Depth Inverting Autoencoders for Images under Occlusion

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
BITS F421T Thesis

By

Aiswarya Subramanian
ID No. 2015B4A70585G

Under the supervision of:

Prof. Nikolaus Kriegeskorte
&
Prof. Basabdatta Sen Bhattacharya

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS
December 2019

Declaration of Authorship

I, Aiswarya SUBRAMANIAN, declare that this Undergraduate Thesis titled, 'Depth Inverting Autoencoders for Images under Occlusion' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Certificate

This is to certify that the thesis entitled, "*Depth Inverting Autoencoders for Images under Occlusion*" and submitted by Aiswarya Subramanian ID No. 2015B4A70585G in partial fulfillment of the requirements of BITS F421T Thesis embodies the work done by her under my supervision.

Supervisor	*Co-Supervisor*
Prof. Nikolaus Kriegeskorte	Prof. Basabdatta Sen Bhattacharya
Professor,	Associate Professor,
Columbia University	BITS-Pilani Goa Campus
Date:	Date:

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS

Abstract

Bachelor of Engineering (Hons.)

Depth Inverting Autoencoders for Images under Occlusion

by Aiswarya SUBRAMANIAN

Brain is one of the most fascinating machines humans can ever find. It has a great ability to see and interpret the world around, by what we call 'visual perception'. It is known that the brain has recurrent connections in the visual cortex which aids it in perception. So we hypothesize that, adding feedback and lateral connections to the existing artificial neural networks (ANNs) can improve the performance of existing models in object recognition. We implement an autoencoder architecture, with recurrent connections and train and test the model on a variety of data sets to understand the role recurrent connections in visual perception. . .

Acknowledgements

I am really grateful to my mentor, Ruben van Bergen, who has always been with me throughout this project. His ideas and suggestions has been the most valuable to complete this thesis. I am thankful to Prof. Nikolaus Kriegeskorte for giving me this wonderful opportunity to work in his lab at Zuckerman Institute, Columbia University. I also thank my on campus supervisor, Prof. Basabdatta Sen Bhattacharya for her constant support. Lastly, I thank my university, BITS Pilani, Goa campus, for giving me a chance to work at one of the best places anyone can ask for. . . .

Contents

List of Figures

I dedicate this work to everyone who has supported me throughout

Chapter 1

Introduction

Visual displays such as art and illustration benefit from concise presentation of information. The difficulty of abstraction lies in selecting what is important. Our capability to see things around and understand them in a matter of seconds is a result of encoding object information that is robust to transformations in scale and position [1].

1.1 The Art of Seeing: Graphics and Inference Tasks

Everyday, we see thousands of objects and we recognise them within hundreds of milliseconds.This ability of our brain to make sense of what we see is known as visual perception. Visual perception in brain happens via two tasks namely graphics and inference. Inference is inferring the latent causes that best explain the image where as graphics is generating the image from the latent representation.

When an image of some object falls on our retina, it carries out a transformation on the input and each subsequent stage (V1, V2 and so on) in the visual pathway carries out its own transformation on the previous stage as input. So, these transformations converts the input into an abstract representation all the way up the optical pathway which finally reaches the brain. It is already known that inference happens in the brain but we are not sure how graphics plays its role. We presume that graphics or generative modeling may help with inference. Graphics may be used at each stage to check whether its inferences are consistent with the input that is, whether the higher level representation accurately explains the lower level representations.

So, as already mentioned, inference task is the conversion of image into the "abstract high level representation" and graphics task is the conversion of the high level representation back to the low level representations. Visual perception is in a way an inference task as the it tries to find the representation that best explains the input. And if we start with this representation, we get back the image, which is the graphics task. Hence, we may term visual perception as ”inverse of the graphics task”. [2]

1.2 Artificial Neural Networks

Human visual system is one of the wonders of the world. We perceive the world through our eyes which feeds the visual information to the millions of neurons in the brain where the visual information is processed. Though visual perception seems so easy to us, when we sit down and write an algorithm for a computer to recognize objects, we realise how difficult a problem it is, that our brain does with such ease.

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the biological nervous systems, such as the human brain's information processing mechanism. The key element of this paradigm is the novel structure of the information processing system. There are neurons in brain, that have many dendrites, which connects to other neurons and an axon where computations take place. Connections between neurons are called synapses. These synapses can be strong or weak depending upon how important the connection is. These synapses are characterised by **synaptic plasticity**, which is the ability of the synapses to change its weights, thereby adapting to newer stimuli.

Similarly in ANNs, there are computational units, called ”artificial neurons” (as shown in Figure 1.1) which are connected to other neurons by weight matrices, the values of which depend on how strong or weak the connections are. The network is allowed to see many training examples (stimuli) and the weight matrices are adapted according to whatever is newly seen. This mimics synaptic plasticity in brain.

What sets ANNs apart from other computer vision / AI algorithms is that information is processed in a distributed fashion: each ”neuron” performs a simple local operation to part of the data, and all these local computations together perform the task. These large number of highly interconnected processing elements working in unison to solve specific problems. They are computational models of the brain which perform very well in object detection and recognition tasks.
The architecture of ANNs is an attempt to mimic brain networks, as shown in Figure 1.2.

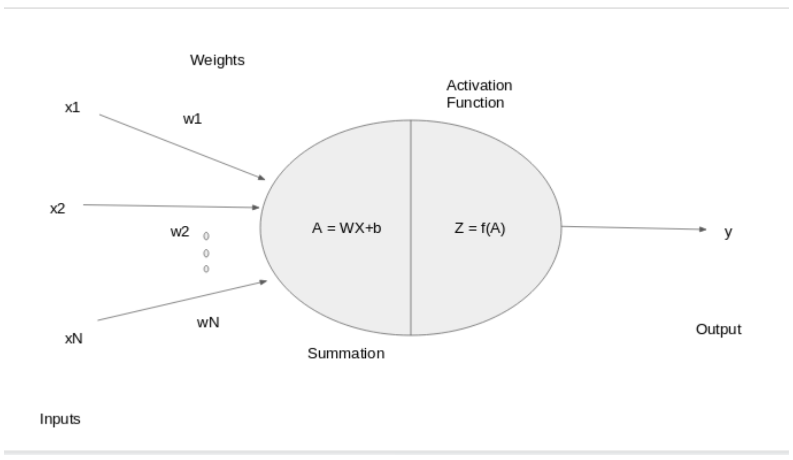

Figure 1.1: A typical Neuron

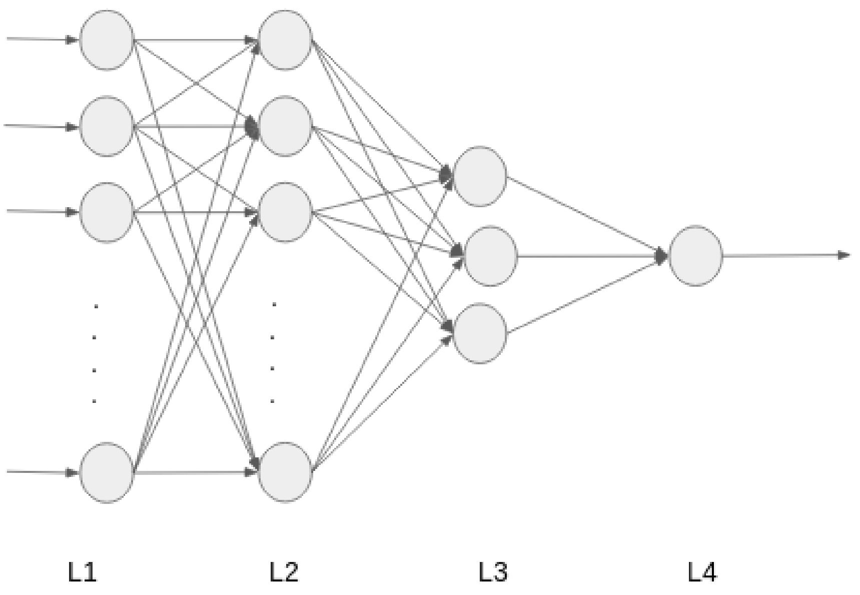

Figure 1.2: A typical ANN network

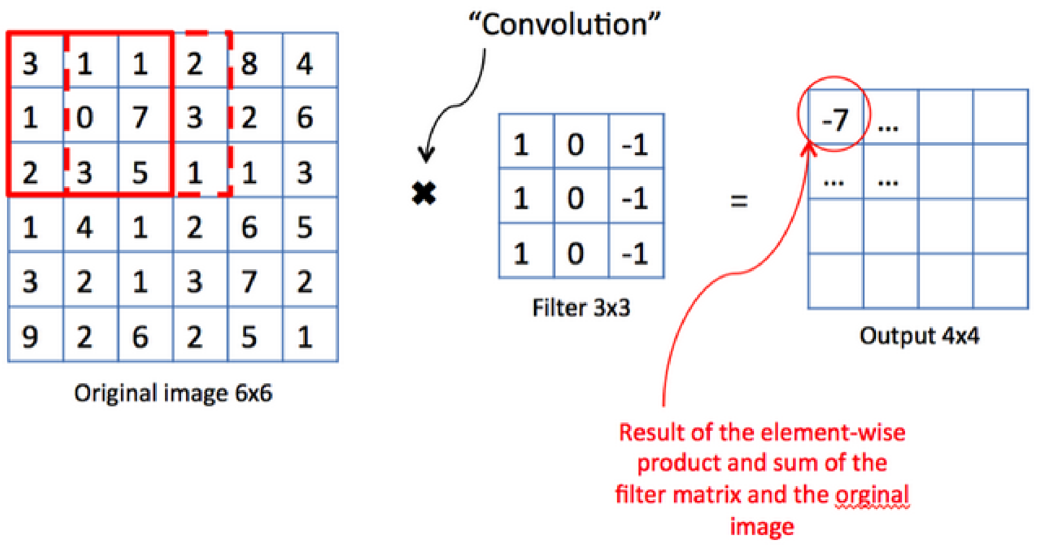

FIGURE 1.3: Convolution Operation

1.2.1 Convolutional neural networks

Convolutional Neural Networks (CNNs), like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The whole network has a loss function and all the tips and tricks that we developed for neural networks still apply on CNNs. But the main difference from ANN is that CNNs work in volumes. In CNNs, we have something called as 'kernels' which are closely related to 'receptive field of a neuron', which is the portion of the input, that the neuron is allowed to see.
An image is nothing but a matrix of values, each element of the matrix representing the pixel value in the image. Any image is made up of many features. The key role of the kernel is identifying these features. A kernel is generally, a matrix smaller than the image matrix size. This smaller matrix is run over the entire image and is convoluted with that part of the image matrix. This kernel values are learned such that we are able to extract features from the image. The size of one step of the kernel is termed as 'stride'.

In Figure 1.3, I have shown how convolution works on an input.

