mixture of convolutional layers, transposed convolutional layers and fully connected layers. In this section, I will introduce the former two.

Every image can be represented as a matrix of pixel values. The height of and width of the matrix correspond to the height and width of the image. If the image is grey-scale, the depth of this matrix is one, whilst if the image if of colour, the depth is of three, where each depth layer corresponds to the red, blue and green (RGB) components of that pixel. In our datasets, all images are grey-scale, with height 32 and width 32, meaning that the height, width, depth dimensions of the matrix of pixels are (32,32,1).

Images typically can be broken down to a small number of components that occur very often and make up the image, for example a vertical line, or a diagonal edge etc. In order to detect where each of these components are in the image, a matrix called a kernel is moved across the image, multiplying the values each element within its matrix with the corresponding in the image, and summing at every step. Figure 2.7 shows an example of this process, called a convolution. As hinted in the figure, the values within this kernel matrix can be learned, such that kernel becomes responsible for detecting the location a specific feature in the image, with the feature being chosen to optimises the task at hand. For example, if the weights learned to take the values: $w_1 = -5, w_2 = 5, w_3 = -5, w_4 = 5$, then the kernel learns to take responsibility for the detecting vertical edges. By moving around the image, it produces an output that only highlights the vertical edge. This output is shown at the bottom right of figure 2.7.

The step size of the kernel, called the stride, is shown by the green arrow. In the

example in the figure, the stride is of one.

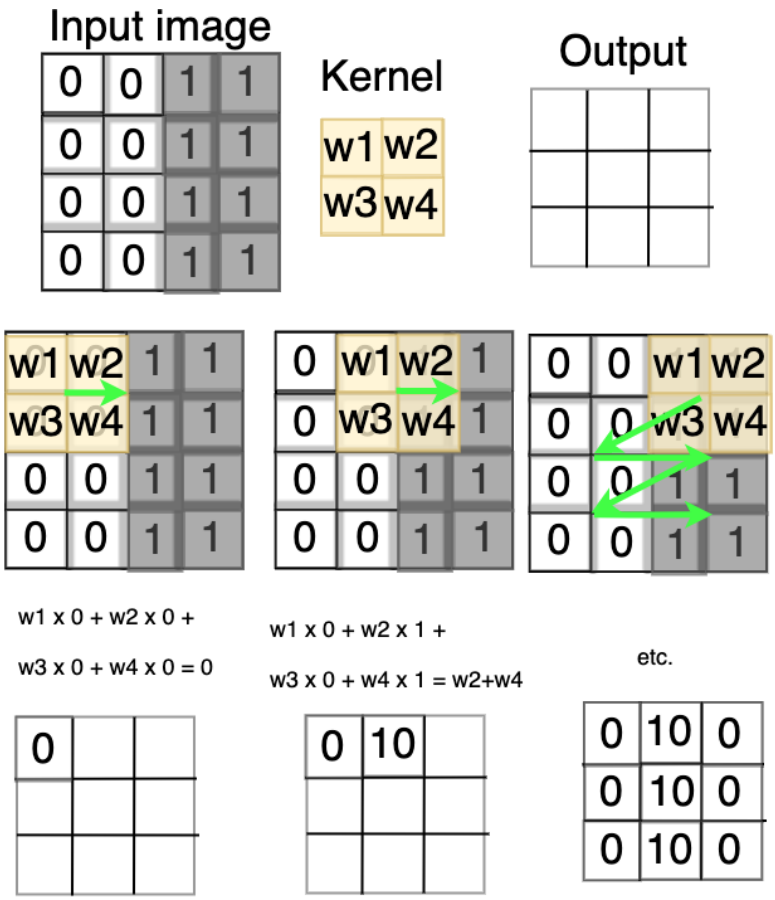

Figure 2.7: Example of a convolution over 2D image

If the kernel is run over the image as shown in figure 2.7, the pixels in the middle of the image will be used as input more often than the pixels at the edge of the image. In order to remove this effect, input images can be 'padded' with surrounding layers of zero-valued pixels, as shown in figure 2.8.

Figure 2.8: Example of padding in a 2D image

In ANNs, 'layers' of units can be organised in cuboidal as well flat dimensions. In the inference task, the ANN is given images as inputs. This means that the pixel values of the image become the activations of the first layer of the ANN. If a convolution is run

with a single kernel, a single output 'image' will be created. This 'image' actually then corresponds to the values the first 'filter' of the second layer in the ANN. If a second kernel is run over the input layer, then a second filter is stacked in depth behind the first in the second layer and so on.

To run a convolution on a matrix with depth greater the one, the kernel itself becomes a cuboid that can then be moved around the 3d input as shown in figure 2.9.

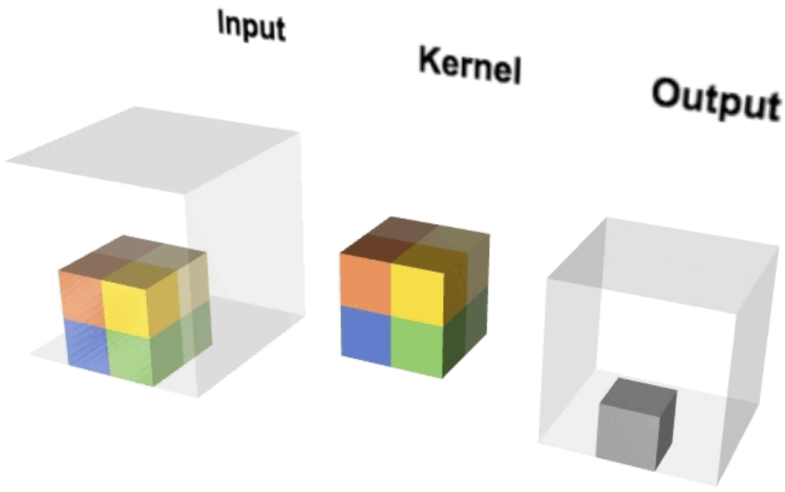

Figure 2.9: Example of a convolution over a 3D cuboid

There are a certain number of parameters that can affect the behaviour of convolutions. These are the kernel height and width and depth, the stride and the padding. The choice of these parameters affects the height and width of the output image, such that it can either be of the same size as the input image or smaller. Convolutional layers are used therefore, to typically compress images.

Transposed convolutional layers aim to do the opposite, and from a small image converts it to a larger one. This process follows a similar set of steps, with the main difference, as shown in figure 2.10, being that the kernel size and stride relate to the output, not the input, and that every time a pixel is covered in the output, its value is summed to its previous.

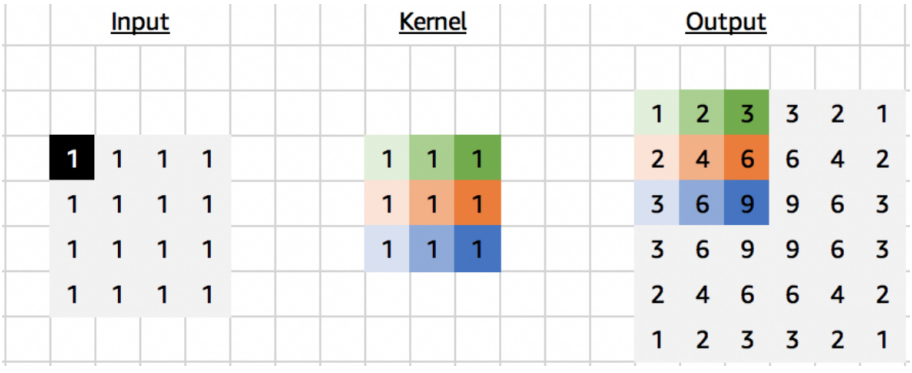

Figure 2.10: Example of a transposed convolution

Recurrent connections

To investigate the role of feedback connections for inference and graphics we took inspiration most directly from Spoerer, McClure, and Kriegeskorte 2017, and previous work that built ANNs with feedback modelled after the architecture of the visual stream (Liao and Poggio 2016, Liang and Hu n.d.).

In these papers, and in this thesis, the convolutional networks are split into having different combinations of bottom-up (B), lateral (L) and top-down (T) connections. These connections correspond to feedforward information processing pathways (B) from lower regions of the visual stream , feed back pathways (T) from higher to lower regions, and pathways within each region of the visual hierarchy (L). Figure 2.11 shows a simple example of the B, L and T connections between two layers. The arrows refer to some processing, that in this example can be done by a convolution or a transposed convolution.

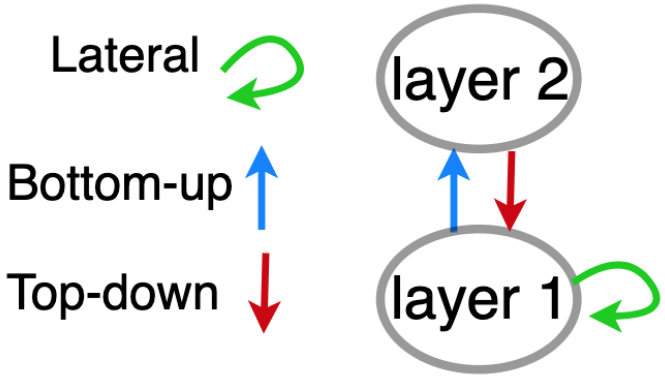

Figure 2.11: Simple example of B, L and T connections

Adding top-down and lateral connections to the feedforward B networks creates recurrent dynamics that can be unrolled over time, as shown in figure 2.12. At each time point, the input stays the same, and the outputs of the B, L and T computations are added element-wise. Crucially, the weights associated with the B, L and T computations also remain the same over each time step.

To recap, convolutions are used to decrease or maintain the dimensionality of the input. Transposed convolutions instead are used to increase the dimensionality of the input. Because the outputs are combined by element-wise summations, they have to be the same size. Therefore, L computations are always convolutional computations that keep theirs outputs in the same height and width as their inputs. The B and the T connections, however, switch depending on the task.

In the inference task, the network needs to convert an image to an abstract representation that requires decreasing its dimensionality. To do this, the bottom-up (B) connections are convolutions that half their inputs' height and width. The T connections instead become transposed convolutional computations that double their inputs' height and width.

In the graphics tasks, the direction of dimensionality is reversed, as we start from a low dimensional representation and want to build a high dimensional image. Therefore, the T and B connections switch roles, such that the B becomes a transposed convolution computation and the T a convolutional one.

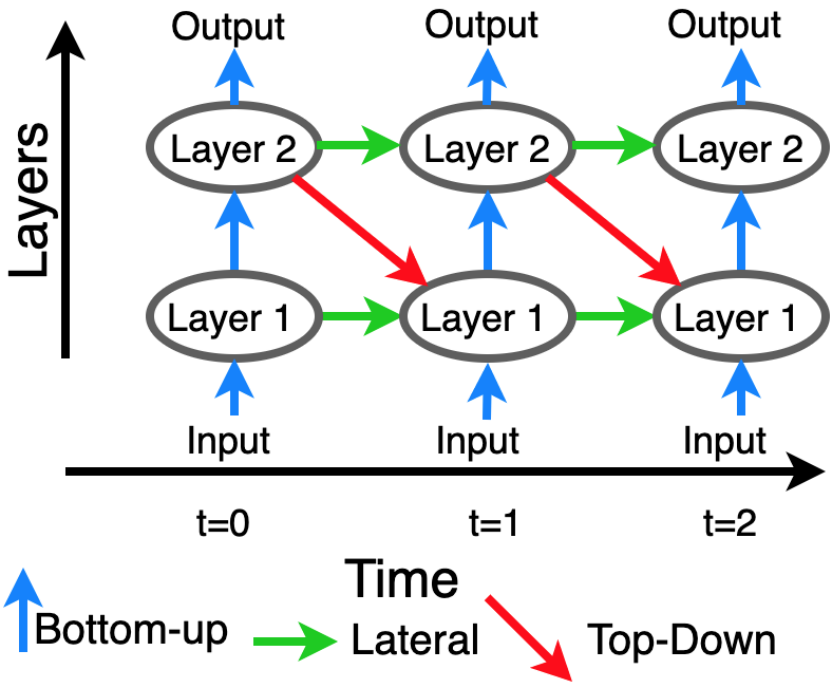

Figure 2.12: Simple example of unrolling a BLT network

The connections in these recurrent networks can be described more precisely in the following way.

The input to each arrow can be denote as $\mathbf{h}_{t,l}$ where t is the iteration time and l is the layer number. In this way, $\mathbf{h}_{t,0}$ is the input to the network. For a B network, the pre-activation at time step t for layer l is defined as

$$z_{t,l} = (\mathbf{w}_l^b)^T \mathbf{h}_{t,l-1} + b_l$$

where t is 0 (as B runs for a single time step), and $\mathbf{w}^b$ is the bottom up convolutional/transposed convolutional weights of that layer.

For BL networks, the lateral inputs are added to the activation without new biases, giving:

$$z_{t,l} = (\mathbf{w}_l^b)^T \mathbf{h}_{t,l-1} + b_l + (\mathbf{w}_l^l)^T \mathbf{h}_{t-1,l}$$

The lateral connections rely on the activity of the same layer at the previous time step. Therefore when t=0 the lateral connections are null.

In BT, the top-down connections are added to the bottom-up instead of the lateral, giving:

$$z_{t,l} = (\mathbf{w}_l^b)^T \mathbf{h}_{t,l-1} + b_l + (\mathbf{w}_l^t)^T \mathbf{h}_{t-1,l+1}$$

In these model, the top down connections can only be received from other hidden layers.

Finally, both the lateral and the top down can be added to the B networks giving the BLT:

$$z_{t,l} = (\mathbf{w}_l^b)^T \mathbf{h}_{t,l-1} + b_l + (\mathbf{w}_l^l)^T \mathbf{h}_{t-1,l} + (\mathbf{w}_l^t)^T \mathbf{h}_{t-1,l+1}$$

As described in Box 1, once the activation z is computed, it is passed through a non-linear function to create the output. The non-linearity chosen in these networks was the rectified linear unit (ReLU) non-linearity, defined as:

$$\sigma(x) = max(0, x).$$

After this, local response normalisation was also applied. This is defined as

$$\omega^{i}_{x,y} = \frac{a^{i}_{x,y}}{\left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} \left(a^{j}_{x,y}\right)^2\right)^{\beta}}$$

where $a(i, x, y)$ represents the i_{th} convolutional kernel's output (after ReLU) at the position of (x, y) in the feature map, $\omega_(i, x, y)$ represents the output of local response normalisation and N is the depth of the convolutional ouput cuboid, otherwise called the number of channels. The parameters n, k, α and β can all chosen arbitrarily. k is typically one and is used to avoid divisions by 0. α is used as a normalisation constant and β as a contrast constant. Finally n is used to define the neighbourhood length i.e. how many consecutive pixel values need to be considered while carrying out the normalisation. Values for these hyperparameters were chosen to match those in Spoerer, McClure, and Kriegeskorte 2017, as the datasets used in these thesis was similar to the ones used in the paper.

Putting it all together, the output of layer l at time step t is:

$$\mathbf{h}_{t,l} = \omega(\sigma(\mathbf{z}_{t,l}))$$

Models for inference

Task

As light hits your retina, it excites rods and cones that are arranged in a grid. The activity of each neuron in this grid is analogous to the pixel value of a digital image. Imagine an image that is of size 100x100 pixels, and that each pixel can only take two values, either 0 or 1. This gives 2^{10000} possible images. Our eyes have around 6 million cones each, and even if we assume that these cones can only be on or off, this allows for $2 \times 2^{6,000,000}$ distinct input patterns. In the natural world, however, the images we see are a very small subset of this huge set of possible distinct input patterns. This is because the images we have evolved to interpret are not random pixels, but rather highly correlated in space and time. It is because of this that visual inference is possible.

In the inference task, we train a neural network to infer depth, object identity and location from images with occlusions. These networks are called **encoders**, as their aim is to encode high dimensional images into low dimensional representations (see figure 2.13). This process is exactly the inference that is required in visual processing to understand the causes, or objects, that best explain the retinal activities. This kind of learning is also called a discriminative learning, where the aim is to discover $p(\mathbf{y}|\mathbf{x})$ where $\mathbf{x}$ is the input data and $\mathbf{y}$ is the low dimensional representation.

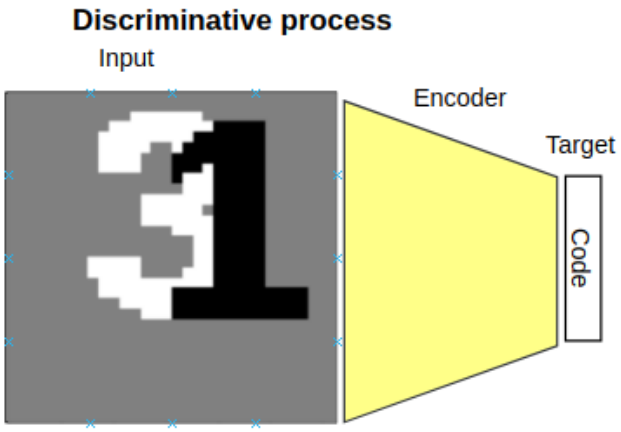

Figure 2.13: Example of discriminative process with an encoder network

Architecture

To test the role of recurrency in the inference task, we trained five encoders, two with only feedforward connections and three with recurrent connections (BL, BT, BLT).

The size of a network is described by the number of parameters that can be optimised for the task. These include the weights between units, and the bias terms inherent to each unit. The process of adding feedback connections adds more weights, and therefore more parameters. Typically adding more parameters give the network a greater capacity to perform on tasks. To test that changes in performances were not due to changes in parameter numbers, but instead to the specific computations induced by changes in the network architectures, we built a modified B encoder, called B-matched, that had the same feedforward architecture as the B encoder, and the highest number of parameters out of all the encoders.

There are two ways of increasing parameters in a convolutional/transposed convolutional network: increasing the number of filters and increasing the kernel size. The most appropriate way in the scenario is to increase the kernel size, as it mimics recurrency by increasing number of connections between two layers. However, each increment of kernel size adds or subtracts many parameters at once, so in order to get a closer match we also changed the number of filters.

In the encoder architectures, the recurrent convolutional layers describe in the sections above are the start of the the network. The input image is passed through 3 bottom-up convolutions, each time its height and width being halved, until it becomes a 32x4x4 dimensional cuboid. At this point, the cuboid is flattened and weights each unit connected to 256 units with an all-to-all connection pattern. This is an example of a adding fully connected layer.

Two more fully connected layers follow, the first of the same dimensionality of 256 units, and the last of the dimensionality the representation. As described in Box 1, the activation of each unit in one of these layers is a weight sum of the outputs of the previous layer multiplied by respective weights and add to a bias term. To convert this activation to an output, the ReLU non-linearity is used.

Figure 2.14 shows a diagrammatic representation of the B, B-matched, BL, BT and BLT encoders. The squares represent cuboid-shaped layers, and the ovals represent vector-shaped layers. The arrows represent the weights that connect one layer to the other. These

also represent the type of computation that is performed, for example blue bottom-up arrows represents convolutions that halve the height and width of their input; the red top-down arrow represents transposed convolutions that double the height and width of its input and and green lateral arrows represent convolutions that maintain the same size input to output. Finally, the yellow arrows represent the all-to-all connections between two vector-shaped layers.

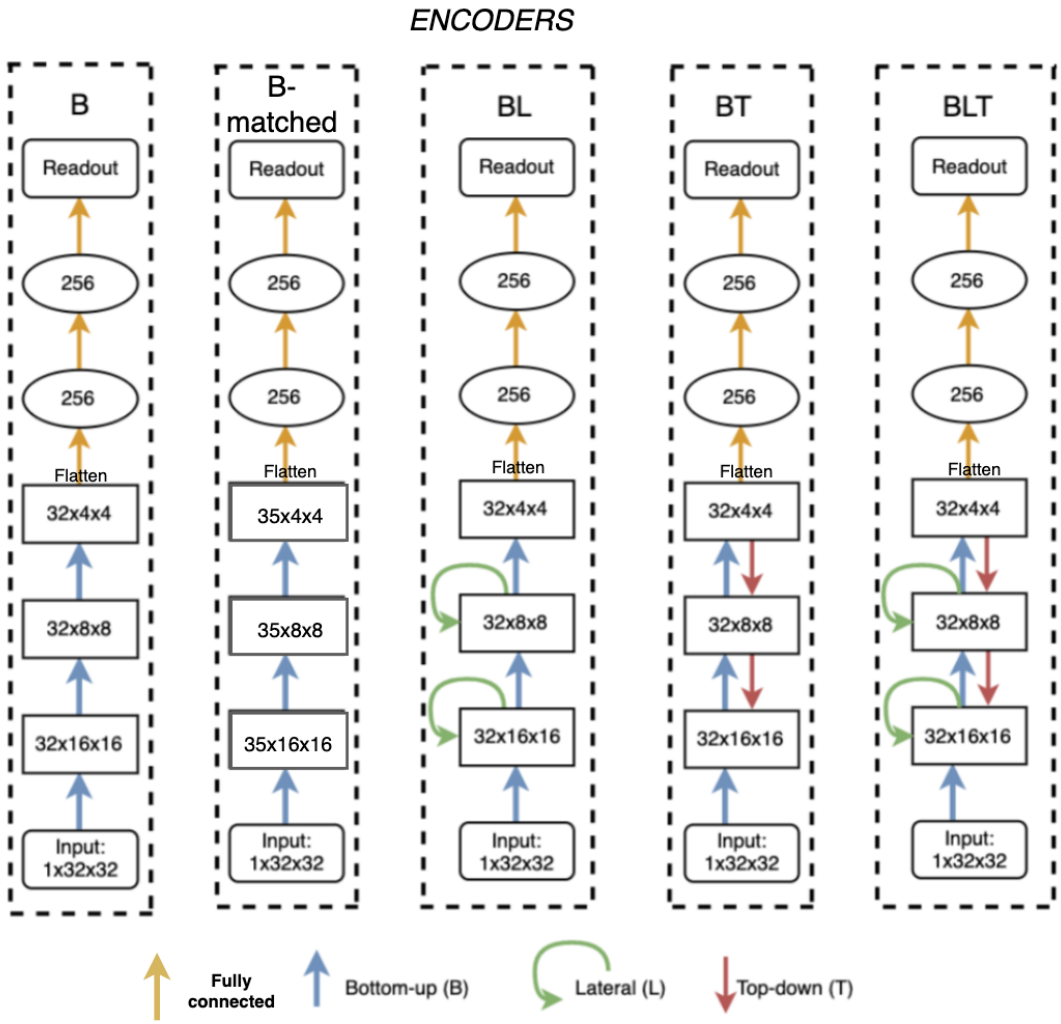

Figure 2.14: Schematic diagrams for each of the encoder architectures used. Arrows indicate bottom-up (blue), lateral (green), top-down (red) convolutions and fully connected layers (yellow).

Learning

At each time step, the encoders give an output from the readout layer. In the B and B-matched, the computation finishes at time step 0. In the BL, BT and BLT networks instead the computations were run for 4 time steps.

The objecting in training is to match the readout to the target representation. To match the digit identities the appropriate parts of the readout (see section 2.1.2) were put through a softmax function to map the non-normalised output activities to a probability distribution over predicted output classes.

A cross entropy loss was used to calculate the error between the softmax digit identities and the target digit identities. If $\hat{\mathbf{y}}$ is the softmax output and $\mathbf{y}$ is the target, cross entropy is defined as:

$$H(\hat{\mathbf{y}}, \mathbf{y}) = \sum_i y_i log(\hat{y}_i)$$

A mean squared error loss term was also calculated between the readout values that encoded the locations of the digits and the target locations. This is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

where n is the size of the locations vector, $\hat{y}$ is the network output vector and y is the target vector.

L2-regularisation, with a coefficient of $\lambda = 0.0005$. This is defined as:

$$L2Loss = \lambda||\mathbf{w}||_2$$

where $\mathbf{w}$ is the vector of all trainable parameters and square of the summed values of the parameters.

This makes the loss function for each time step:

$$Loss = CrossEntropyLoss + MSELoss + L2Loss$$

The losses for each time step were then added together to create one final overall loss. This encourages the networks to converge to their best answer from the first time step, mimicking the fact that biological visual processing is often correct after a single feed forward sweep.

Networks were trained by minimising this loss function with respect to the connection weights. We used stochastic gradient descent with a batch size of 100 images, and parameter-wise learning rates scaled by the ADAM method with default parameters $beta_1 = 0.9$ and $beta_2 = 0.999$.

A weight decay rule was also set up, such that every 40 epochs the learning rate decreases by a factor of 10. This is given by:

$$\epsilon_n = \eta\delta^{e/d}$$

where ϵ is the new learning rate, η is the initial learning rate, δ is the decay rate e is the epoch (a whole iteration through the training inputs) and d is the decay step. Other than the initial learning rate, all the networks were trained with the same parameters: $\delta = 0.1$ and $d = 40$.

Models for graphics

Task

Whilst in inference the challenge is how to deal with the ambiguity of the input images, the challenge for graphics is how to draw occluding objects. This requires selecting which parts to draw and which not to, so as to create the illusion of depth.

In a non-connectionist framework this is easy as, one can draw the back digit, place the pixel values in some random accessible memory, and overwrite pixels values to match

the next object in front. From a connectionist perspective, however, it is not clear how it would be able to achieve the same effect, a computations in ANNs are not performed in an arbitrary sequence, but rather with all units working at the same time, so there doesn't seem to be a natural ability for "overwriting" a existing values.

By adding feedback connections, recurrent dynamics start to occur within the network. It may be that through dynamics, ANNs can match the symbolic process of creating images with occlusion.

Therefore to investigate the role of recurrency in the graphics task, we built, trained and tested five ANNs, which we refer to as decoders, as they take the representation and decode it to create an image, as shown in figure 2.15.

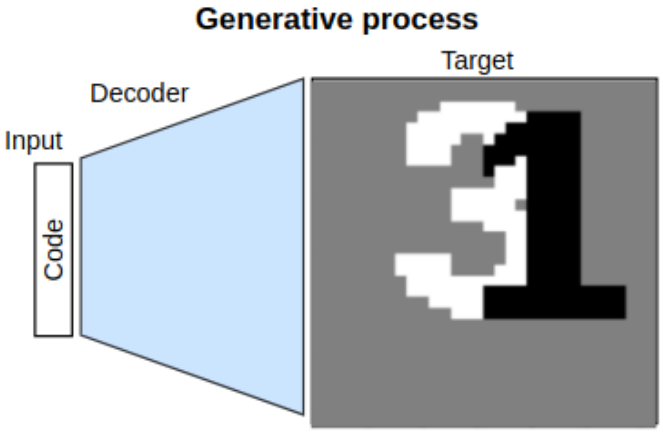

Figure 2.15: Example of generative process with a decoder network

Architecture

Similarly to the encoders, two of the decoders were feedforward (B, and B-matched), and three were recurrent (BL, BT, BLT).

Like the encoders, the decoders also included three fully connected layers. These, however, were placed at the beginning of the network. Figure 2.16 shows a diagrammatic depiction of the five architectures. Similarly as in the encoders, flat-dimensional layer are shown as ovals, and cuboid shaped layers are shown as rectangles. The depth, height and width of these cuboids is written within each rectangle.

The blue arrows, as described in the recurrent section above, refer to transposed convolutions which double their inputs' height and width. The lateral arrows refer to normal convolutions that maintain the size. The top-down connections refer to the convolutions that half their inputs' height and width.