Here, the image is of size 6x6 matrix. Kernel is of 3x3 size. We run the kernel over the image as shown in red. The convolution works as follows:

$$3*1+1*0+1*-1+1*1+0*0+7*-1+2*1+3*0+1*-1=-7$$

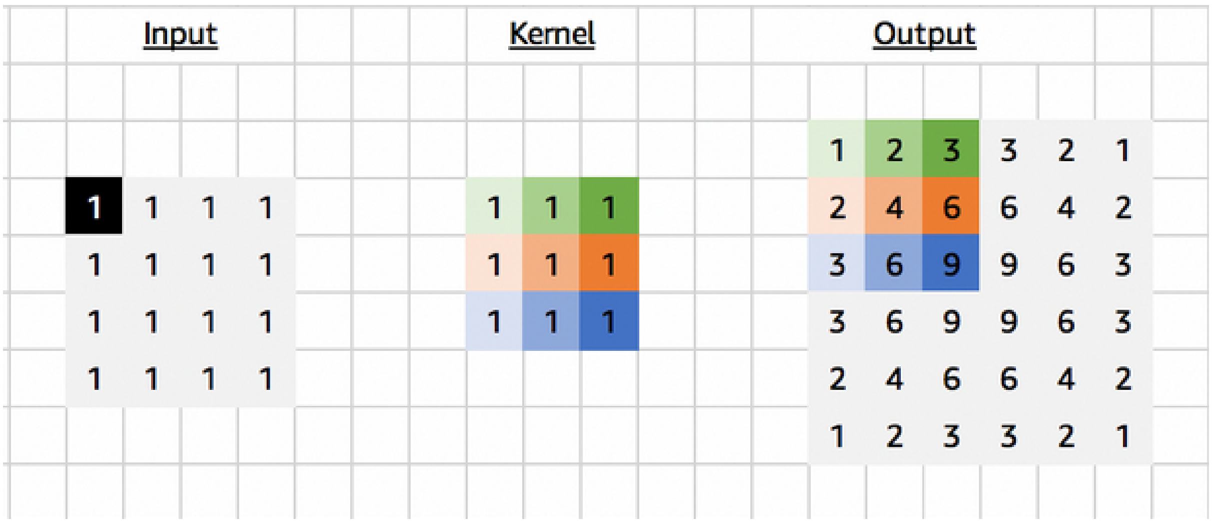

FIGURE 1.4: Trasposed Convolutional Layer operation

Final image is of size 4x4.
The kernel height, width, depth, padding and stride are parameters of the model that affect the size of the output. There are two operations, one is to compress an image and the other is to go from a smaller matrix to bigger one. Compressing a matrix (or keeping a matrix at same size) is done by convolutional layers. The transposed convolutional layers take a smaller matrix to a bigger matrix. A typical transposed convolutional layer operation is as shown in Figure 1.4.

1.3 Autoencoders

An autoencoder is an unsupervised learning technique.It has an encoder and a decoder in its architecture, the encoder converts the input into an intermediate representation, which shall be decoded by the decoder thereby reconstructing the input. It's architecture is as shown in Figure 1.5.

These networks try to approximate an identity function I(x) such that whenever it gets input x, $y = I(x)$. So basically, an autoencoder is trained to attempt to copy its input (x) to its output (y), i.e. $y \approx x$. Internally, it has a hidden layer that describes a code used to represent the input. This hidden code layer is a 'bottleneck' layer, which allows it to compress the data into the required details alone.
So in a nutshell, autoencoder‘s objective is to minimize reconstruction error between the input and output. This helps them to learn important features present in the data. When a representation allows a good reconstruction of its input, then it has retained much of the information present in the input.

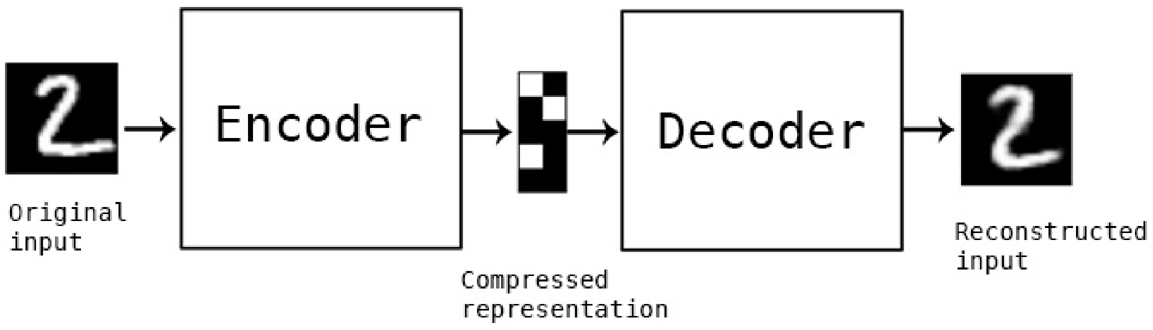

FIGURE 1.5: Autoencoder

1.4 Occlusions in Images

It is quite possible in many cases that the actual image object is occluded by the background or by some other object. In that case, it is a very challenging problem as far as a machine learning model is concerned. It is even more difficult when both the occluding object and the occluded image are targets of the model.

1.5 Recurrent connections and Occluded Images

Feedforward networks are those networks which have just the feed forward connections and no recurrent connections from higher levels to the lower levels whereas, recurrent networks are those, which have feedback connections in their architecture.

State-of-the-art ANNs are highly feedforward networks and have performed well in object detection tasks.[3] But at the same time, there is ample evidence that the visual cortex has a lot of recurrent connections [4]. This is quite surprising as feedforward networks exhibit high accuracy in object recognition in the absence of these feedback connections. [5] Furthermore, adding these extra connections introduces cycles in the structure, which introduces temporal dynamics to the models as well. Spoerer, McClure, and Kriegeskorte [2017] [6], have already shown that these additional recurrent connections (top down connections (T) and lateral connections(L)) in convolutional neural networks along with feedforward connections (B) improve the model's performance when tested on images with occlusions (for an introduction to this architecture, see Goodfellow et al., 2016 [7]). Figure 1.6 illustrates these connections. In Riccardo's thesis (see appendix), he has investigated the role of recurrence in supervised scenario and has provided evidence that recurrent connections help improve the performance of the model.

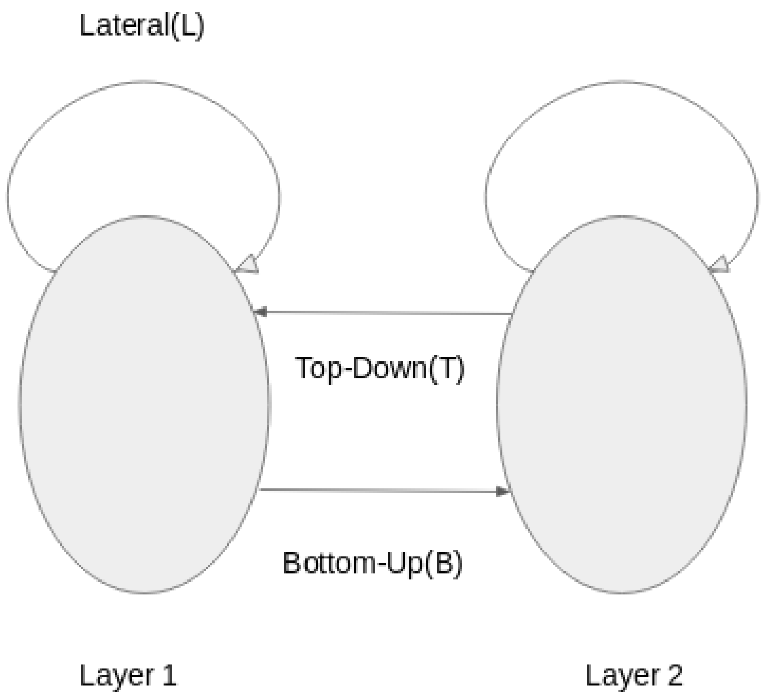

FIGURE 1.6: B, L, T connections

Occlusions are non-linear and they induce drastic information loss in the image. They introduce the concept of "depth" in an image, as in one object occluding the other induces a relative depth order. It has already been shown that recurrent connections help the network in recognising images under occlusion [6]. Now what we need, is to get a deeper understanding of how recurrence helps and to see if and how these recurrent networks represent depth when they are free to learn their own computations.

So in this thesis, I investigate the role of recurrent processing in object detection using autoencoders, especially in more challenging situations like occlusions in images. As a preview to the reader, chapter two deals with the architecture of the autoencoder, chapter three deals with the analysis and interpretation of the model. In the final chapter, I have stated my conclusions and future work.

Chapter 2

Methods

2.1 Data

We generated datasets for analysing the role of recurrent processing in object recognition under occlusion scenarios (The original code for these data sets was taken from Spoerer, McClure, and Kriegeskorte 2017 and adapted for these experiments). I have used digits as objects throughout the thesis. There are two cases, one where there are two digits occluding each other and second case where in there are three digits occluding one another.
A typical digit looks as shown below in Figure 2.1. Each image is first rendered at the resolution of 256x256 which is then down sampled to a resolution of 32x32.
The digits are black with white border and grey background. The font size used throughout is 190. One digit is drawn over the other thereby creating occlusions in the image. We can either fix the positions of the digits or place them randomly. Random placement of digits is taken by an offset value which is taken from uniform distribution. Randomly placing digits causes random occlusions which is a more challenging problem.

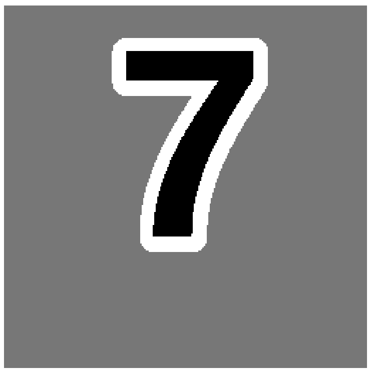

FIGURE 2.1: Digit seven

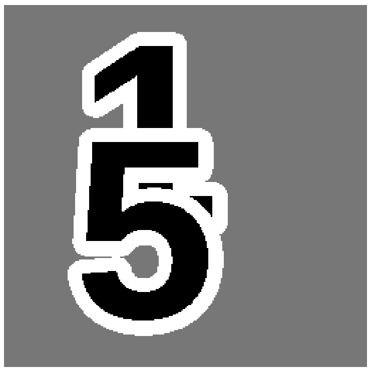

FIGURE 2.2: A five on top of one

A typical data point looks as in Figure 2.2

2.2 Architecture Overview

2.2.1 Recap: Autoencoder Architecture

In these experiments, I have used an autoencoder to reconstruct the input images. This autoencoder has an encoder, which compresses the information in the input image into a bottleneck layer and a decoder, which learns to reconstruct the input image from the code layer it has constructed.

2.2.2 Encoders

The first part of an autoencoder converts the image into a compressed code layer which encodes the important characteristics of the image into a smaller space, which are enough to reconstruct the input image. This is the inference task as mentioned in chapter 1. These networks are called encoders, as their aim is to encode high dimensional images into low dimensional representations. This kind of learning is also called a discriminative learning, where the aim is to discover $p(y|x)$ where x is the input data and y is the low dimensional representation. The Figure 2.3 shows a simple example of discriminative process with an encoder network.

2.2.3 Decoders

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. This is accomplished by decoders,

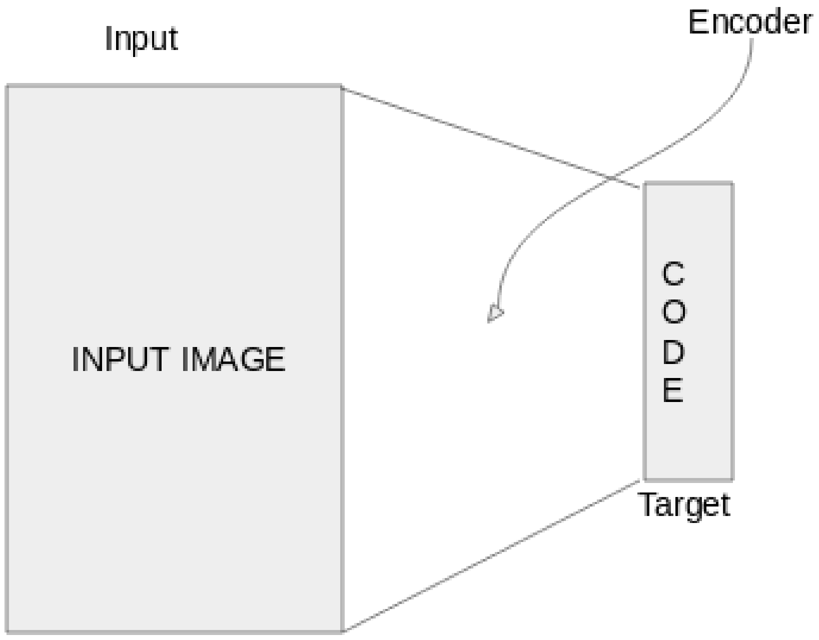

FIGURE 2.3: Encoder

they are called so as they take the representation and decode it to create an image. The Figure 2.4 shows a simple example of generative process with a decoder network.

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. A typical autoencoder architecture is shown in Figure 2.5.