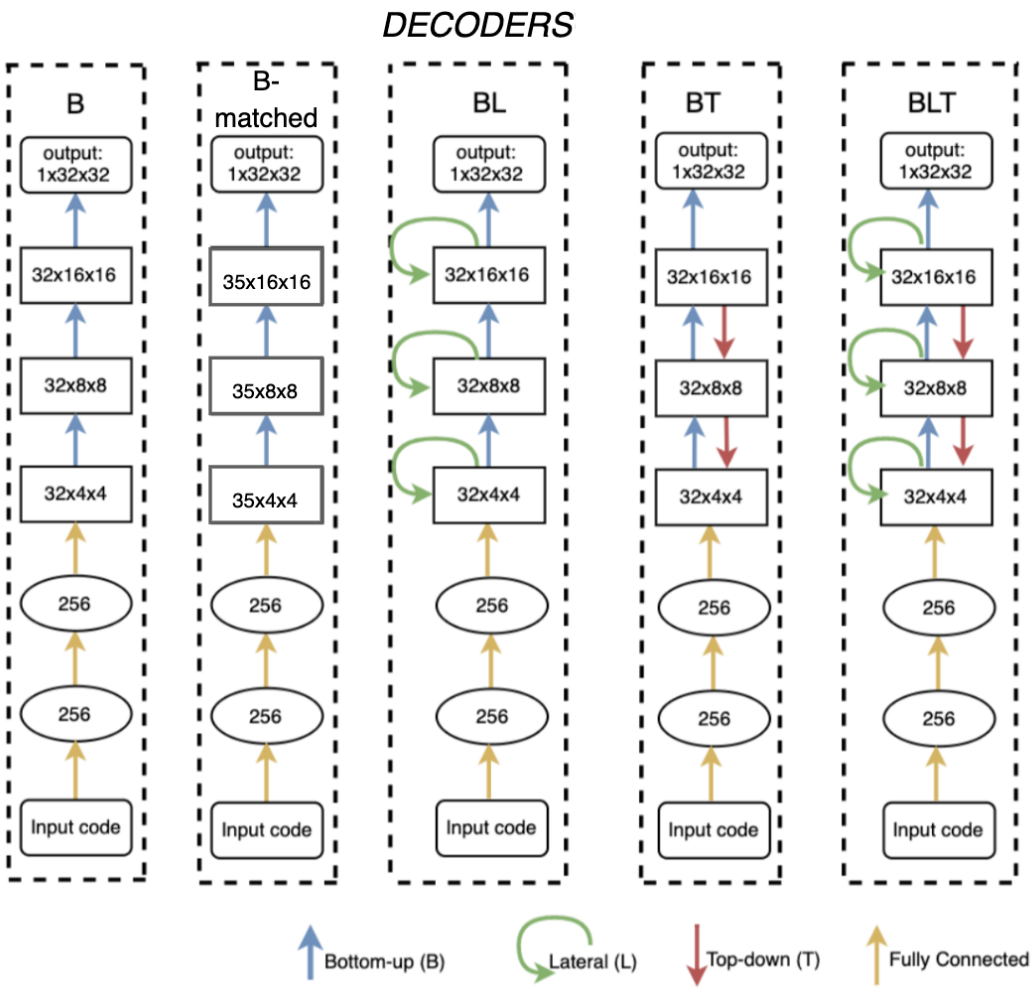

Figure 2.16: Schematic diagrams for each of the decoder architectures used. Arrows indicate bottom-up (blue), lateral (green), top-down (red) convolutions and fully connected layers (yellow).

The recurrent dynamics that occur by adding feedback connections can be unrolled over time, and viewed in a clearer fashion. Figure 2.17, is an example of the BLT decoder unrolled over four time steps.

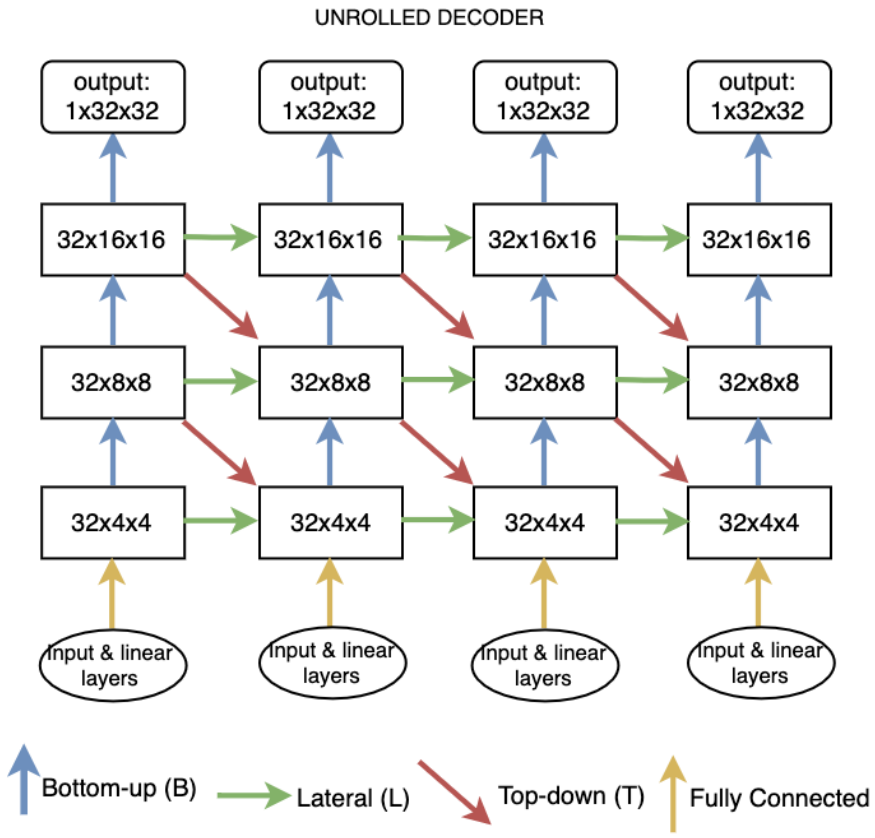

Figure 2.17: Computational graph of the decoder BLT unrolled over four time steps. Arrows indicate bottom-up (blue), lateral (green), top-down (red) convolutions and fully connected layers (yellow).

Learning

At each time step the decoders produce an 1x32x32 sized image as their output. The objecting in training is to match the output image to the target image.

To do this, a MSE loss was calculated. The overall loss, like in the encoders, was a sum of the losses at each time step. An L2 loss was also added, making the final loss:

$$Loss = jointMSELoss + L2Loss$$

This loss was backpropagated using the same ADAM optimisation method and weight decay rule.

Models for unsupervised learning: Autoencoders

Task

The unsupervised task asks if by combining the encoder and the decoder networks into one, a better representation of the occluded objects emerges. The name given to the joint encode and decoder network is an autoencoder.

As both the input and the output are images, this type of problem is unsupervised. However, we want the autoencoder to learn that the occlusion in the images it sees as inputs

can be explained by the concept of relative depth. To do this, we trained the autoencoder to reconstruct an image of the digits in the input but with the relative depths inverted (see figure 2.18 for an example).

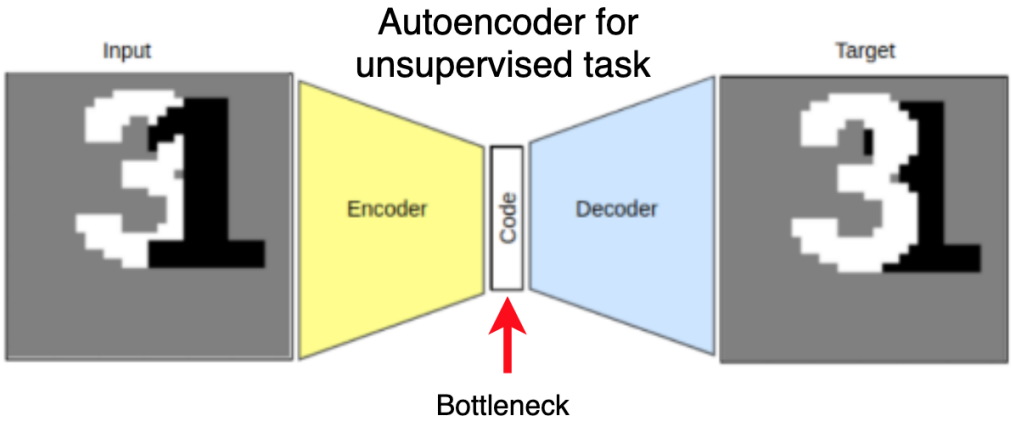

Figure 2.18: Example of autoencoder network with an encoder and decoder, each of which can have recurrent dynamics.

Architecture

Only one autoencoder was trained. This was made up of a a BLT encoder and BLT decoder jointed together by bottleneck code.

A key aspect of the autoencoder structure is the bottleneck in the middle of the network. Information has to pass through the whole network, and has to fit through the narrowest part of the network with the least capacity to encode information. Therefore, bottlenecks encourage compression of the input image, in order to preserve as much as possible the relevant information needed to construct the output image. The question therefore is what kind of representation emerges in this bottleneck layer.

Learning

At each time step, the autoencoder outputs an image of size 1x32x32. The object in training is to match the output image with the target image. To do this, the MSE loss was calculated. However, unlike encoders and decoders, only the MSE loss for the final iteration was used in training. This was an arbitrary decision, and one that, in future work, I would change to match the cumulative loss of the decoders. Also unlike the encoders and decoders, no L2-regularisation was applied.

Analysis and comparison of model performance

To test the ability of the encoder networks on the inference task, we measured their performance by calculating the accuracy of their predictions across the test set. The encoders had to predict both the digit identity and its depth layer to be correct. The accuracies shown in the results below, therefore, focus on the accuracy of the front digit, the back digit, and, for the dataset border3, the middle digit.

The accuracy of the encoders was compared within datasets by performing pairwise McNemar's tests between all the combinations of trained models. This test uses the variability of performance across stimuli as the basis for statistical inference Dietterich 1998.

This test does not require re-training models on different random seeds, as this is computationally expensive and the results converge to very similar performances. However, as the accuracies of the digits in each image are not independent, a variation of McNemar's test that takes this into account was used Durkalski et al. 2003. To minimise false positives from multiple testing, we controlled the false discovery rate (FDR) at 0.05 for each group of pairwise tests using the Benjamini-Hochberg procedure. No statistical tests were performed to compare the inferred locations by the encoders.

To compare different decoder models on each dataset, the average reconstruction error on the full test set was computed and compared using a paired t-test.

Chapter 3

Results

Inference

In order to test the role of recurrency in the ability to recognise and localise objects under occlusion, we trained and tested five encoders with different levels of recurrency. These were the simple feed-forward encoder (B), the feed-forward matched for parameters (B-matched), the feed-forward with lateral connections (BL), feed-forward with top down connections (BT), and the feed-forward with lateral and top-down connections (BLT).

These networks were each tested on the three datasets, solid2, border2 and border3, creating increasingly difficult tasks. The tasks in each case were to correctly identify the digits in the image and order these digit by depth, or by colour. Details on these target representations are in the methods.

Encoders on the Solid2 dataset

The first set of experiments was on the solid2 dataset. First we tested the ability of encoder networks to accurately detect the two digits without any explicit representation of depth, as shown in figure 3.1. The black and white digits in the dataset were equally likely to be at the front or at the back of the image, so these accuracies are representative of an average accuracy of the front and the back digits. With increased number of parameters, the B-matched performs better than the B encoder. However, B-matched still is not as good as the recurrent networks. Statistically significant differences are shown as black boxes in the grid in figure 3.2.

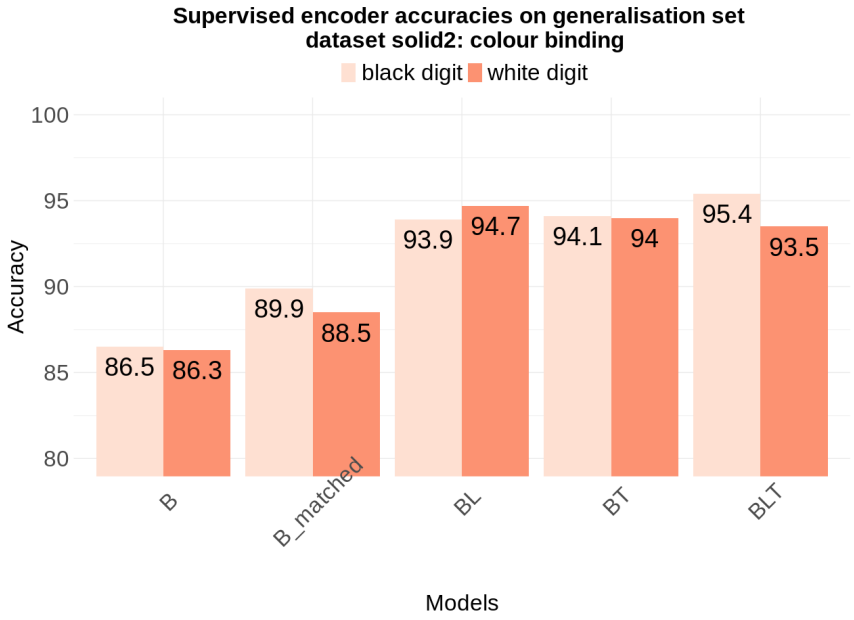

Figure 3.1: Encoders accuracies on generalisation set of solid2 dataset with colour binding

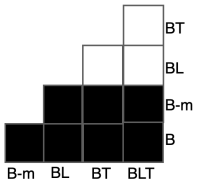

Figure 3.2: Significant differences solid2 encoders colour (McNemar test expected FDR = 0.05)

The second representation used to train and test the network was one where the digit identity and the depth were bound. With this set up, all the networks were able to accurately predict the front digit. However the accuracy on the back digit was much lower. Moreover, comparing the networks shows a decrease in the back digit accuracy between the normal B and B-matched, and from the BL, BT and the BLT. This trend follows the increasing number of parameters in these networks, suggesting that the larger models are over-fitting the training data and unable to perform as well on the test set.

On the one hand, comparing networks that are most matched for parameters (B and BL; B-matched and BLT) shows that recurrency is still helping. On the other hand, the difference in the average accuracies of both digits between the encoders trained with the colour-binding representation (3.1) and this representation shows how explicitly understanding depth is a much greater challenge for the encoders than learning to match parts of digits to a semantic identity.

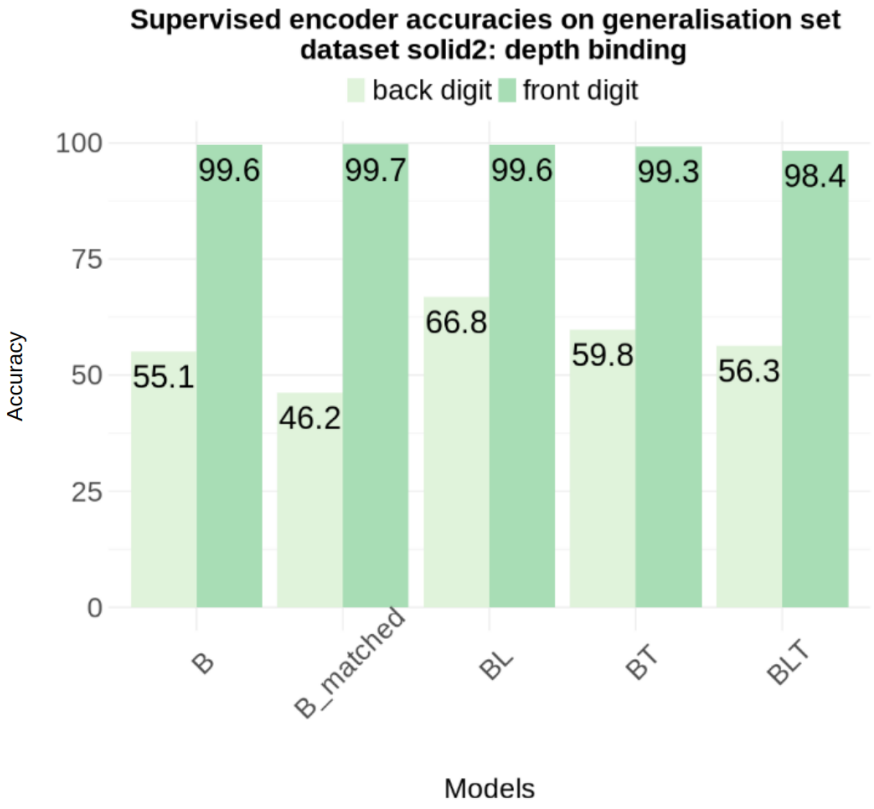

Figure 3.3: Encoders accuracies on generalisation set of solid2 dataset with depth binding

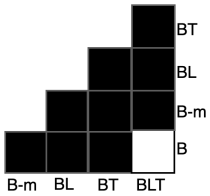

Figure 3.4: Significant differences solid2 encoders depth (McNemar test expected FDR = 0.05)

Encoders on the Border2 dataset

Next we tested the encoders on the border2 dataset. With this dataset, the encoders can no longer rely on a separation of objects by colour, as they each have the same mixture of black and white. Instead, the process of separating digits has to be based on their shapes, which can change depending on their location relative to the other digit.

One way in which recurrency is thought to help in these problems is by iteratively making a best guess about pixel segmentation, which is then used for digit identification, which is then used again to improve the pixel segmentation and so on. This is a similar iteration technique as the one found in the Expectation Maximisation algorithm and relies

on the idea of explaining away the occlusion in the image by having the best guess of the digit identities.

As shown in figure 3.5, as hypothesised above, all the networks can detect the front digit with accuracy. However, there is sudden jump in the accuracies of the back digit between the feedforward networks B and B-matched and the recurrent BL, BT and BLT, providing evidence for the role of recurrency in dealing with the ambiguity of inference.

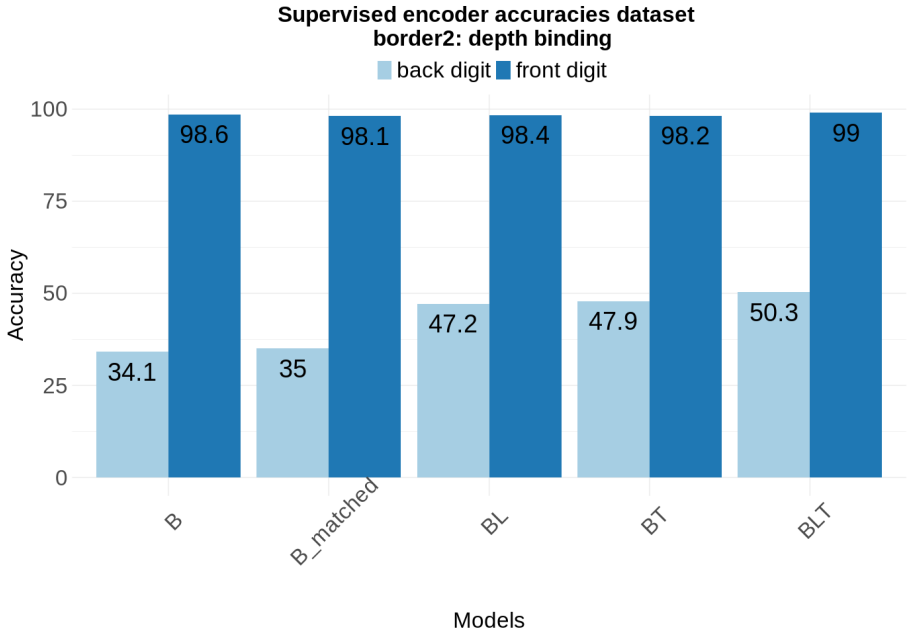

Figure 3.5: Encoders accuracies on generalisation set of border2 dataset with depth binding

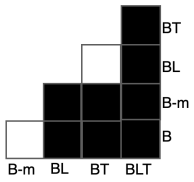

Figure 3.6: Significant differences border2 encoders (McNemar test expected FDR = 0.05)

Encoders on the Border3 dataset

To test how well the performance of these networks would generalise to images with more digits, and therefore more occlusions, we trained and tested them images with three digits (the border3 dataset).

The results in figure 3.7 show again that the front digit is nearly always correctly labelled across all encoders. The B encoder does the worst, and the BLT does the best, closely followed by the BL. The accuracy of the middle digit is also very similar between the feed forward and recurrent encoders once matched for parameters. The main difference between these networks is on the accuracy of the back digit, which rises between the and the B-matched due to greater parameters, and then rises again between B-matched and the recurrent networks, peaking at the BLT.

These results again suggest that recurrency plays a role in successfully performing object recognition, in this case especially on the objects that are likely to be occluded the most.

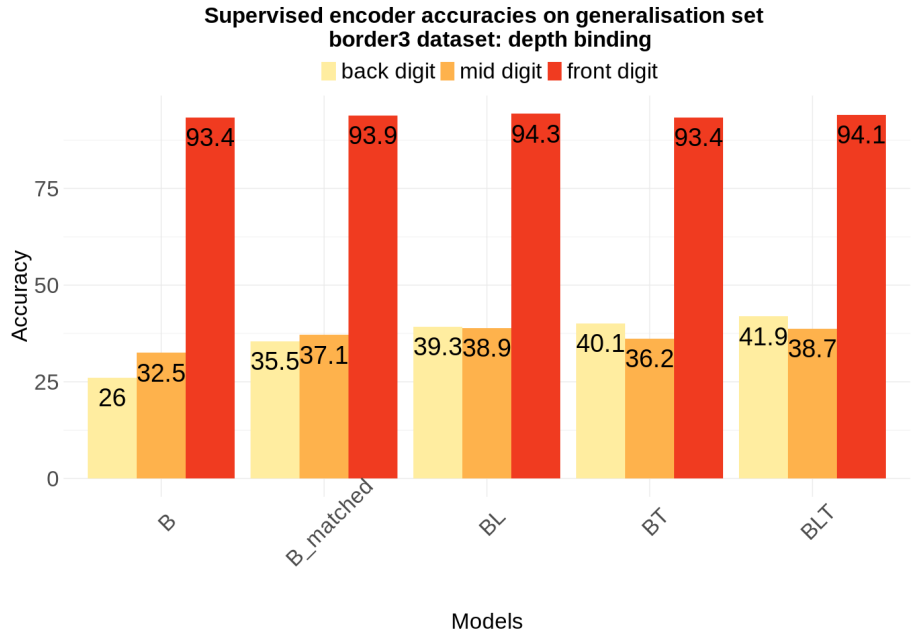

Figure 3.7: Encoders accuracies on generalisation set of border3 dataset with depth binding

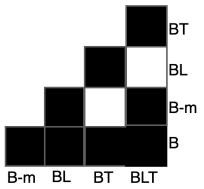

Figure 3.8: Significant differences border3 encoders (McNemar test expected FDR = 0.05)

Behaviour in revealing tasks

The digit revealing tasks are a way of interpreting the behaviour of the network under three situations of ambiguity (figure 3.9), created by moving one digit relative to another which is fixed.

The first ambiguity is when the digits are exactly in the same location, and only the front one shows. Being trained on two digits, the encoder have to give an answer about the back digit. The question is if it accurately expresses the uncertainty about the rear digit that an ideal observer for this task would have.

The second ambiguity occurs when the back digit is shifted from its location and partly visible. Depending on its shape and what is occluded we might specific best guesses of what it could be. The comparison here is to see if the encoders also give the same guesses.

Finally, the third ambiguity is in depth. The back digit keeps moving away from the front digit until there is no more occlusion. The networks again have only been trained on images with occluding digits, so they might not put all the answers on one and instead hedge their bets by sharing digit identity over the two depths, with the ideal observer equally sharing the probabilities.