2.2.4 Depth Inverting Autoencoders

2.2.4.1 Task

The task of reconstructing the input as it is, is itself a major task, as there is a bottleneck in the autoencoder which pushes the network to learn a good way to compress the input with minimum information loss. What is more interesting and challenging is to see whether and how recurrence might help networks to do inference on images with occlusions and to infer depth from them. So, the task of the autoencoder is to understand the identity of the digits, understand the relative

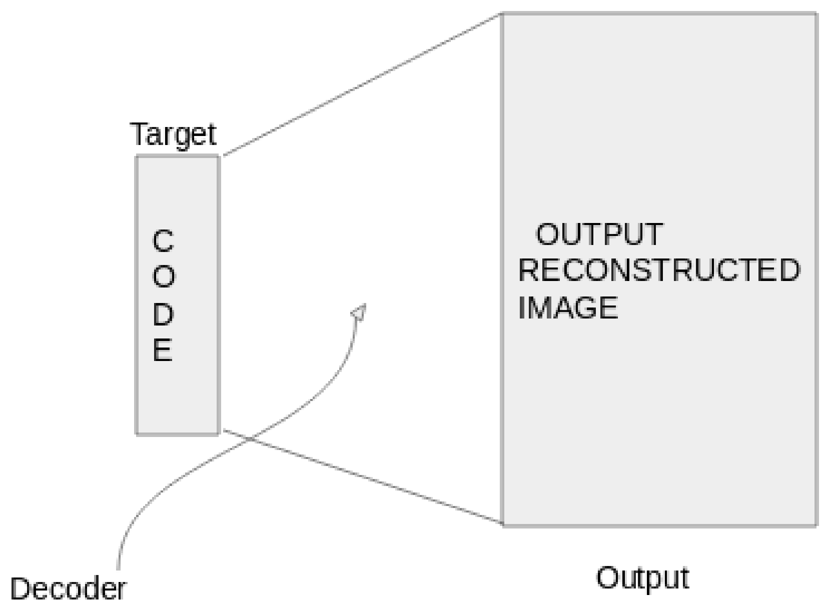

Figure 2.4: Decoder

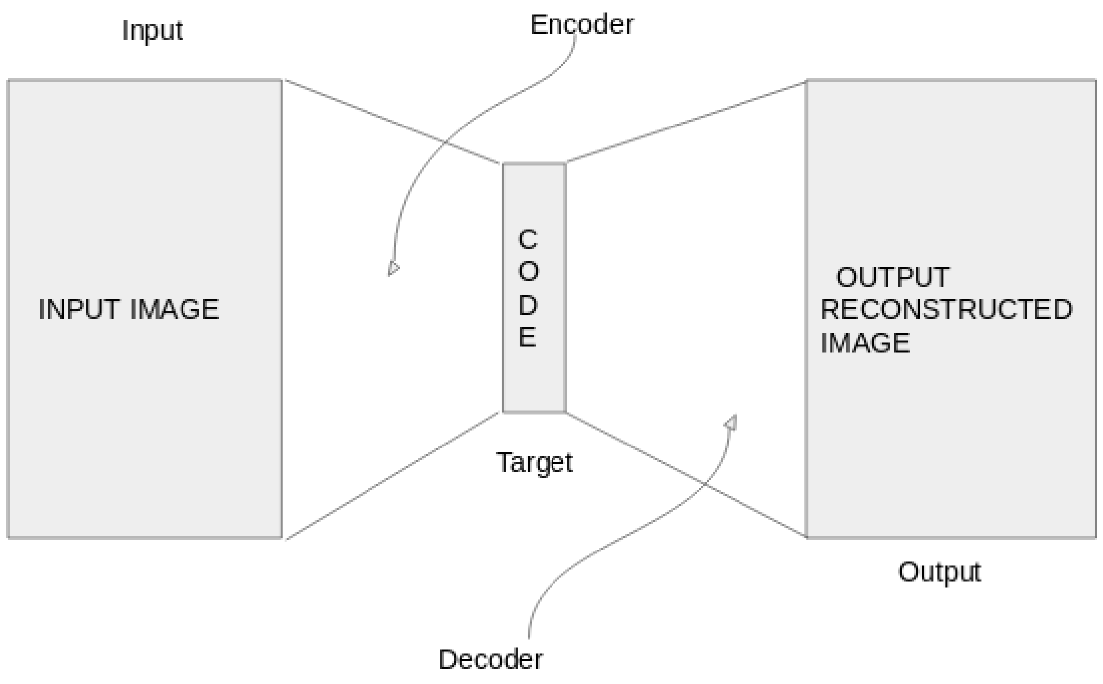

Figure 2.5: Autoencoder architecture

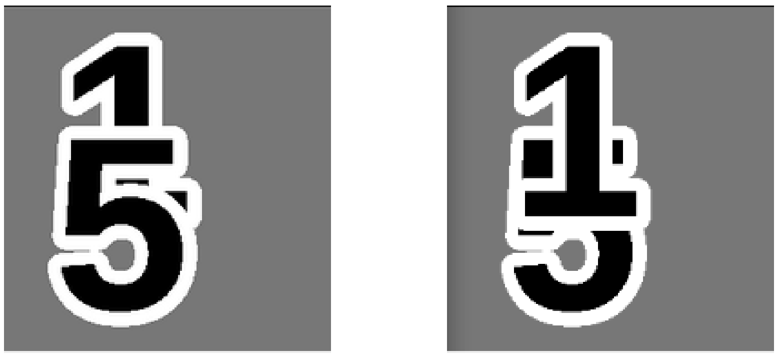

Figure 2.6: Input (Left), Target (Right)

depth (or the order in which the digits are presented), invert the order and reconstruct the digits as output. The input and corresponding target are as shown in Figure 2.6.

So, the task of the autoencoder is to understand the identity of the digits, understand the relative depth (or the order in which the digits are presented), invert the order and reconstruct the digits as output.

2.2.4.2 Architecture

In these experiments, both the encoder and decoder are convolutional neural networks along with the presence or absence of the lateral and the top down connections. As it does not make sense to have an architecture without feedforward, we are left with four possible architectures for encoder and decoder, which are B, BL, BT and BLT. These are illustrated in Figure 2.7. Adding top-down and lateral connections to the feedforward B networks creates recurrent dynamics that can be unrolled over time, as shown in figure 2.8.

As BL, BT and BLT have recurrent connections, they have more number of parameters than the purely feedforward network B. In order to make the number of parameters comparable, we can use parameter matched feedforward networks (B-K) wherein we increase the kernel size. This ensures that any change in results of B model from other recurrent models is not due the difference in the number of parameters but solely because of architecture differences.

All the models tested consist of three bottom up convolutional layers followed by three fully connected layers. In case of the encoder architecture, the dimensions (width and height) of the image is halved each time the input is passed through a convolutional layer. Input dimension is

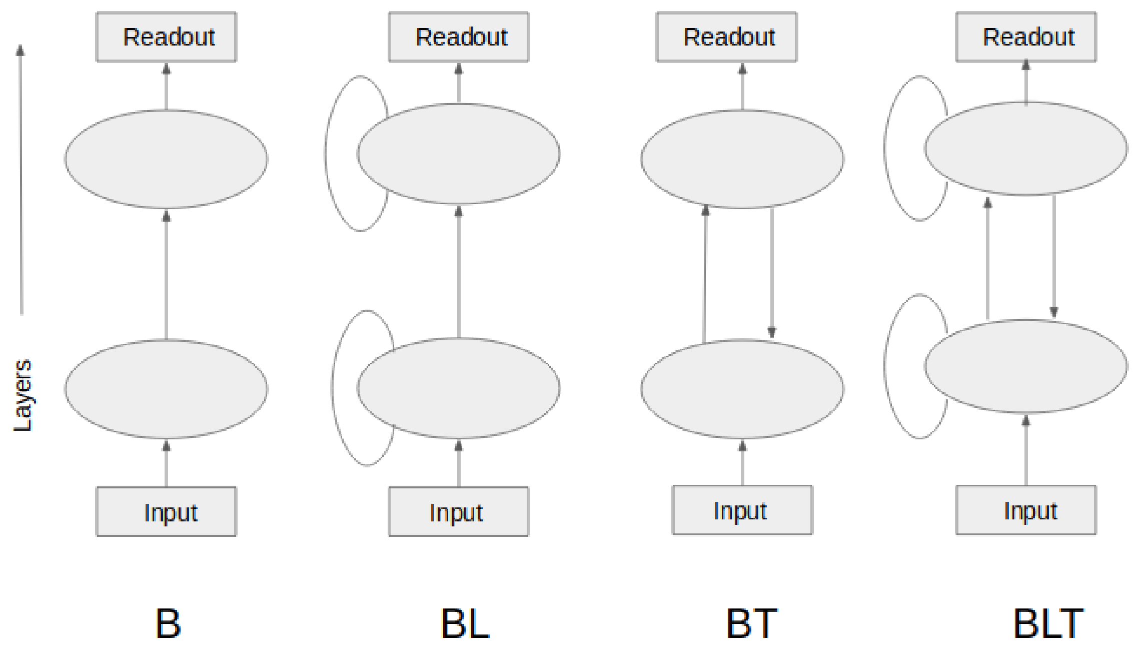

FIGURE 2.7: Schematic diagrams for each of the architectures

1x32x32, which becomes 32x16x16 after first conv layer, 32x8x8 after second and 32x4x4 after the third conv layer. This is then flattened (so dimension = 512) and passed through the linear layers, where it's dimension becomes 256 after first linear layer, remains at 256 after second linear layer and finally of code layer size.
In case of the decoder architecture, it is exactly the opposite of the encoder architecture. Initially, the code layer output passes through three fully connected layers, followed by three convolutional layers. After first linear layer, the dimension becomes 256, at remains at 256 after second linear layer and finally becomes 512 after third linear layer, which is reshaped to 32x4x4 and fed to the first conv layer where the dimension changes to 32x8x8. After second conv layer, dimensions become 32x16x16 and finally 1x32x32 after third conv layer. ReLU is used as the non linearity throughout all the models. These are illustrated in Figure 2.9 and 2.10.

2.2.5 Nomenclature

The autoencoder is named based on the encoder and decoder present in its model. The encoder and decoder architectures can have different levels of recurrence depending on the presence and absence of the 'L' and 'T' connections. Hence there can be four different types of encoders namely 'B', 'BL', 'BT' and 'BLT' and similarly for the decoders. Hence, for example, if 'BL' encoder and 'BLT' decoder is used, the the corresponding autoencoder shall be called as 'BL-BLT' autoencoder. As there four possible encoders and four possible decoders, there are a total of 16