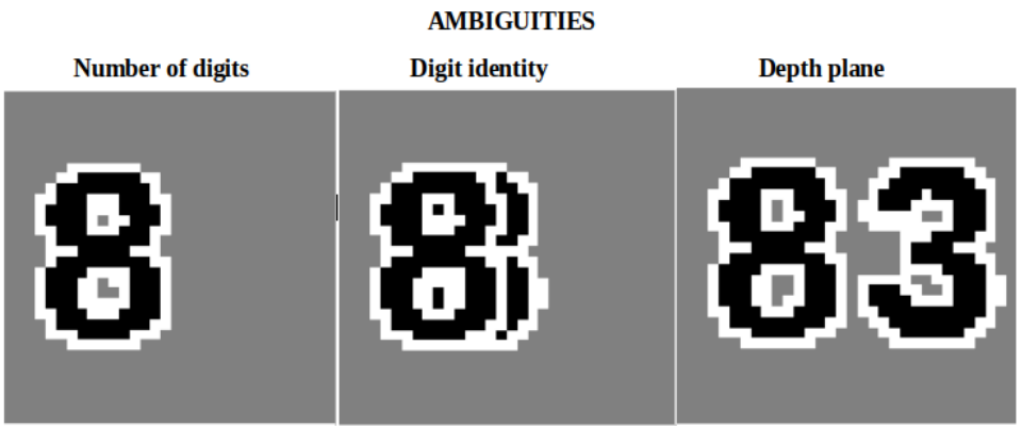

Figure 3.9: Ambiguities created by the digit reveal task

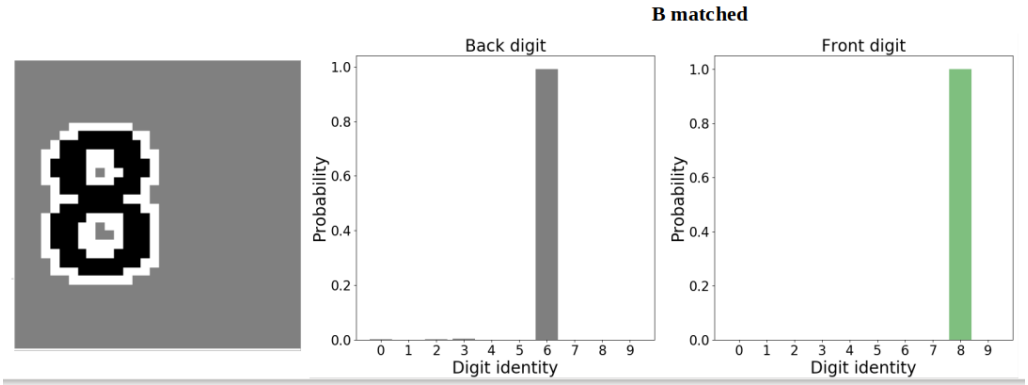

Figure 3.10: B-matched on number ambiguity

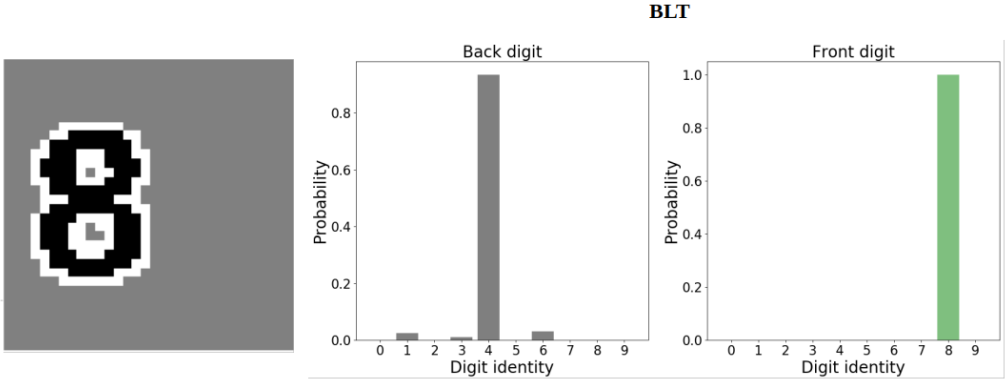

Figure 3.11: BLT matched on number ambiguity

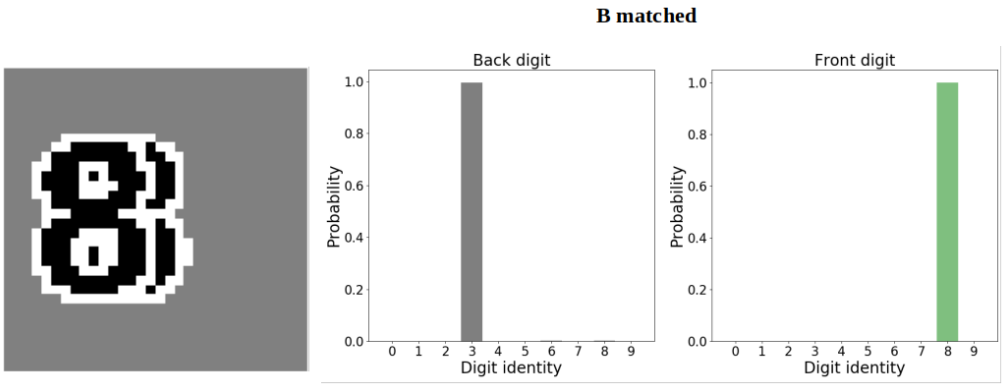

Figure 3.12: B-matched on digit identity ambiguity

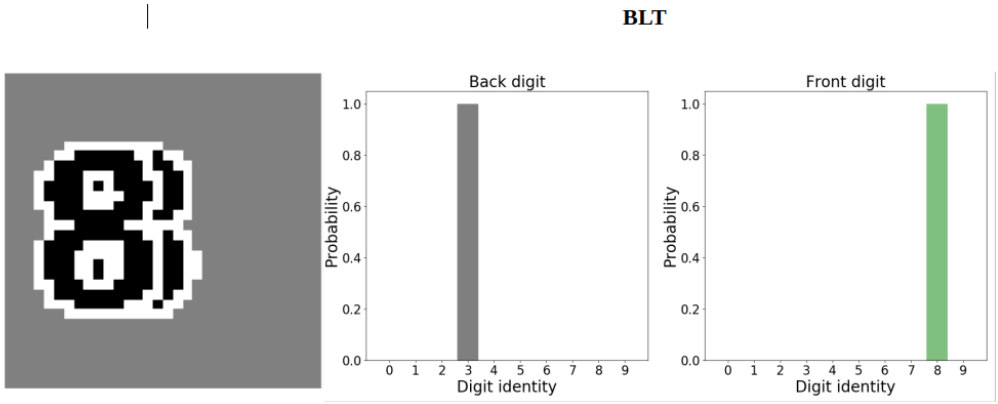

Figure 3.13: BLT on digit identity ambiguity

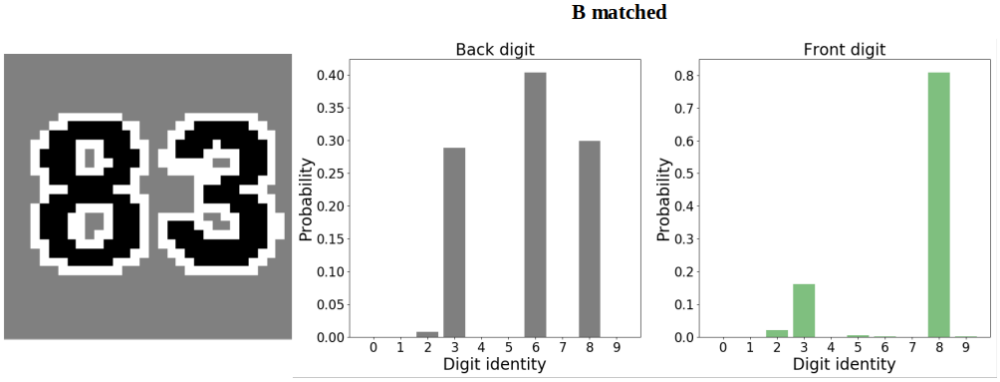

Figure 3.14: B-matched on depth plane ambiguity

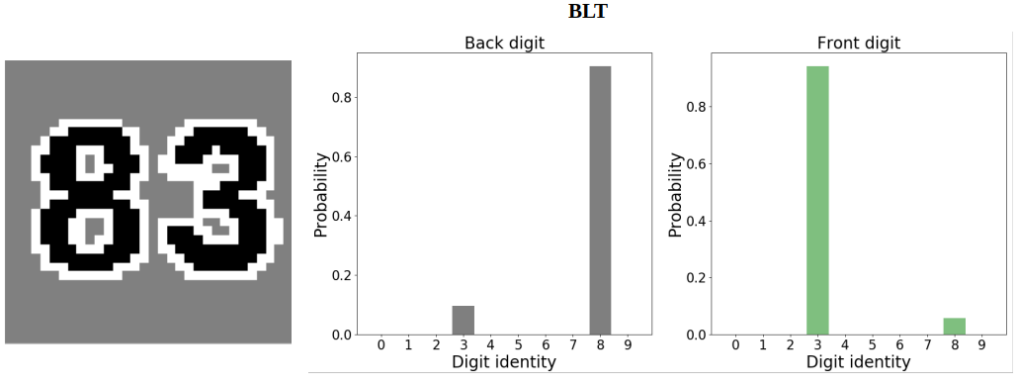

Figure 3.15: BLT on depth plane ambiguity

In the digit number ambiguity, both B-matched and the BLT encoders manage to get the identity of the front digit correct, but are very confident about the identity of the back digit, suggesting they have failed the optimal compared to the optimal observer. The BLT encoder, whilst overconfident, does show some ambiguity towards other digits.

In the digit identity ambiguity, both networks manage to get the back digit identity correct, despite the fact that it is still ambiguous between it being an eight or a three. The ideal observer therefore would weight these options as equally probable, unlike either of the networks.

Finally, in the depth plane ambiguity, the B-matched encoder decides to give the "back" digit an identity of six, despite also showing high probabilities for the digits three and eight. The BLT encoder, instead sticks to the correct digits, but does not assign share the probability of depth equally. It is interesting to note that as the data used to train the networks is exactly the same, and the B-matched and the BLT favour different digits to be in front, this is a product of the network architecture rather than a bias inherent to the dataset.

Graphics

In the graphics task we investigate the role of recurrent connections in the ability to generate images of occluding digits from a pre-set representation.

Decoders on solid2 dataset

To test the role of recurrency on a dataset where digits could be distinguished by both colour and shape, we trained and tested the 5 decoders (B, B-matched, BL, BT and BLT) on the solid2 dataset using the colour-bound representation.

A comparison of the mean reconstruction error, otherwise referred to as the reconstruction loss, across the training and the test sets for different models is shown in figure 3.16. This figure brings to light once again with the solid2 dataset, the tendency of higher parameter models to overfit. The loss on the training set decreases in a similar shape as previous results, but the test set it rises over the three recurrent networks and between the B and the B-matched.

Interestingly the best result is within the recurrent architecture, and is held by the BL decoder, the recurrent network with the least parameters.

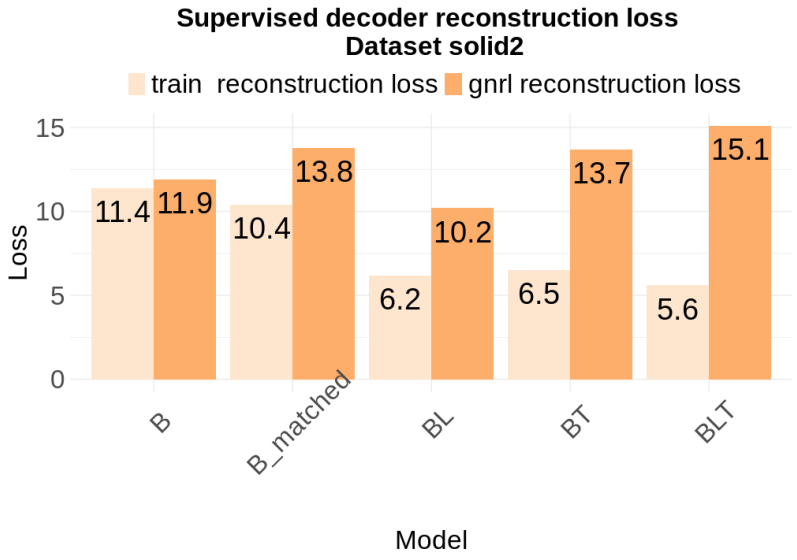

Figure 3.16: Reconstruction loss for training and generalisation of dataset solid2.

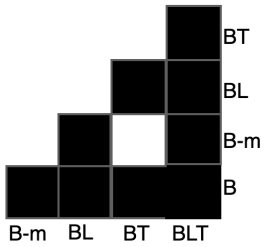

Figure 3.17: Paired T-test (FDR=0.005)

To put these reconstruction losses into context, we also need to look at the reconstructed images. Figure 3.18 shows an example of the reconstructions from the B, BL and the BLT networks.

From these images we can see how all three decoders actually succeed in this task, with the BLT actually recreating the most sharp of the images. The high reconstruction loss therefore may be a product of blatant mistakes of digit identity in the test set rather than of blurriness.

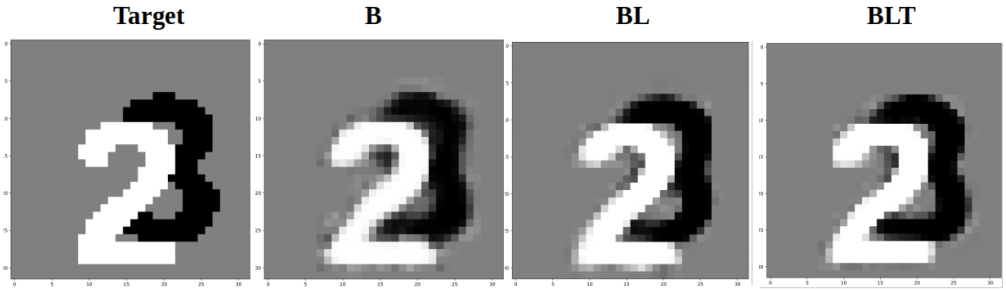

Figure 3.18: Example reconstructions for B, B-matched and BLT on solid2 dataset

Decoders on border2 dataset

To see whether removing the distinction of digits by colour, as in the solid2 dataset, has an effect on the role of recurrency in performing graphics with the decoders, we trained and tested the same five networks, B, B-matched, BL, BT and BLT on the border2 dataset.

First we looked at the comparative reconstruction losses on the training and the test set (figure 3.19). Here, the difference in the reconstruction loss between the feedforward networks and the recurrent networks is much larger than in the solid2 dataset. Moreover, the recurrent networks no longer seem to overfit as dramatically, and the B-matched network also performs better than the B.

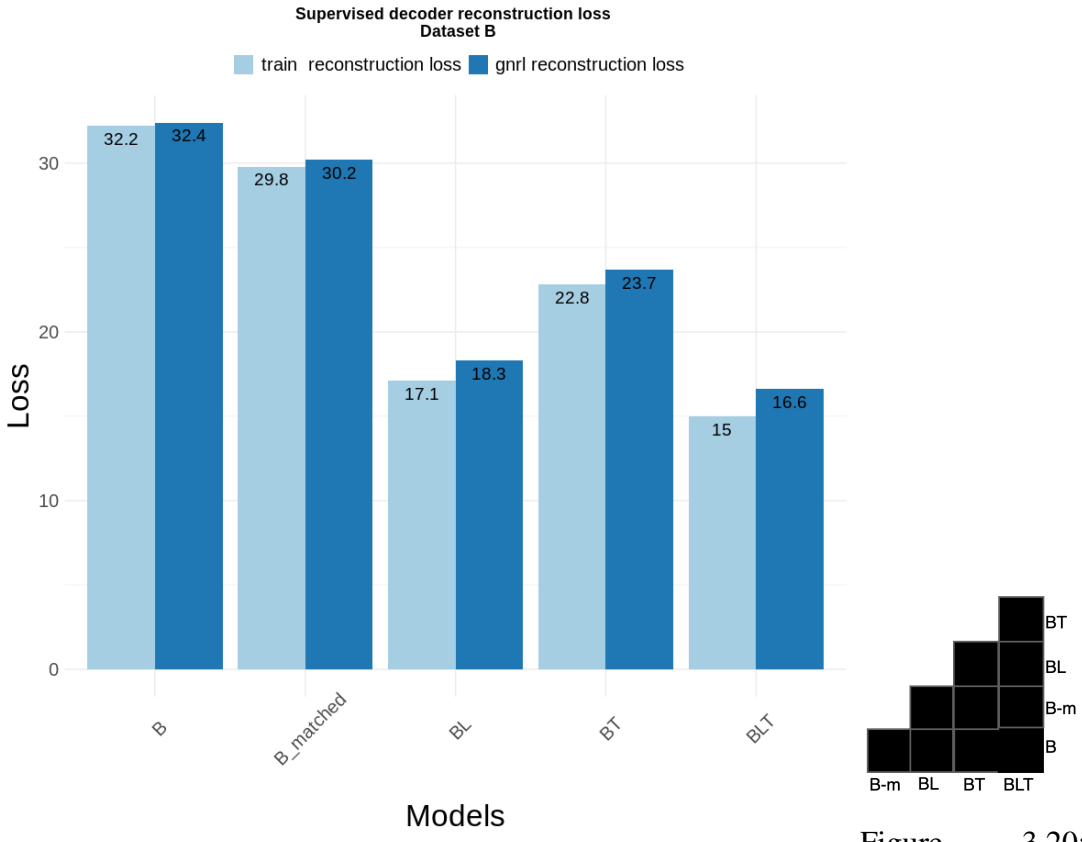

Figure 3.19: Reconstruction loss for training and generalisation of dataset border2.

Figure 3.20: Paired t-test (FDR=0.05)

To put these losses in context, we can compare the actual reconstructions. Figure 3.21 and 3.22, show two examples of the target, B-matched and BLT reconstructions on two test cases. As is evident, both the feedforward and recurrent architectures are able to draw the front digit well. However, the back digit in the B-matched is blurry when close to the front digit, whilst in the BLT the back digit is tightly drawn.

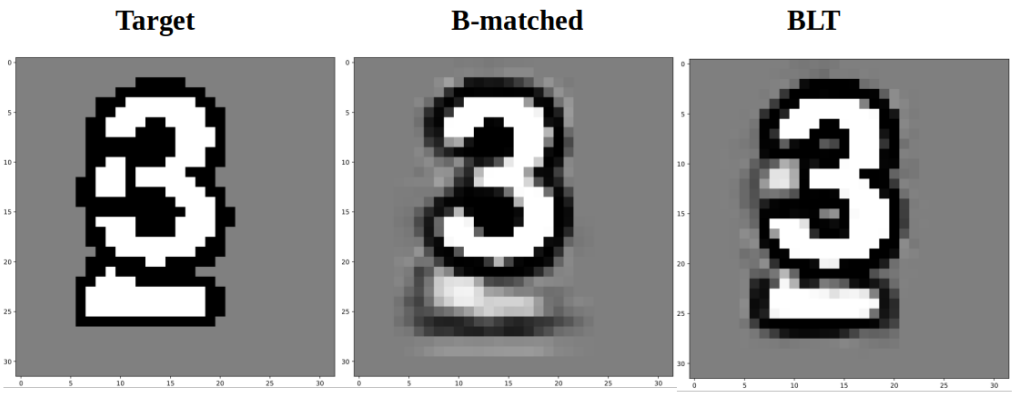

Figure 3.21: Target, B-matched and BLT graphics example 1

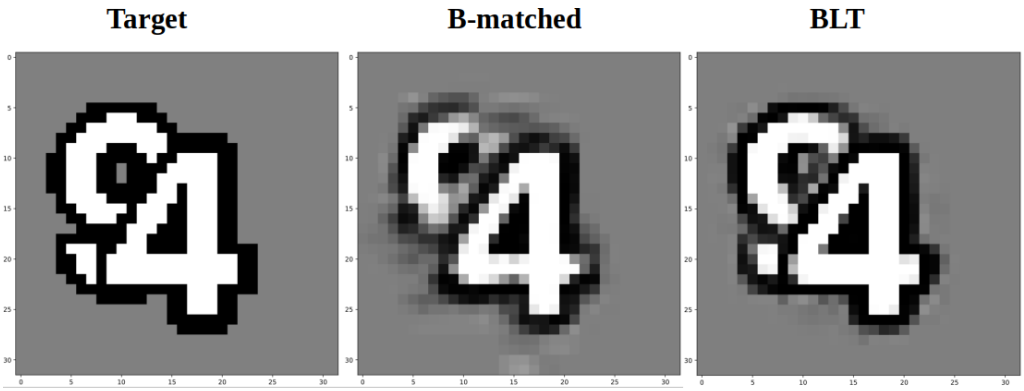

Figure 3.22: Target, B-matched and BLT graphics example 2

To build an intuition on how the recurrent networks are able to do this, we plotted the reconstructions of the BLT decoder at each of the four time-steps, as is shown in figure 3.23. We can see how in the time-step 0, when the BLT acts the same as a simple feedforward network, the result is very similar to the reconstruction of the B-matched in figure 3.22. But with recurrency, at each new time-step, the gap between the back and the front digits becomes smaller and less blurry, until the final correct image is created.

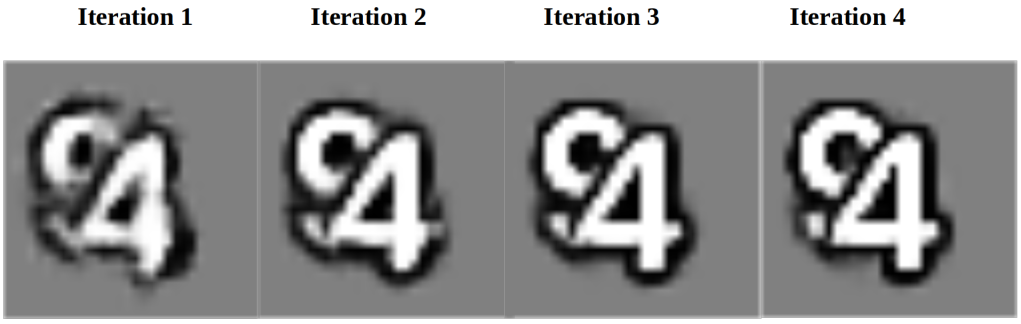

Figure 3.23: Output of BLT decoder at each iteration

These results suggest that when the digits can only be segmented by their shape, recurrent connections in graphics play a large role in refining the output image. Furthermore, recurrency in graphics allows the digits to start off draw with a gap in between them that is then minimised until this gap disappears.

Decoders on border3 dataset

Finally, we wanted to investigate how the role of recurrent connections is affected by a dataset with more digits. Therefore, we trained and tested the same five decoders, B, B-matched, BL, BT and BLT, on the border3 dataset.

With a three digits instead of two, the networks should experience greater difficulty in performing the graphics task. From the results shown with the border2 dataset, the role of recurrent connections should be maintained as the digits present the same problem of each having the same colour. Moreover, networks with greater parameters should also have an advantage compared to smaller networks, as they can be used to deal with the larger input.

A comparison of the training and testing reconstruction losses is shown in figure 3.24. The similar trend that has been shown so far was also repeated, with the lowest errors

shown in the BLT decoders, and the B-matched helping but not managing to reach the same level as the recurrent networks.

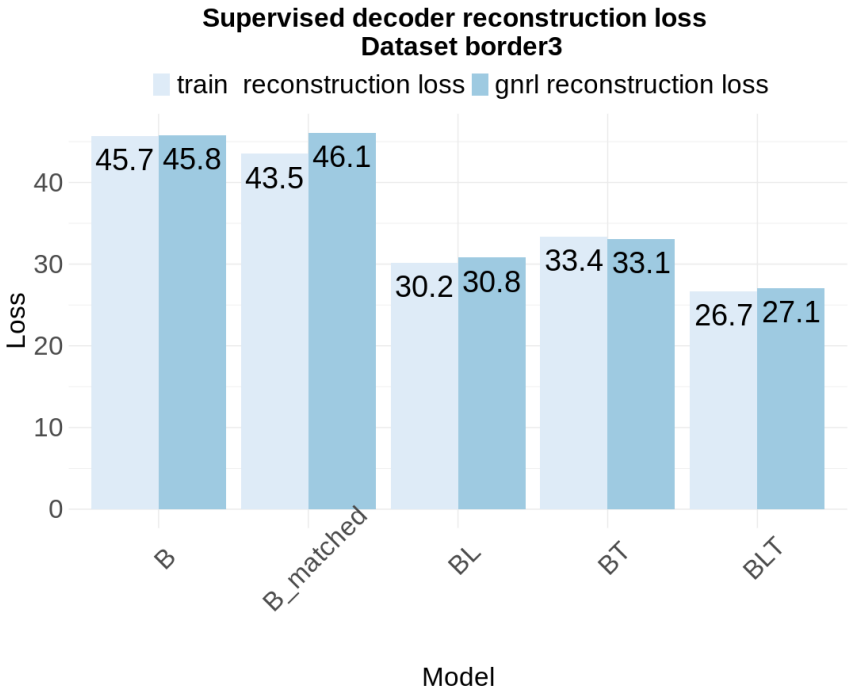

Figure 3.24: Reconstruction loss for training and generalisation of dataset border3.

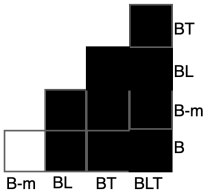

Figure 3.25: Paired t-test (FDR=0.05)

To make sense of these losses, we plotted some of the reconstructions form the test set. Figure 3.26 shows an example of these test reconstruction using the B, B-matched and BLT networks.

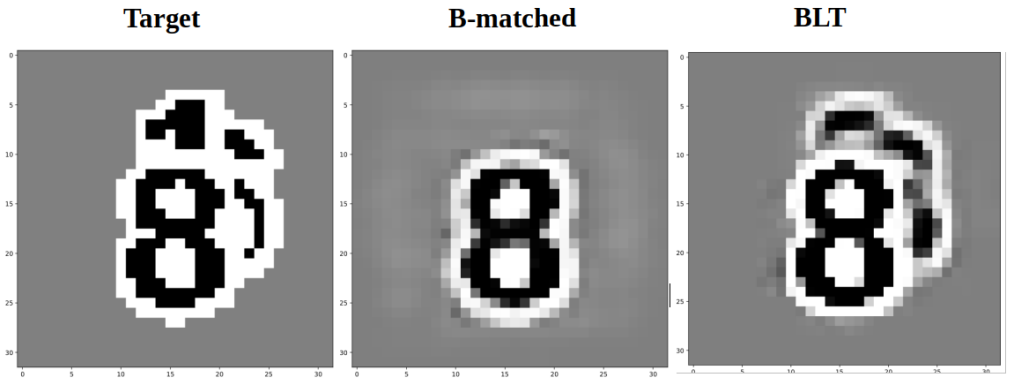

Figure 3.26: Target, B-matched and BLT graphics example from dataset border3.

Finally, to investigate how recurrent dynamics improve the performance on the graphics task, we plotted the reconstructions of the BLT network at each time-point. As is shown in figure 3.27, the recurrent connection seem to perform a similar task of closing the gap between the front digit and the back ones. In figure 3.27 however it also seems to to miss the identity of the middle digit. Figure 3.28 shows a second example from the test set, where instead the network manages to draw first the front digit, then the next largest middle digit and finally the smallest at the back.

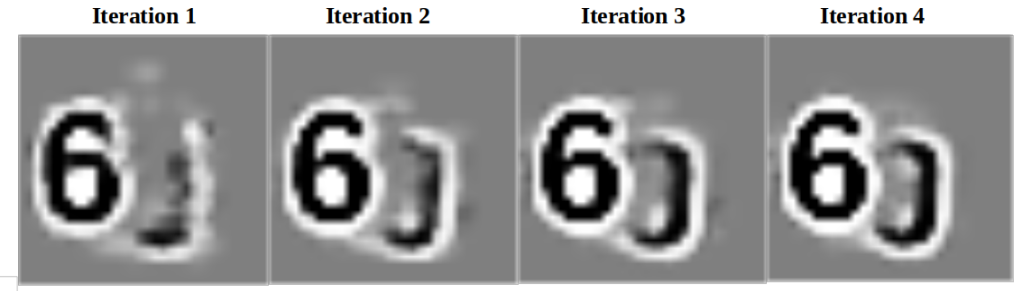

Figure 3.27: Plotting of image at each time point: BLT decoder on border3 example 1

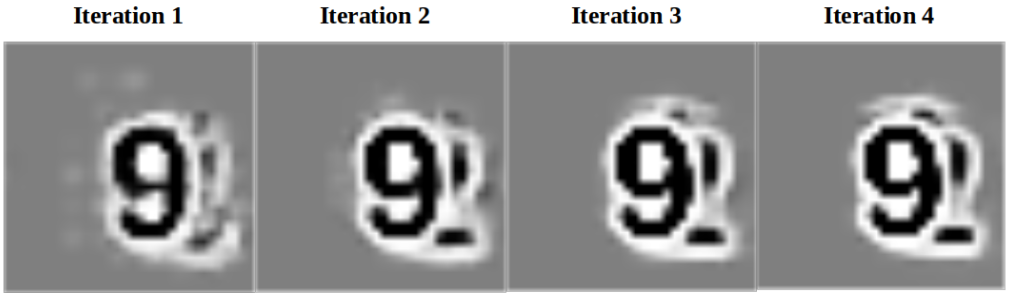

Figure 3.28: Plotting of image at each time point: BLT decoder on border3 example 2

Autoencoders

We have shown that recurrency is important for both inference and graphics. However, in both these cases we pre-set a representation that the networks had to use for each tasks. We therefore wanted to investigate whether, by combining the inference and the graphics tasks together into one, a network could learn an insightful way of representing depth and other properties of the input image for for further graphics.

To do this we then trained an autoencoder on a pretext task, where it received an image with occlusion, and had to reconstruct the same image but with the depth order of the objects reversed. Ideally to successfully solve this task the network would need to to create a useful representation of depth.

In light of the results on the inference and graphics tasks, we chose to build the autoencoder using the BLT architecture both in the encoder and the decoder. We then trained and tested this BLT-BLT autoencoder on the border2 and the border3 datasets.

Reconstruction performances on datasets

To test the ability of the autoencoder to perform the unsupervised task, we started by training it and testing it on the border2 dataset, as it the simplest case where digits have to be differentiated by shape, not colour.

We did not compare other architectures in this task, so to understand the performance of the network we plotted the inputs, targets and reconstructions of a subset of the test set. Three examples from this process are shown in figures 3.29. However, to build a stronger sense that the network was able to perform successfully across the test set, we compared the final reconstruction loss of the autoencoder with the best decoders. This

showed that the autoencoder actually performed with a lower reconstruction loss than the best decoders in the border2 dataset (16.6 for BLT decoder vs 14.7 for autoencoder), but with a slightly higher reconstruction loss than the best decoder on the border3 dataset (27.1 for BLT decoder vs 32.4 for autoencoder). From these results, and from the samples shown below, we can say that the autoencoder solved the task with the border2 dataset successfully, and was also able to perform well on the border3 dataset.

To build an understanding about what the network learned, I have also plotted two examples (figures 3.30 and 3.31) where the network gave interesting reconstructions to its inputs. In these examples the identity of the back digit is ambiguous. In figure 3.30 it could be either a six or a zero. The reconstruction shows that the network was believed the six was more probable.

In figure 3.31 the back digit could be either an eight or a three. In this case, the reconstruction is blurry exactly at the point which would distinguish these two numbers, suggesting that the network has an awareness of the conflicting possibilities and was able to maintain this ambiguity all the way to the output.