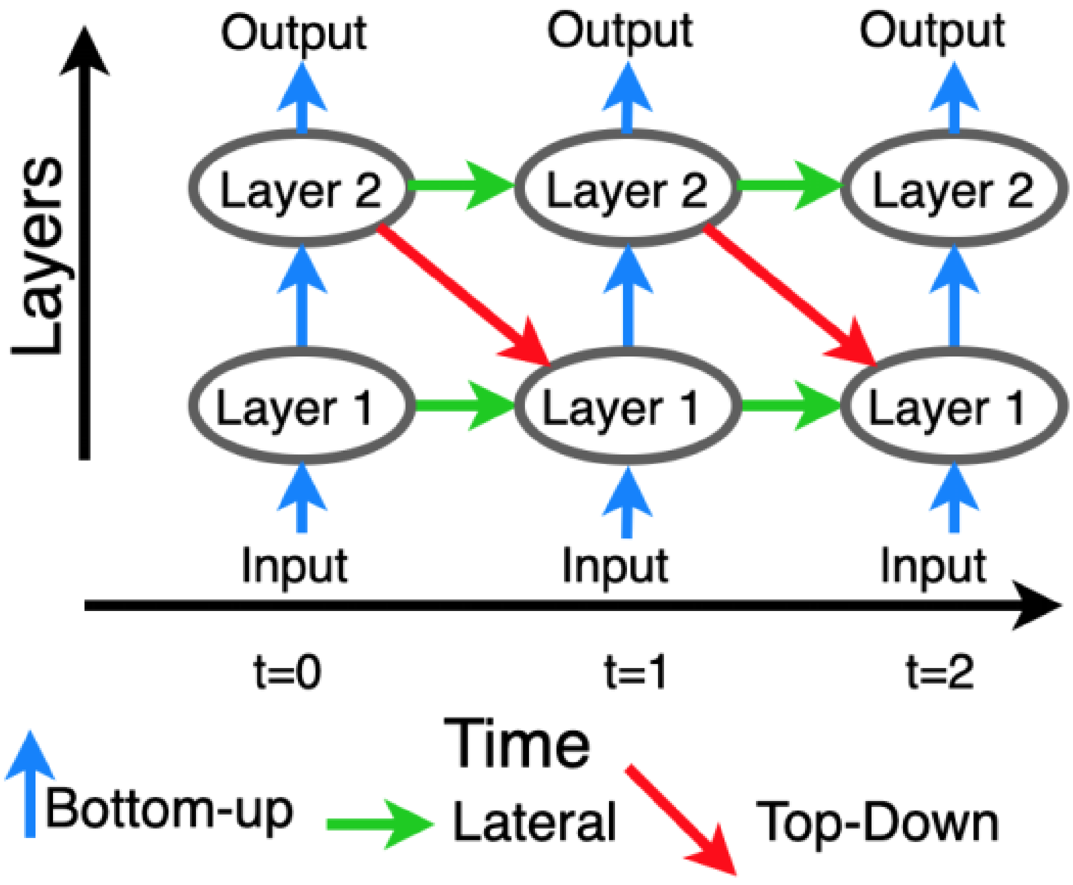

FIGURE 2.8: Example of unrolling a BLT network : Image taken from Riccardo's thesis

different autoencoder models possible.

2.2.5.1 Learning

Networks are trained to match the readout to target representation. The number of time steps defines the number of times computations are run before the final output is obtained. For the pure feedforward network (B network), the computations end at step 0. For all the other networks, BL, BT and BLT, the number of time steps used is set to four throughout the thesis. The encoder first runs for four times and then the decoder runs for four times and then the readout is compared with the target representation.

Mean squared error (MSE) is calculated between the readout values and the target. It is calculated as follows :

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

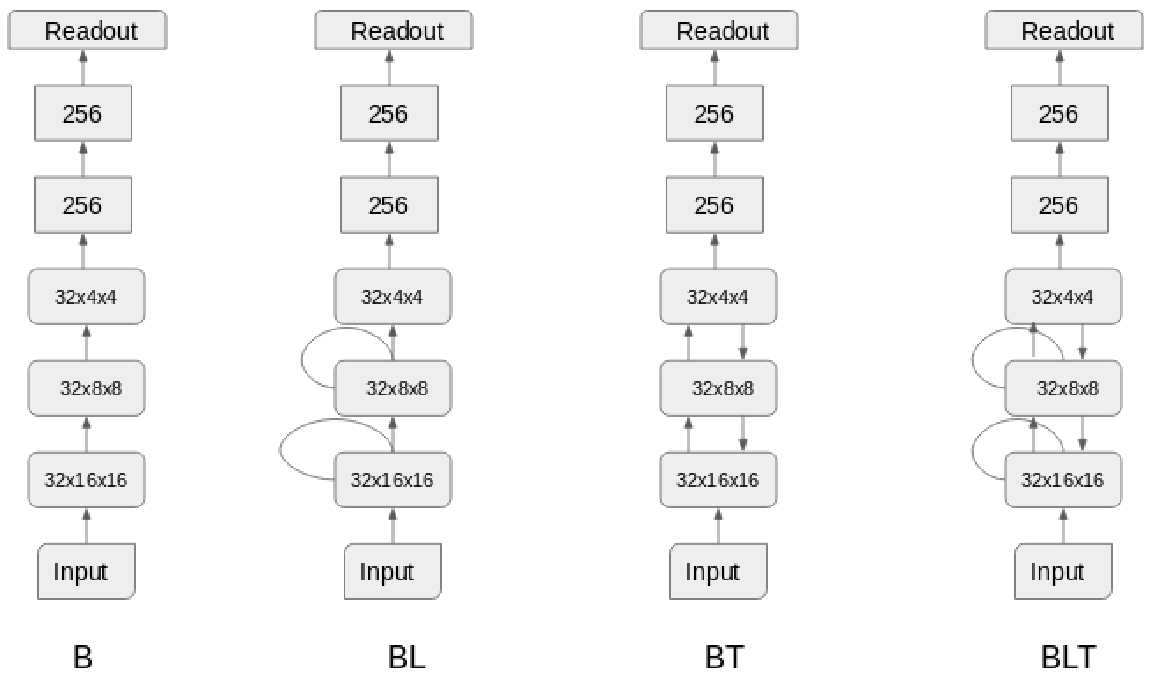

FIGURE 2.9: Schematic diagrams for each of the encoder architectures

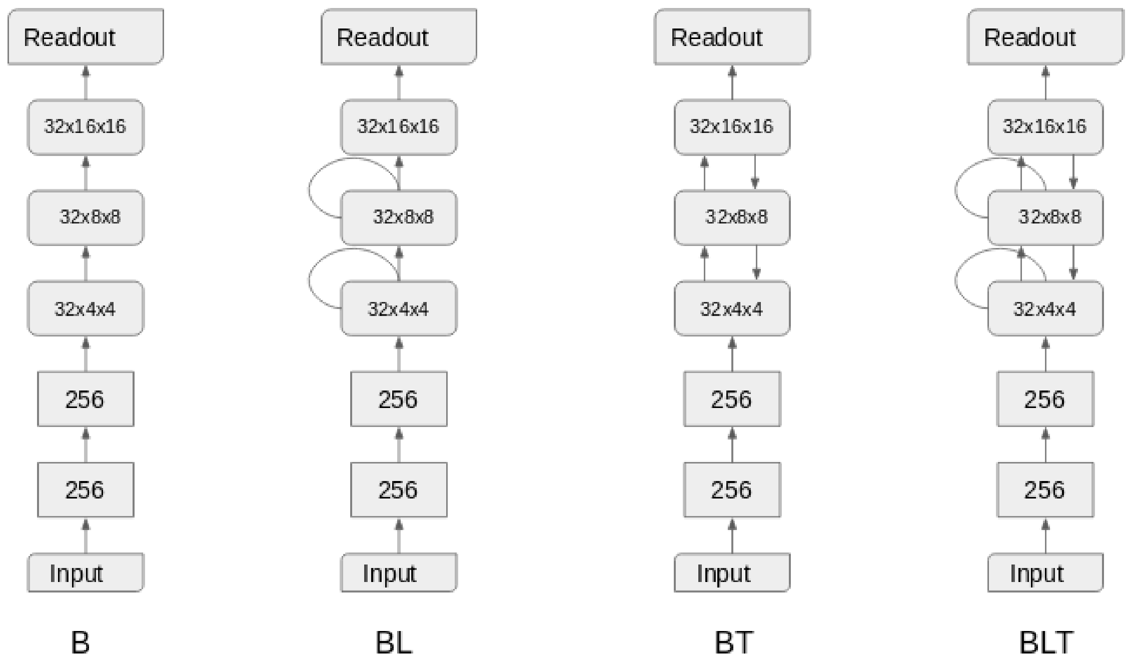

FIGURE 2.10: Schematic diagrams for each of the decoder architectures

To train the network, the mean squared error is backpropagated through time for each time point,which means that the network is trained to converge as soon as possible, rather than at the final time step.
I have not used any form of regularization, hence the total loss is just the mean squared error as calculated above. The loss is calculated after four time steps are run, and the loss at the final step is considered as the total loss. 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 beta1 = 0.9 and beta2 = 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 the following equation:

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

where ϵ is the new learning rate, η is the initial learning rate, δ is the decay rate, e is the epoch and d is the decay step. I have kept δ to be 0.1 and d to be 40 throughout all experiments.

As per the thesis by Riccardo, the best performances under supervised conditions are given by the BLT encoder and the BLT decoder. So, we presume that the best performance in unsupervised scenario shall be given by the autoencoder with BLT encoder and BLT decoder. In order to give the readers how a typical BLT-BLT autoencoder performs on the border2 data, I have shown the performance of the BLT-BLT autoencoder in the next subsection.

2.2.6 The performance of BLT-BLT autoencoder on border2 data

The model is first trained on the border2 training set with 100,000 examples and then tested on a test data of 1000 examples. As to see if the model is generalizing, the test data is made such that the model has not seen those combinations of digits while training. This is done so as to ensure that the model is not just memorizing the examples it has seen but is indeed executing the task.
The encoder and decoder are run for four time steps each. The size of code layer is set to 24. The font size is 190 and border size is maintained at 20. In the Figure 2.11, I have shown the train-generalization (gnrl) error plot. Lowest train loss and the lowest gnrl loss corresponds to the lowest loss point in the corresponding curve. Each point in the curve is the loss corresponding to a batch of examples from the train and test sets respectively. We can see that the error constantly decreases and flattens out after 80000 iterations. A few of the reconstruction outputs are as shown in figure 2.12.

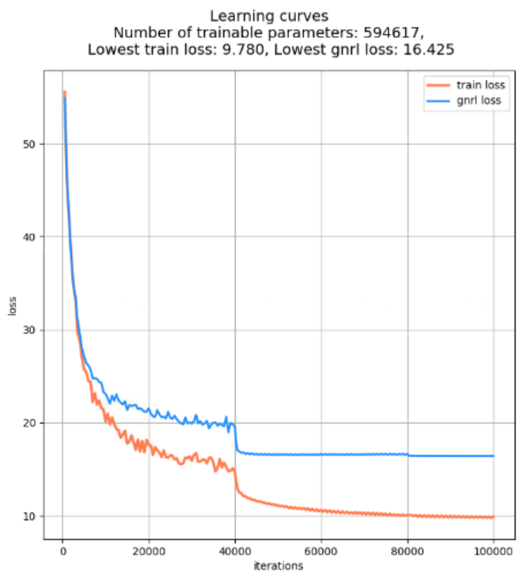

FIGURE 2.11: Learning Curve for BLT-BLT autoencoder : border2

In the first two figures, we can see that the model has reconstructed the input images pretty well. It has also inverted the order of the digits correctly. But the last two images show that the model is not able identify the digits in all cases. In the third figure, the model confuses between a zero and a six, whereas in the last figure, it confuses an eight to be a three.

It is necessary to see how changes in the code layer size or increasing the number of time steps affects the model performance.

2.2.6.1 Size of Code layer

The code layer contains the latent representation the model has come up. This hidden representation encodes information about the characteristics of the image like the position and identity of the digits present, which digit is in front and which one at back and so on.
Currently, in all experiments, I have kept the code layer size to be 24. It is possible by reducing the code layer size, the model would perform worse as now the model has to compress the entire information to a smaller space or perform better by increasing the code layer size.
Figure 2.13 to 2.18 are the images of reconstructed inputs for code layer sizes : 4, 8, 16, 24, 32, 64 respectively.

In Figure 2.19, I have plotted the lowest losses when increasing z values from 4 to 64. We can see that, after z = 24, the decrease in error is very less.