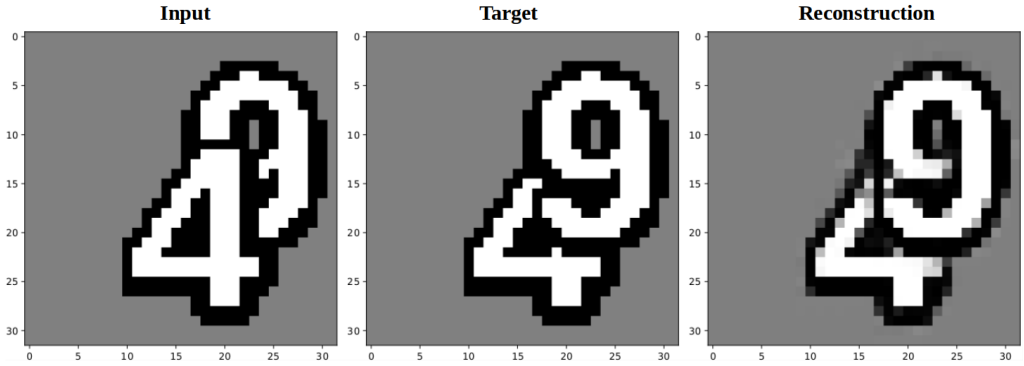

Figure 3.29: BLT-BLT autoencoder on dataset border2: a normal example

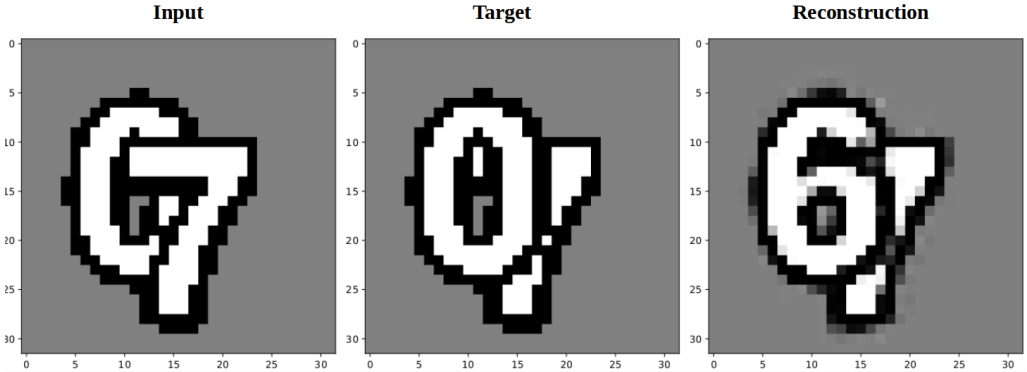

Figure 3.30: BLT-BLT Autoencoder on dataset border2: ambiguity resolved

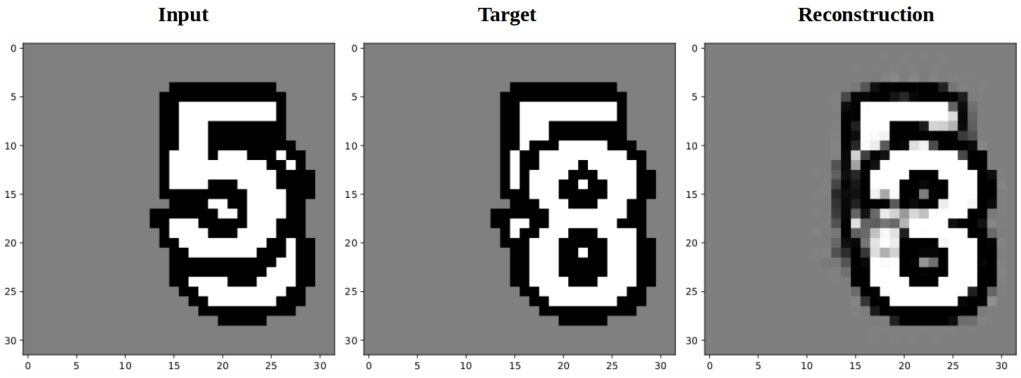

Figure 3.31: BLT-BLT Autoencoder on dataset border2: uncertainty maintained

Given the success of the unsupervised task using the border2 dataset, we trained the same BLT-BLT autoencoder on the border3 dataset,to see if the increased number of digits would lead it to fail or not.

Again, for a subset of the test set we plotted the reconstructions. Three examples of these are shown in figures 3.32, 3.33 and 3.34.

As these figures show, the autoencoder is able to correctly invert the relative depths of the digits, and when it can, also draw the correct identity. In figure 3.33 we see for example how the digit at the back is nearly completely occluded. In the reconstruction, the autoencoder clearly draws the nine and the three, but in between where the back digit should be, it leaves a blur, again representing its ability to correctly represent ambiguity.

Interestingly, there are also examples where the digit identity is not ambiguous and yet the reconstruction fails (figure 3.34). In this example, the vertical horizontal cross of the four is not enough to identify it even though there are no other digits that have this specific pattern.

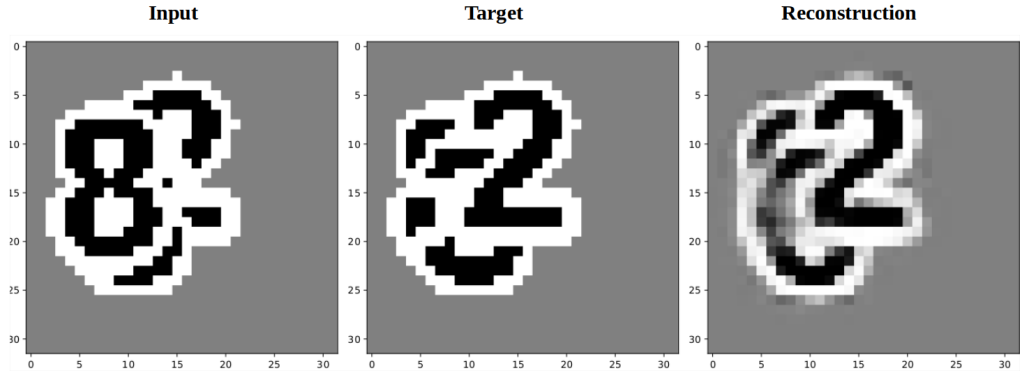

Figure 3.32: BLT-BLT autoencoder on dataset border3: a normal example

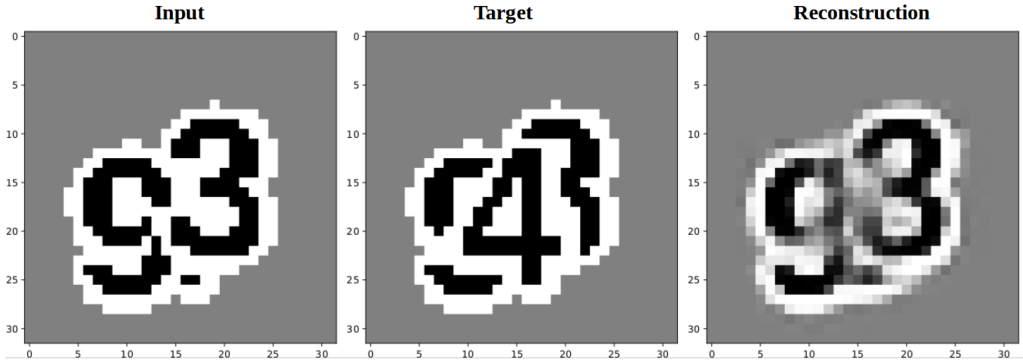

Figure 3.33: BLT-BLT autoencoder on dataset border3: dealing with nearly complete occlusion

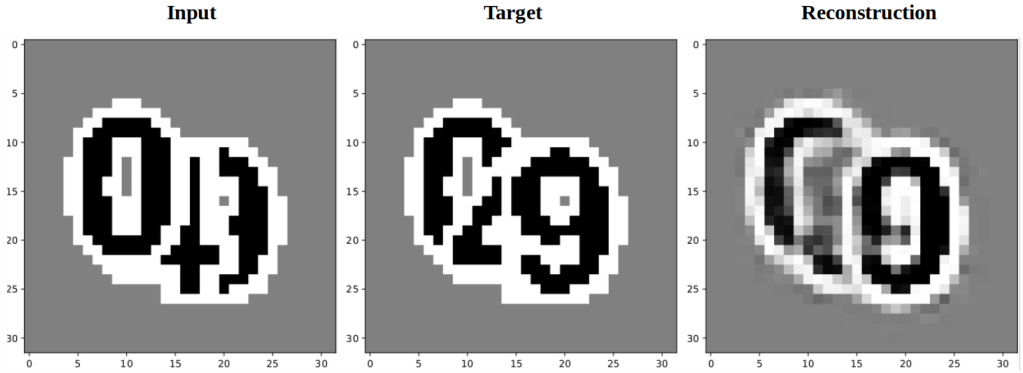

Figure 3.34: BLT-BLT autoencoder on dataset border3: problems with reconstructions

To note: the reader may have noticed that the border2 dataset for the autoencoder had digits with white in the middle and black as the border. Running the network on this dataset was accidental. However, it worth noting that other than this switch in colour, the dataset was created in exactly the same was as the original border2, and was self-consistent with its images. For comparisons with the decoder, the autoencoder was run again on the correct dataset, so they also hold.

Effects of code size

For correct reconstruction the bottleneck code needs to encoder a compressed representation of the image appropriate for the task. An interesting question is at what stage of decreasing the code size does the reconstruction start to degrade. Theoretically there is no limitation to the the values of the weights and the activations in the code layer, as these are both continuous variables. It may happen that as the code size decreases the numerical range of the activations increases as a way to try to maintain its capacity.

To test this we repeated the unsupervised task changing only size of the autoencoder bottleneck, and collected the reconstruction losses. A bar plot of the reconstruction losses is shown in figure 3.35 for border2 dataset and in figure 3.36 for the border3 dataset. The range of activations was collected by visual inspection of the maximum and the minimum values of the code during runs over the test set.

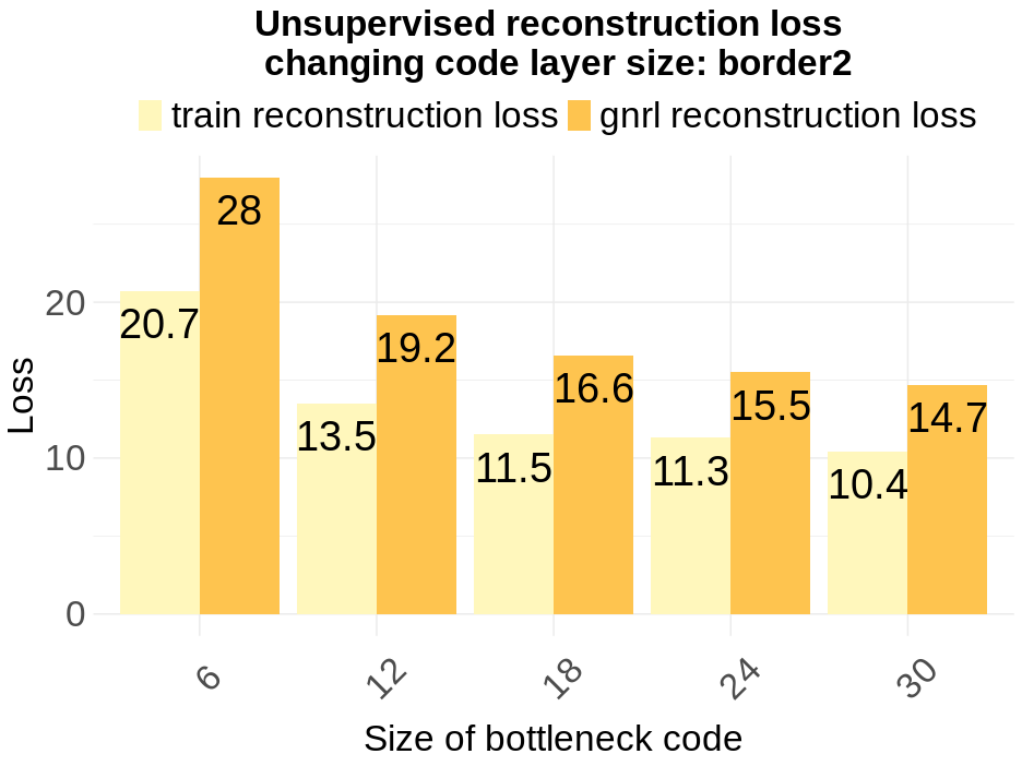

Figure 3.35: Loss over decreasing and increasing size of bottleneck code with border2 dataset

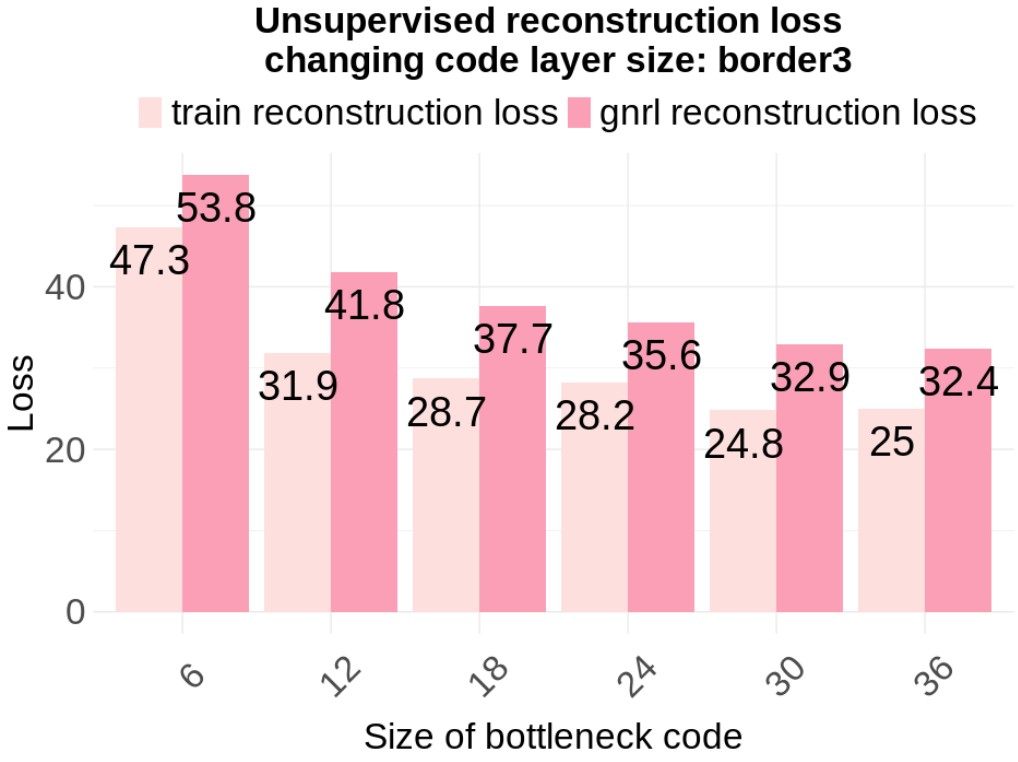

Figure 3.36: Loss over decreasing and increasing size of bottleneck code with border3 dataset

As is shown by figures 3.35 and 3.36, the reconstruction loss decreases asymptotically as the code layer size increases. Interestingly, the loss really starts to flatten out around the same dimensions that we used in the inference and graphics tasks: 24 for the border2

dataset and 36 for the border3 dataset. Interestingly, the activations actually increased in range as the code size also increased. This went against the theoretical hypothesis, suggesting that in practice, when there are so few units, it becomes difficult to break up the continuous number line into areas of semantic meaning.