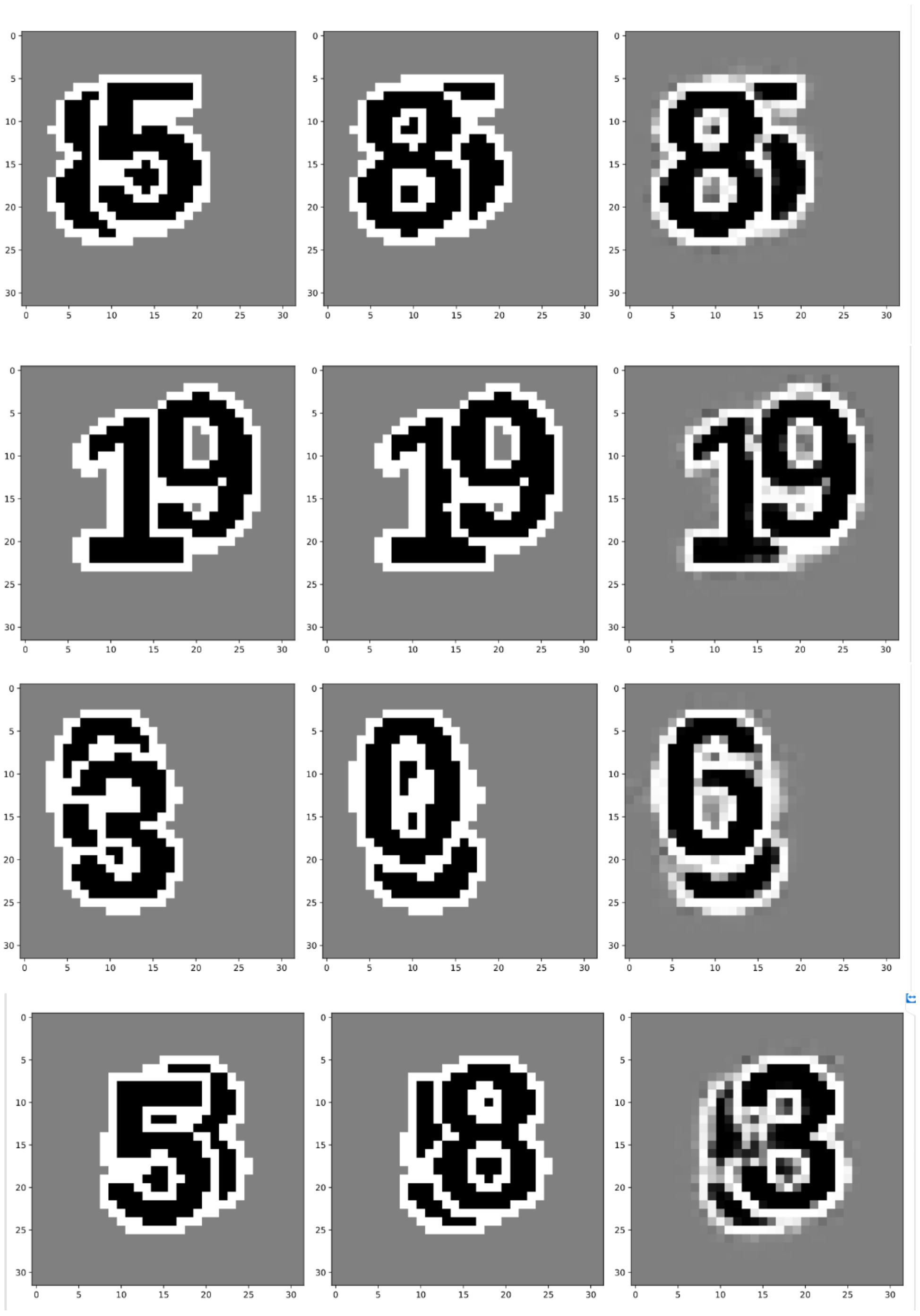

FIGURE 2.12: Input, Expected Output, Reconstruction Output

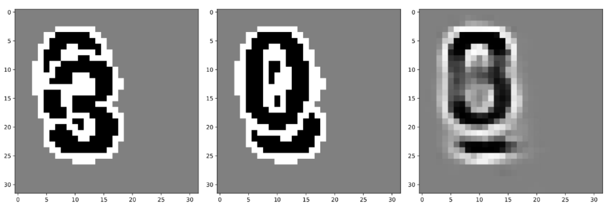

FIGURE 2.13: z = 4

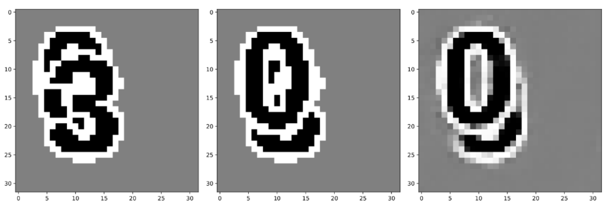

FIGURE 2.14: z = 8

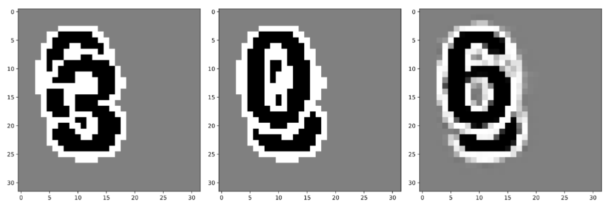

FIGURE 2.15: z = 16

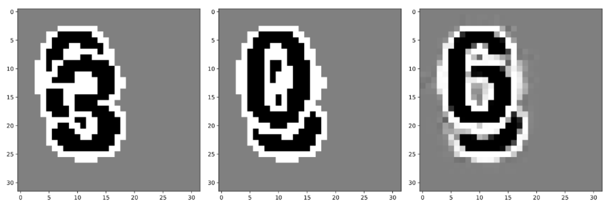

FIGURE 2.16: z = 24

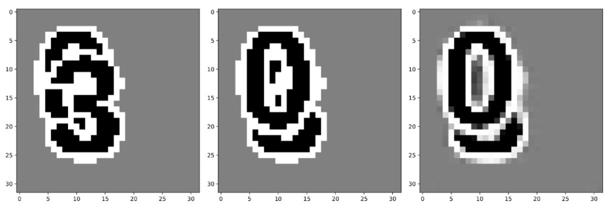

FIGURE 2.17: z = 32

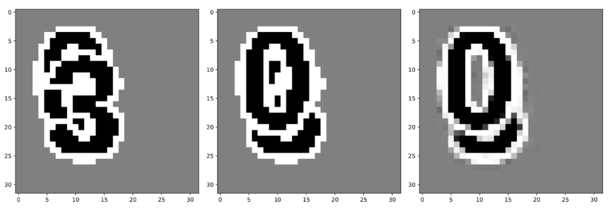

FIGURE 2.18: z = 64

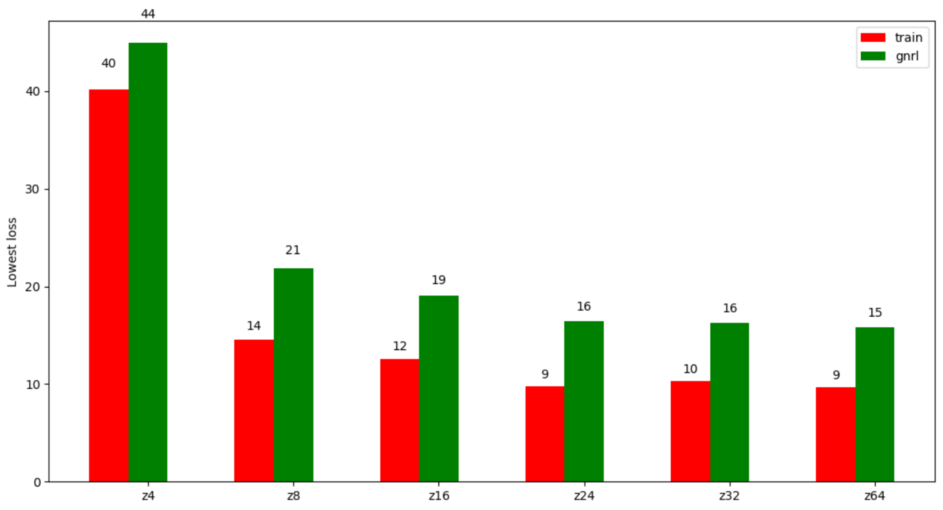

FIGURE 2.19: Lowest loss vs code layer size

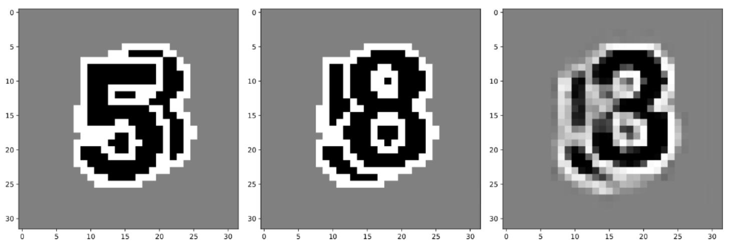

FIGURE 2.20: nrep = 1

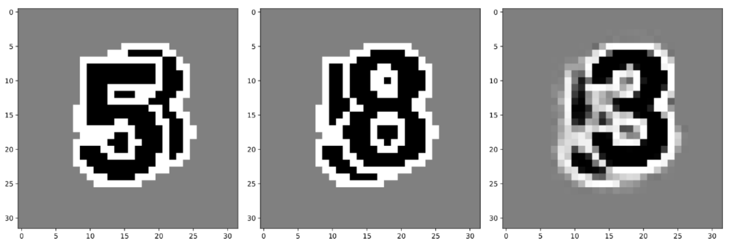

FIGURE 2.21: nrep = 2

2.2.6.2 Effect of the number of time steps

The number of time steps is the number of times the recurrent step is run in the model. Currently, I have run the model for four time steps. Number of time steps = N implies the encoder runs for N steps first and then the decoder runs for N steps. So intuitively, increasing the time steps indirectly deepens the model hence we can expect an improved performance of the model. Figures 2.20 to 2.23 are the images of the reconstructed inputs for time steps = 1,2,4,8. nrep = 1 implies, the model is equivalent to having no recurrent connections, it is the pure feed forward network. I have plotted the lowest losses when increasing the number of time steps (nrep) from 1 to 8 in Figure 2.24.

Above graphs and reconstruction outputs are shown to give the reader a view of how the

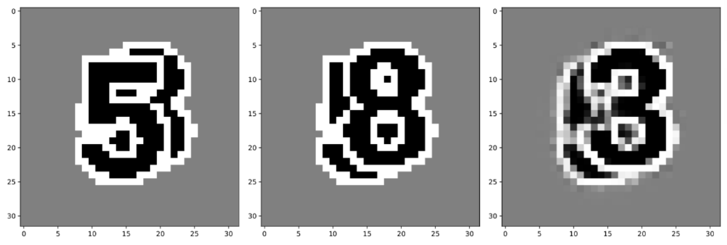
FIGURE 2.22: nrep = 4

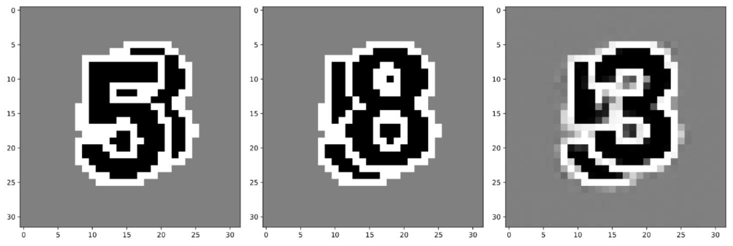
FIGURE 2.23: nrep = 8

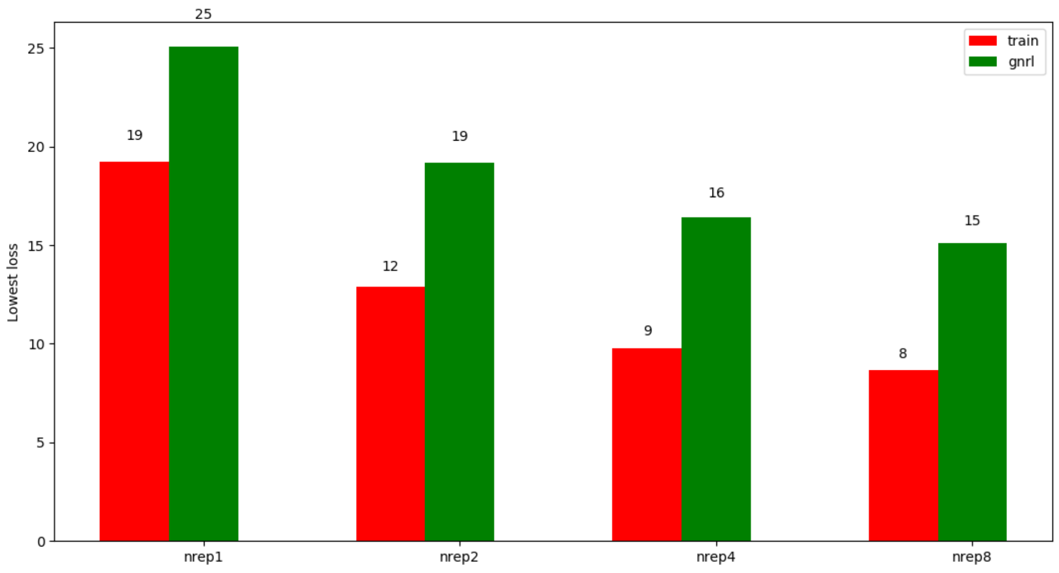

FIGURE 2.24: Lowest loss vs number of time steps

BLT-BLT autoencoder, which is presumably the best of the architectures, works on the border2 data. These ideas are take further in the coming chapters.

Chapter 3

Results

The hypothesis is that recurrent dynamics might improve recognition performance in the challenging scenario of partial occlusion. To validate this, we systematically compare architectures comprised of bottom-up (B), lateral (L) and top-down (T) connections, which is applied to convolutional neural network architectures. We choose to use the convolutional architecture as it is a parameter efficient method for building large neural networks that can perform real-world tasks. It is directly inspired by biology, with restricted receptive fields and feature detectors that replicate across the visual field (Hubel and Wiesel, 1968).[8] Advances based on this architecture have produced useful models for visual neuroscience as well (Kriegeskorte, 2015).[9] 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.

The current model is a convolutional recurrent neural network with lateral and top down connections. The main analysis conducted is to see how well the recurrent connections help in depth reversal as well and identifying digits under occlusions. We also want to see how well the model generalizes to higher number of digits in one data point ($n >= 3$).We first consider the results for two digit occlusions.

The analysis of the model is mainly concerned with interpreting the model. In this, I analyse the roles of the different connections in the network and state important modifications to the existing model in order to improve the current performance.

3.1 Interpretation of the model

The architecture proposed has reconstructed the input image in two occlusions scenario pretty well as shown in previous chapter. Now we would like to analyse how the model is able to perform

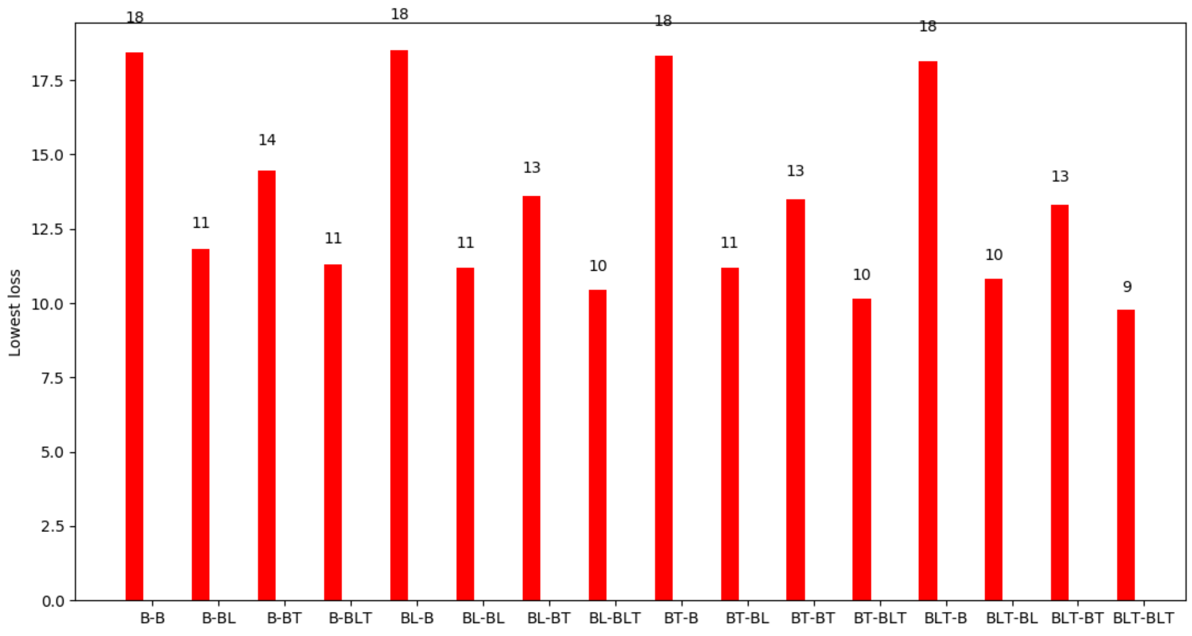

FIGURE 3.1: Comparison Bar Plot : train loss on border2 data

the task of reconstructing depth inverted image and also see how the recurrent connections help the model executing the task.

To interpret the model's behaviour, I proceed in a systematic way. First question is how the model performs by changing the encoder or decoder or both.

3.1.1 Performance of models with different encoders and decoders

We had assumed that the BLT-BLT autoencoder works the best, which has BLT encoder and BLT decoder. Now to understand how the top-down and lateral connections benefit the model, I either include them or not in the encoder and decoder and see how the performance of the autoencoder changes. As already mentioned in previous chapter, there are a total of 16 types of autoencoder models with different levels of recurrence in the encoder and the decoder. The first thing to see is how these 16 models differ in performance.
I have plotted the bar plot of lowest train loss and lowest test loss against the different architectures for autoencoders in Figure 3.1 and 3.2.

As predicted, the B-B autoencoder performs the worst and BLT-BLT autoencoder performs the best. But we have some important observations from the plot. It can be observed that BL-B, BT-B and BLT-B autoencoders performs almost as bad as B-B autoencoder. All the models with the lateral connection in decoder perform almost as well as the BLT-BLT model, which

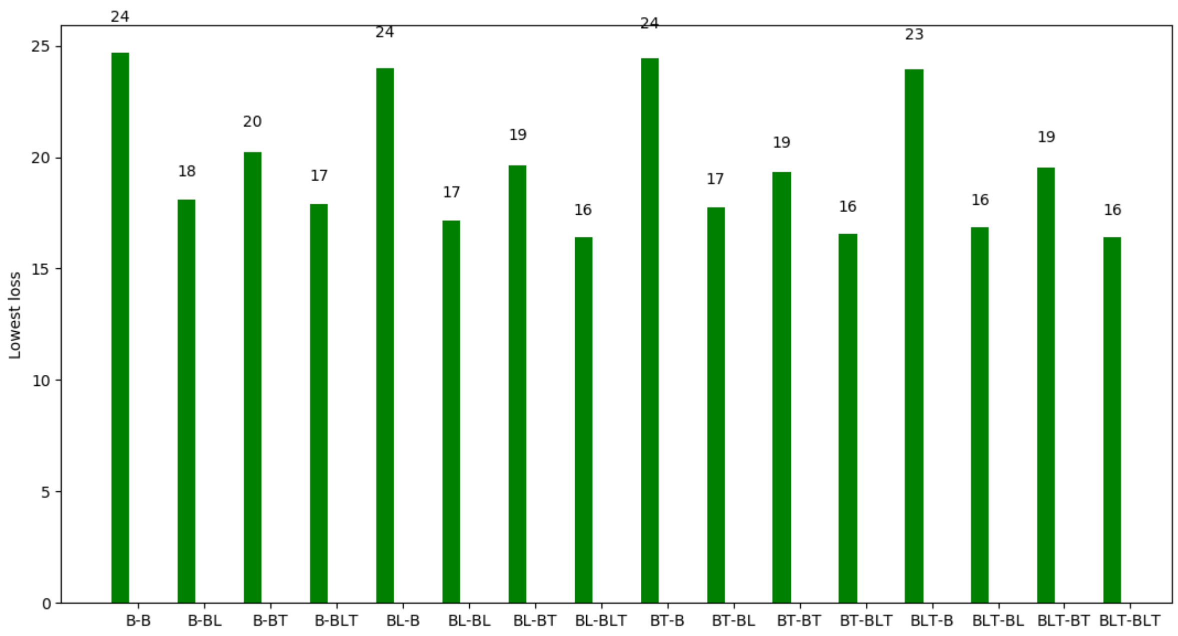

FIGURE 3.2: Comparison Bar Plot : gnrl loss on border2 data

also has lateral connections in the decoder. Thus, we can see that the lateral connections in the decoder plays a major role in the performance of the model.

We can also observe that lateral connections are more important than the top down connections in the decoder. It can be clearly observed that B-BT performs worse that any model with lateral connections in the decoder. But, B-BT performs better than B-B or any encoder along with B decoder. To see this closely, I have plotted a similar bar plot, showing the lowest loss against the different autoencoder models with encoder fixed as BLT and varying the decoder and with decoder fixed as BLT and varying the encoder in Figure 3.3 and 3.4 respectively. We can see that in Figure 3.3, the decrease in error when increasing level of recurrence in the encoder is not significant, whereas, in Figure 3.4, increasing level of recurrence in the decoder is significant especially the lateral connections.

As we can see that some models are equivalent, I have conducted t-test to see how statistically significant our observations are. Each of the 16 models were run on 1000 examples and the reconstruction losses were noted. t test was conducted on the losses across the different models. The results showed that the models with varying recurrence in the encoder are not different. It was also seen that BL and BLT decoders are not significantly different.

3.1.2 Are the digits indeed identified and inverted in order?

Next thing to see is whether the network is indeed learning to reconstruct the input images the way we want it to. It is possible that the network is learning by not exactly identifying the digits

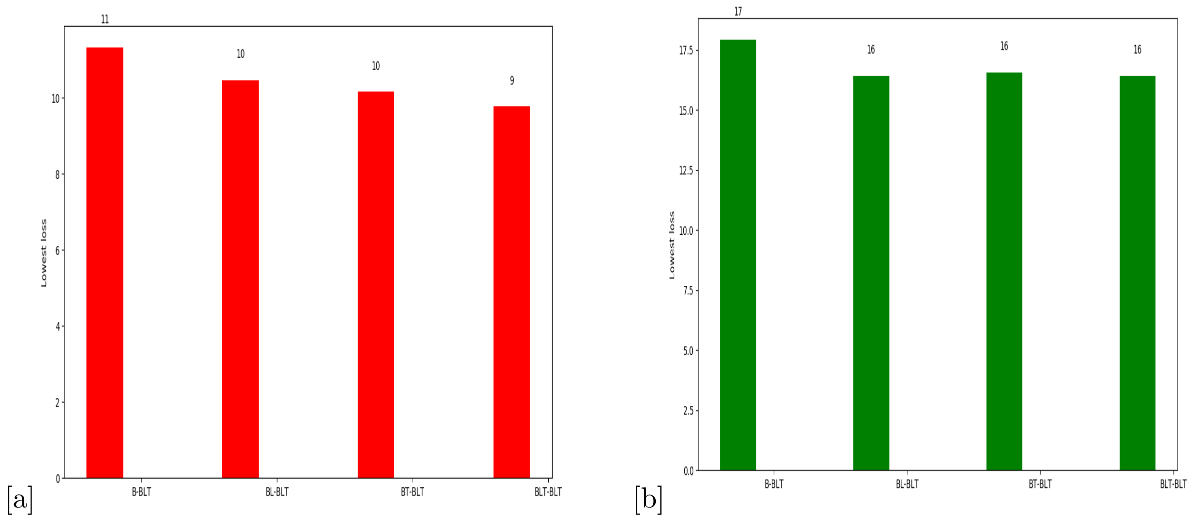

FIGURE 3.3: Encoder is varied and Decoder is fixed as BLT : (a) shows the train loss (b) shows the generalization loss