Read out from code to code

Having shown that the autoencoder network is able to perform well in the unsupervised task both with the border2 and the border3 dataset, we wanted to investigate and interpret the representation that was learned.

We first asked if the representations learned by the autoencoder were linearly related to the one we set for the inference task. To do this, we attempted to train a fully connected layer without an activation function that would have as input the representations built by the autoencoder and as target our preset representations for the same images. If these representations were linearly related, a simple linear decoder would be able to predict our representation from the learned one, giving us a platform to start to interpret the roles of different units in the bottleneck code.

Figures 3.37 and 3.38 show the loss and the digit accuracy over training. As as shown, the accuracies never rise to more than 15% and the digit identity losses also plateau. The location loss, however, decreases much more drastically. This may show that location is linearly decodeable, however, further work is required to ascertain this.

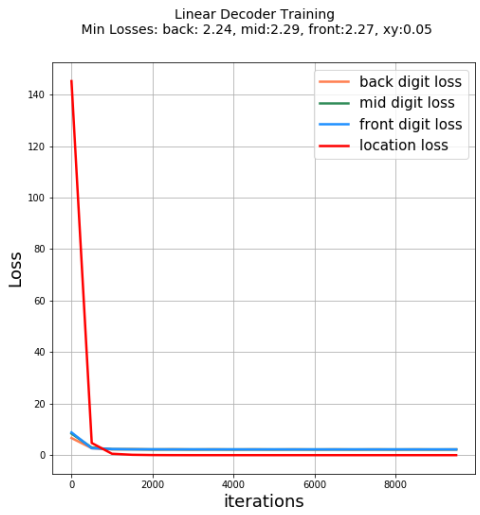

Figure 3.37: Loss of linear decoder during training on dataset border3

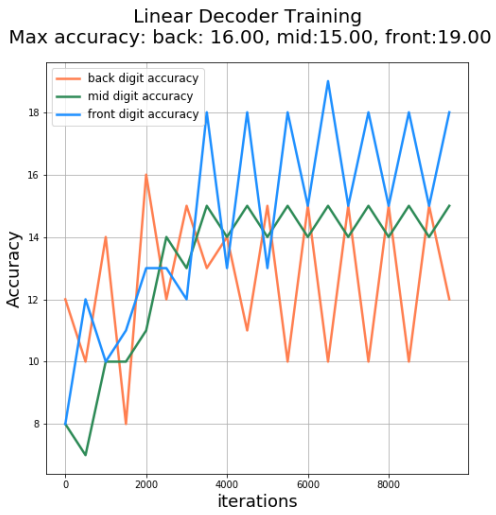

Figure 3.38: Accuracy of linear decoder during training on dataset border3

Chapter 4

Discussion

Summary of results

In the inference task, we found support for the hypothesis that recurrent connections help when recognising objects under occlusion. This benefit was especially strong when digits could not be separated by colour, but only by shape, such as in the border2 and border3 datasets, as it created greater ambiguity and need for refinement of inference hypotheses.

In the graphics task, we also found evidence that recurrent connections aid in performance. More specifically, with dataset border2 and border3, we found the recurrent process to aid by gradually filling in the gap between the front digit and the ones being occluded.

In the unsupervised task, we saw how building an autoencoder with recurrency in both the encoder and the decoder allowed it to successfully invert the depth of the digits of the input image. Increasing the size of the bottleneck layer led to asymptotically decreasing improvements of the reconstruction loss, suggesting that an optimal code size similar to the one used in the inference and graphics tasks.

Training a linear readout showed that the representation that emerged did not map digit identity linearly to the depth-bound digit identity representation that was used in the inference and graphics tasks, but was able to map digit locations. Further investigation of the learned representation showed that depth was not encoded by any single unit, and traversals through the units in the bottleneck code showed that digit identity was both distributed across units and within each unit as well.

Caveats

The performance of ANNs on a task is dependent on the random initialisation of the weights. To check that this variation was not a cause for concern in the comparisons, we ran some of the models multiple times on different random seeds. The differences shown were within one percent difference in final accuracies and similar in the final loss values.

A fair comparison of ANNs is difficult to undertake. On the one hand, each network should be optimised for the task. On the other, the networks should not be altered too much as to lose the commonalities that allow a meaningful comparison to be made. Optimisation can be achieved by architectural changes and changes to the training regimes. To attempt to find this balance, we did not make architectural changes other than those specified in

the methods: adding feedback connections for the BL, BT, and BLT and increasing the kernel size and the number of filters for the B-matched.

The main three main hyperparameters for the training regimes for these models were the learning rate, the L2 loss coefficient, and the number of epochs used for training. In this project, we only attempted to optimise the learning rate, whilst keeping the loss coefficient the same, and the number of epochs for training also equal in all networks.

When initially training the models, a variety of learning rates were used, until two values were chosen due to their ability to create good performances across different models. Each model was then run with these two learning rates, and the network with the best performance on the training set was used for comparison of the test sets.

For stronger evidence, a more principled comparison of learning rates, L2 loss coefficients, and training epochs should be performed. This could be done by creating a cross-validation set to optimise the hyperparameters for each network on each different dataset.

Related work

This project drew considerable inspiration from the work of Spoerer, McClure, and Kriegeskorte 2017, who were among the first to do a thorough investigation of recurrency for images of fully drawn and partly drawn digits with occlusion.

This thesis builds on and strengthens this previous work in a few ways. Firstly, Spoerer, McClure, and Kriegeskorte 2017 built datasets with three, four and five digits occluding all with the same colour pattern of black centre with a white edge. We instead built datasets with two and three digits per image. Therefore, to build a test set that showed real ability to learn and generalise to unseen data, we set up specific combinations of digits that were never shown in the training set and instead were only presented in the test set. This step was not done by Spoerer, McClure, and Kriegeskorte 2017 and was especially crucial to our datasets, as with fewer number digits per image, they would be more amenable to memorisation-based solutions. This becomes less likely as the number of possible combinations of digits becomes larger much larger than the training set size.

Secondly, in Spoerer, McClure, and Kriegeskorte 2017 the task of the networks were to output a vector of size ten, with indexes that matched with the digits zero to nine, representing the relative certainty that specific digits were present in the image. From this vector, accuracy was calculated by whether the index of the highest three values of the output matched the digit identities in the image. In this project, the target representation was very different, such that the network not only had to identify the correct digit but also route it to the correct part of the output vector to describe its relative depth. By training the networks to represent digit identity this way, we explicitly investigate its ability to understand depth.

This difference in task was also matched by the difference in architecture used to solve the task. In the Spoerer, McClure, and Kriegeskorte 2017 paper, after two recurrent convolutional layers, they apply a global max pooling to each filter and use these values as inputs for a fully connected layer linked to the output. This process of max-pooling is useful for object detection on its own, but is an irreversible loss of information that is crucial for the depth-inverting autoencoder. Because of this, and the need to lower dimensionality, in this project we built the networks with four recurrent convolutional layers instead of two, leading to a greater number of parameters, and a greater capacity for the three tasks.

The most closely related work to the graphics task and the autoencoder task is the General Query Network (Eslami et al. 2018). Given two scenes and the point of view from which those scenes were taken from in a full 3D virtual world, they ask the network to recreate a third scene from an arbitrary point of view. To solve the task, the network has to build an understanding of objects and how occlusion can be explained by their relation in 3D space. Whilst the depth inverting autoencoder is a similar task to this, Eslami et al. 2018 do not explicitly investigate the role of recurrency in the inference and graphics parts of the task.

Unanswered questions and future work

Future work regarding this project can be broken down into three parts. The first is to make sure that the networks are doing what we think they are doing, and that the results are indeed correct. The second, if the first is true, is to investigate exactly how recurrency might be aiding performance in the inference and graphics tasks, as well as investigating the representations that emerge from the unsupervised task. Lastly, having understood these computations, one could build more targeted architectures that include the computations thought to be created by the recurrent connections.

To ascertain the quality of results, the networks retrained under a more systematic hyperparameter optimisation regime. This might give answers to some of the results that we see above, such as the overfitting of the recurrent encoders and decoders on the solid2 dataset. Secondly, the tasks should be made harder by increasing the number of varying properties of the digits. For example, at the moment, each digit has a fixed shape and size, which is good to simplify the task but also might lend itself to trivial solutions. One could also vary the digit colours, adding RGB channels and again forcing the network to understand increasingly abstract qualities of objects. Finally, the networks could be trained on a varying number of digits in the image, and also tested on a varying number, maybe outside the range seen in the training set. A true understanding of the relation of depth and occlusion would allow generalisation across many different datasets. All of these experiments could be applied to the encoders, the decoders, and the autoencoders. With the autoencoders, it would then be interesting to investigate how the increasingly difficult tasks affect the representations that emerge.

Several methods could be used to build an understanding of the emergent representations of autoencoders. One could perform a linear traversal of each unit in the code, whilst keeping the others fixed to build an intuitive understanding of what specific units in the code layer represent. Another way would be to give the networks a series of images some with depth-inversion and some without and to visualise the clustering of representations using Multidimensional Scaling, which looks at the representational similarity between different encodings.

Secondly, a series of experiments to understand exactly what computations emerge when feedback connections are added should be set up. This investigation could be done initially by inspection of the output, as is shown in this thesis, and specific units at different time points. Another way would be to hand-engineer models that implement a specific type of recurrent computations and see if these models can also perform well on tasks with occlusion. Capsule networks (G. E. Hinton, Sabour, and Frosst 2018) are a recent example of neural networks that perform recurrent processing analogous to the Expectation Maximisation algorithm for visual perception.

The final aim of these steps would be to derive a network architecture that is biologically plausible, parameter efficient, robust to different inputs and from which we would be able to build predictions that can be tested against behavioural and neuroimaging data to give further insight into how the visual perception is achieved by the brain.

Bibliography

DiCarlo, James J, Davide Zoccolan, and Nicole C Rust (2012). "How does the brain solve visual object recognition?" en. In: *Neuron* 73.3, pp. 415–434.

Dietterich, T G (1998). "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms". en. In: *Neural Comput.* 10.7, pp. 1895–1923.

Durkalski, Valerie L et al. (2003). "Analysis of clustered matched-pair data". en. In: *Stat. Med.* 22.15, pp. 2417–2428.

Eslami, S M Ali et al. (2018). "Neural scene representation and rendering". en. In: *Science* 360.6394, pp. 1204–1210.

Hinton, Geoffrey (2013). *Taking inverse graphics seriously*.

Hinton, Geoffrey E, Sara Sabour, and Nicholas Frosst (2018). "Matrix capsules with EM routing".

Kriegeskorte, Nikolaus and Pamela K Douglas (2018). "Cognitive computational neuroscience". en. In: *Nat. Neurosci.* 21.9, pp. 1148–1160.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F Pereira et al. Curran Associates, Inc., pp. 1097–1105.

Liang, Ming and Xiaolin Hu (n.d.). "Recurrent Convolutional Neural Network for Object Recognition". In:

Liao, Qianli and Tomaso Poggio (2016). "Bridging the Gaps Between Residual Learning, Recurrent Neural Networks and Visual Cortex". In: arXiv: `1604.03640 [cs.LG]`.

O'Reilly, Randall C et al. (2013). "Recurrent Processing during Object Recognition". en. In: *Front. Psychol.* 4, p. 124.

Paszke, Adam et al. (2017). "Automatic differentiation in PyTorch". In:

Rajaei, Karim et al. (2019). "Beyond core object recognition: Recurrent processes account for object recognition under occlusion". en. In: *PLoS Comput. Biol.* 15.5, e1007001.

Rao, R P and D H Ballard (1999). "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects". en. In: *Nat. Neurosci.* 2.1, pp. 79–87.

Spoerer, Courtney J, Patrick McClure, and Nikolaus Kriegeskorte (2017). "Recurrent Convolutional Neural Networks: A Better Model of Biological Object Recognition". en. In: *Front. Psychol.* 8, p. 1551.

Tang, Hanlin et al. (2014). "Spatiotemporal dynamics underlying object completion in human ventral visual cortex". en. In: *Neuron* 83.3, pp. 736–748.

Appendix A

Appendix

Code

The code for creating the dataset was adapted from Spoerer, McClure, and Kriegeskorte 2017. The networks were all built using PyTorch (Paszke et al. 2017). The code is freely available on Github at `https://github.com/rickconci/OcclusionInference`.