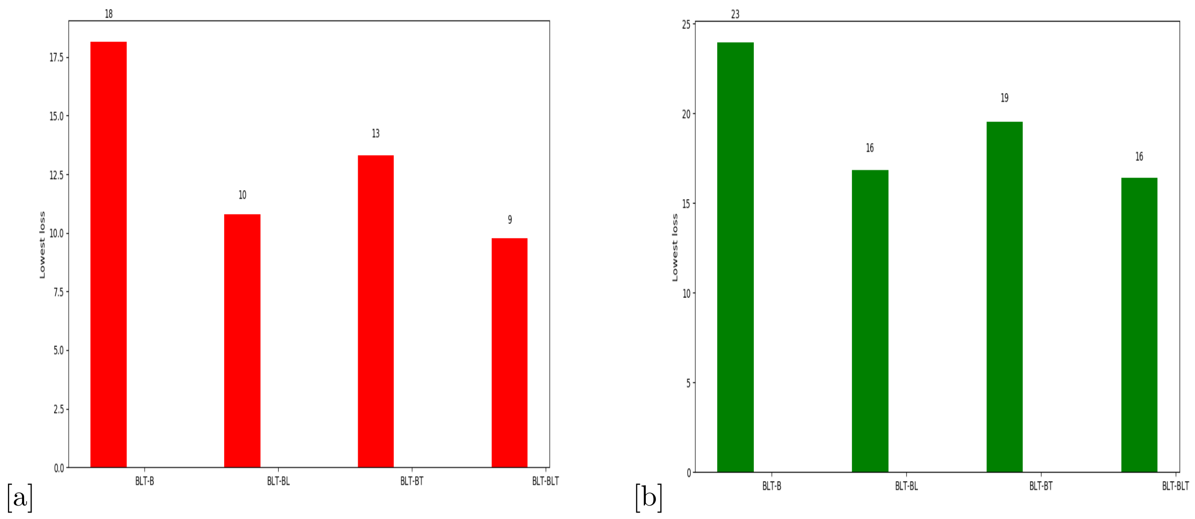

FIGURE 3.4: Decoder is varied and Encoder is fixed as BLT : (a) shows the train loss (b) shows the generalization loss

and inverting them in order. One way is that the network is breaks the digit into different pieces and encodes which all parts are to be present in the reconstructed output. If the model is indeed dividing the digits into parts and deciding on which parts are present in the final output or if digits are not identified as whole, then in any case, it must not be able to get the reconstruction output right, if the input has digits that are partially visible.
Hence, I made another dataset where the digits are only partially visible. Both the digits are drawn first on gray background, like border2 data. Then a gray rod is superimposed randomly on the image so that the occluding digits are partially hidden. A typical data point is as shown in Figure 3.5. Let this data be called part-border2 dataset.

The model is first trained on border2 data. Then this part-border2 data is fed as test data. If

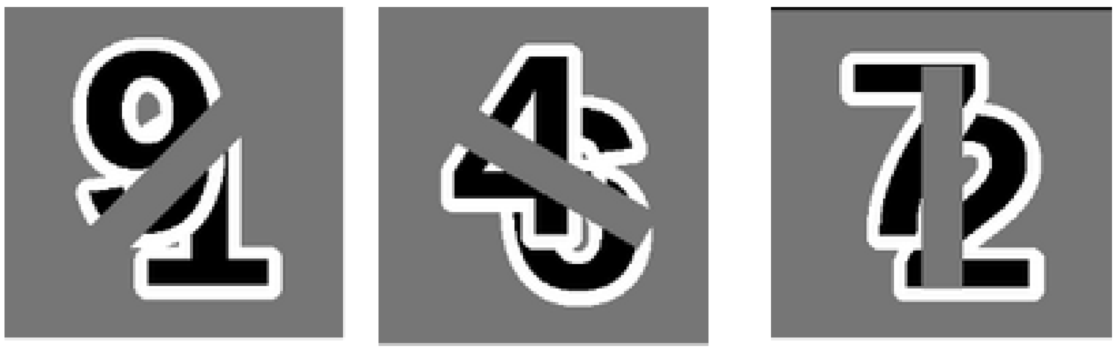

FIGURE 3.5: Examples for part-border2 data

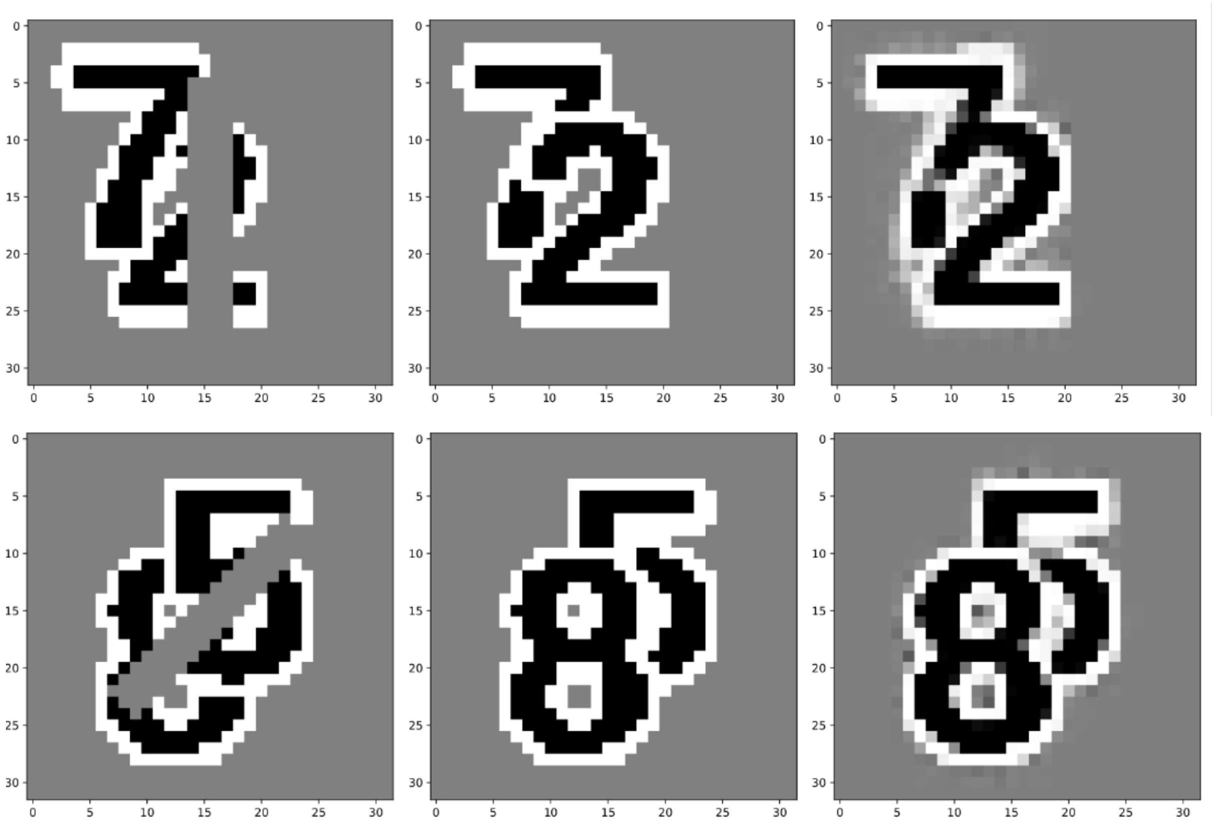

FIGURE 3.6: part-border2 data run on BLTBLT autoencoder

the network is not learning the digits as a whole but instead doing by parts, then the model will replicate the rod as well in the reconstructed output.

The results of BLT-BLT autoencoder as well as B-B autoencoder are as shown below in Figure 3.6 and 3.7. We can see that the rod is absent in the reconstructed output. This ascertains the fact that the model indeed is identifying the digits and inverting them in order. We can

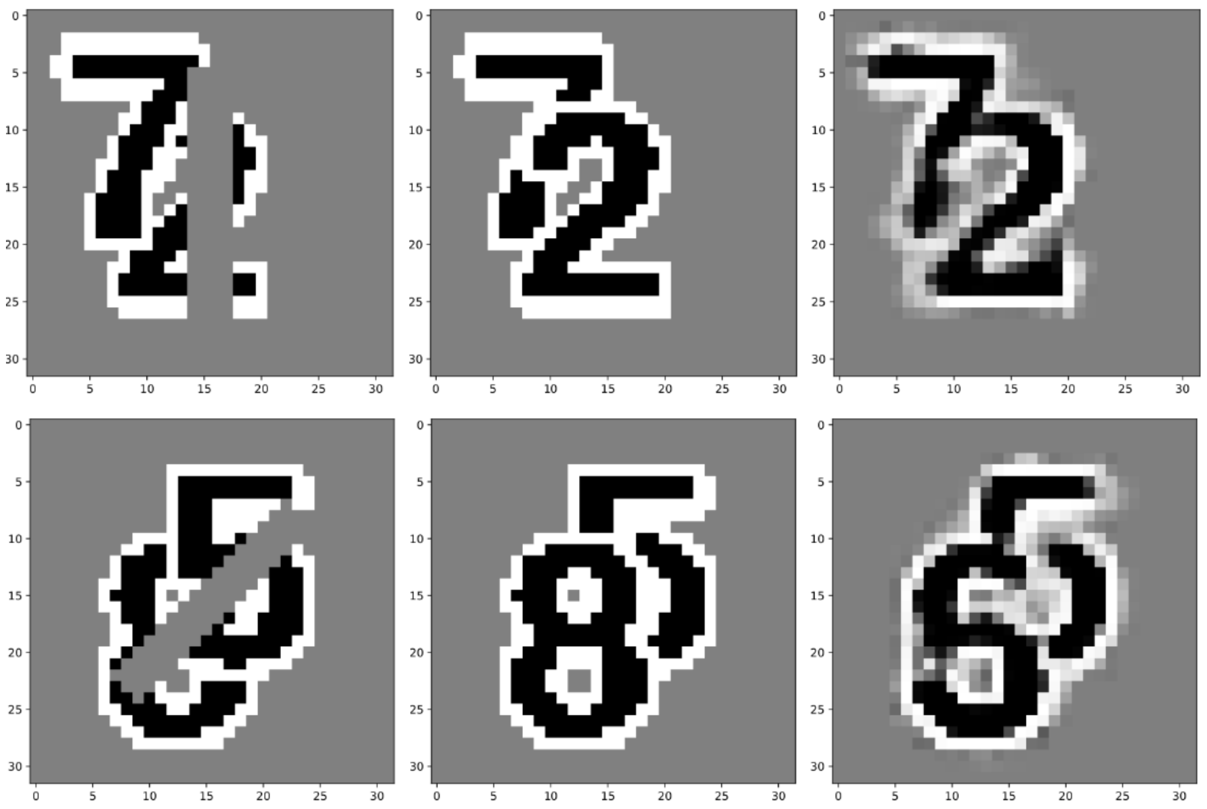
FIGURE 3.7: part-border2 data run on BB autoencoder

observe that both BLTBLT and BB autoencoders reconstruct the input image without the bar. They also identify the digits correctly, even if there is a lot of information loss due to the grey bar. This suggests that recurrence is not a necessity for identifying the digits nor infer the depth from the input and invert them in order.

3.1.3 How well does the model work when occlusion is not present?

We have seen that both B-B and BLT-BLT autoencoders work when digits occlude each other. Now the question is whether the benefit we saw for recurrence with occlusive digits actually had anything to do with the occlusions. It is so possible that they just reflected a general sharpening of the generated images. If that is the case, a similar pattern should also translate to non-occlusive digits also. So, I created a new data set wherein the digits are added together so that both input and target look the same. Here, there is no more occlusion from an algorithm's perspective as both the digits are visible. What I would like to know here is whether the benefit of recurrence actually had anything to do with the occlusions in the previous images.

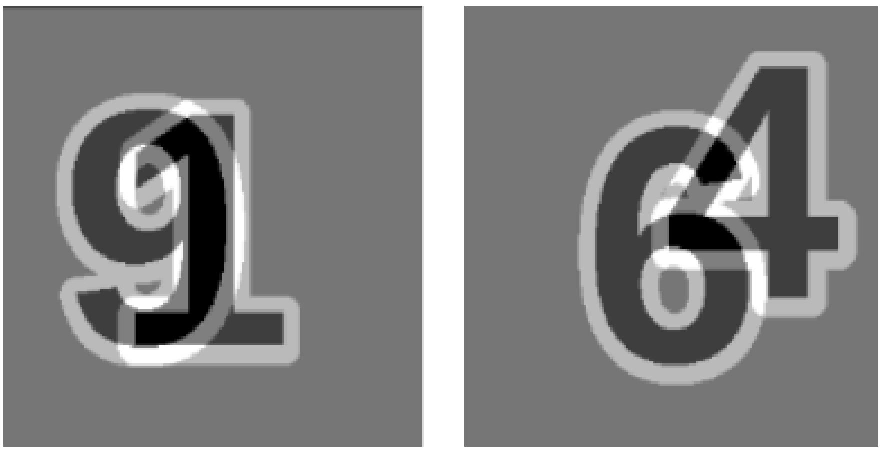

FIGURE 3.8: added-border2 data

3.1.3.1 Data

A typical input and corresponding target is as shown in Figure 3.8.

This data is created by first drawing both the digits in gray background individually in two different images. Both digits are drawn exactly the way as in border2. The digits are of black colour with white background and of font size of 190. Then both the images are blended with each other with blending factor of 0.5. So we can see that wherever the images intersect, the pixels are brighter otherwise, the value is average of the pixel values. This data has only two digits in the input image. We shall term this data as added-border2 dataset from now on.

3.1.3.2 Performance of the models on added-border2

We have seen the performance of the 16 models on the border2 data. Now I have run all the 16 models on this dataset as well. In figure 3.9, I have shown a typical input and the expected target output. So here, as the digit pixels are just added, effectively, the input and target are the same.

I have shown the reconstruction outputs of the BLT-BLT autoencoder and B-B autoencoder in Figure 3.10 and Figure 3.11 respectively. It can be seen that the models do pretty well on this data as well. This implies that the model learns to reconstruct input even when they aren't occluding each other. Moreover, we can observe a similar pattern in the results as in case of border2 dataset.

BLT-BLT autoencoder performs the best and B-B autoencoder performs the worst. BL-B, BT-B and BLT-B performs equally bad as B-B. Here again, autoencoder models with a decoder

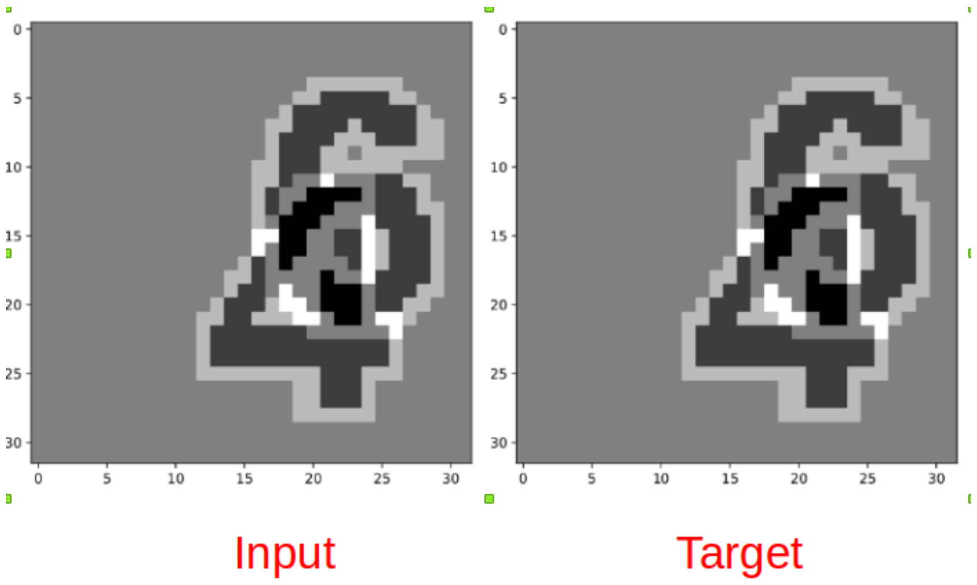

Figure 3.9: Typical input and target for non-occlusive digits

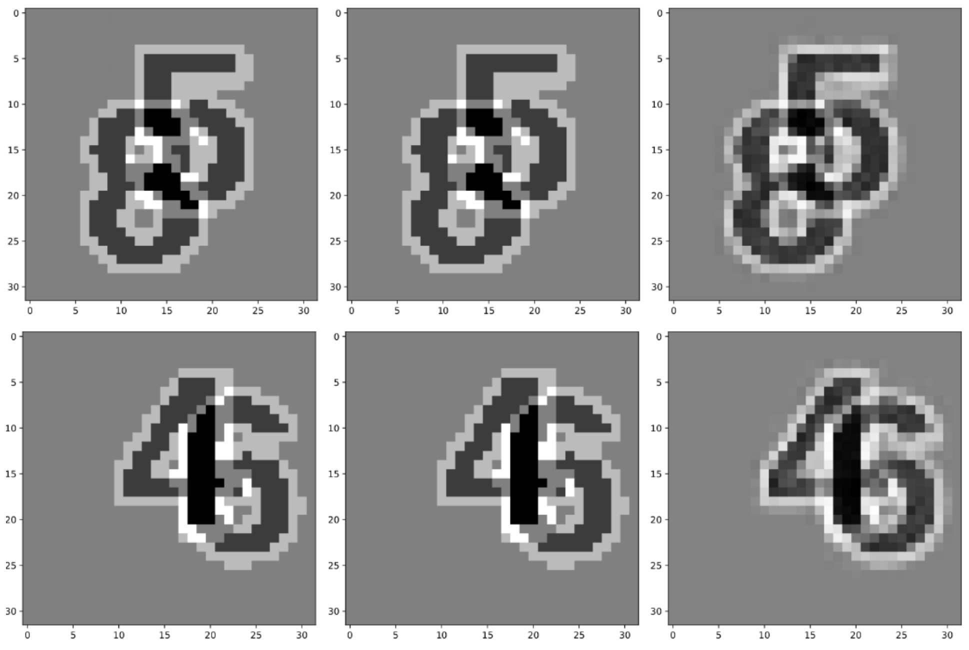

Figure 3.10: Input, Target and the corresponding reconstruction output : BLT-BLT autoencoder

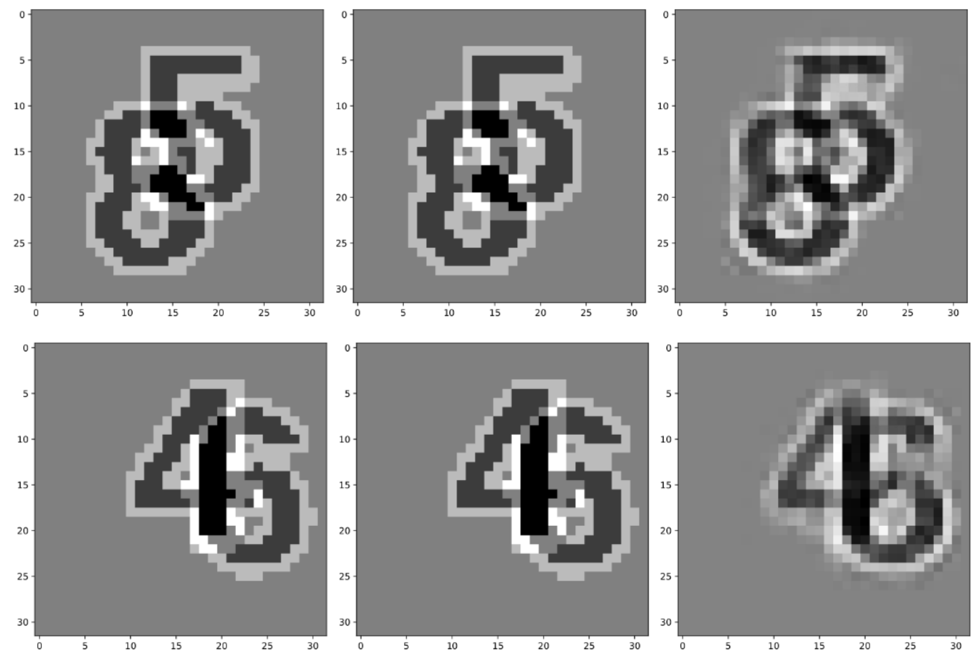

FIGURE 3.11: Input, Target and the corresponding reconstruction output : B-B autoencoder

with lateral connections performs as good as BLT-BLT autoencoder. Models with top down connection in decoder performs better than B-B, BL-B, BT-B and BLT-B models but performs worse than models with decoders with lateral connections. In Figure 3.12 and 3.13, lowest train loss and lowest generalization loss have been plotted against the different architectures used.

In Figure 3.14 and Figure 3.15, I have shown how the the performance varies when the levels of recurrence is varied in encoder and decoder. It can be noted that a similar pattern as in border2 data is followed here as well.

I have conducted t-test to see how statistically significant our observations are, as before. Each of the 16 models were run on 1000 examples and the reconstruction losses were noted. t test was conducted on the losses across the different models. A similar pattern as in the border2 data was seen in t-tests for added-border2 data.

These results along with results with border2 data, affirms the fact that feedback or lateral connections in encoder do not play a significant role in the model performance. This can be understood from the fact that both B-BLT and BLT-BLT performs equally well on the test data. But these connections are important in the decoder and they improve the model's performance significantly. Another important observation is that lateral connections in decoder play a greater role than the top down connections.

We say one model is better than the other solely based on the errors of the model. But, here we must see that there is background of another colour, which has also to be reconstructed.

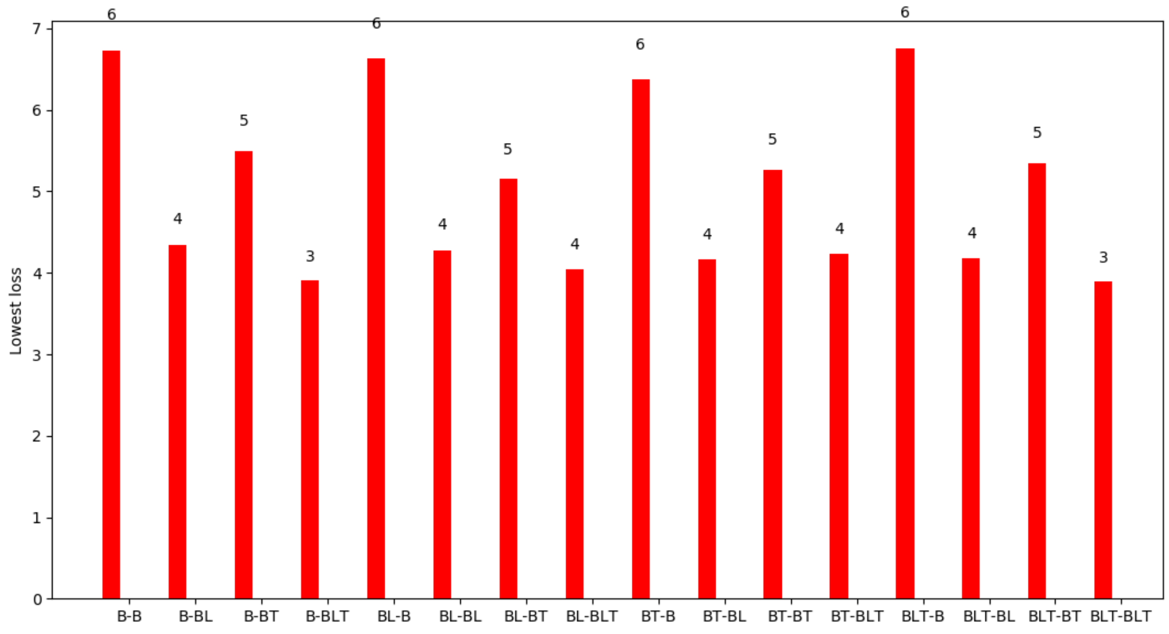

FIGURE 3.12: Comparison Bar Plot : train loss on added-border2 data

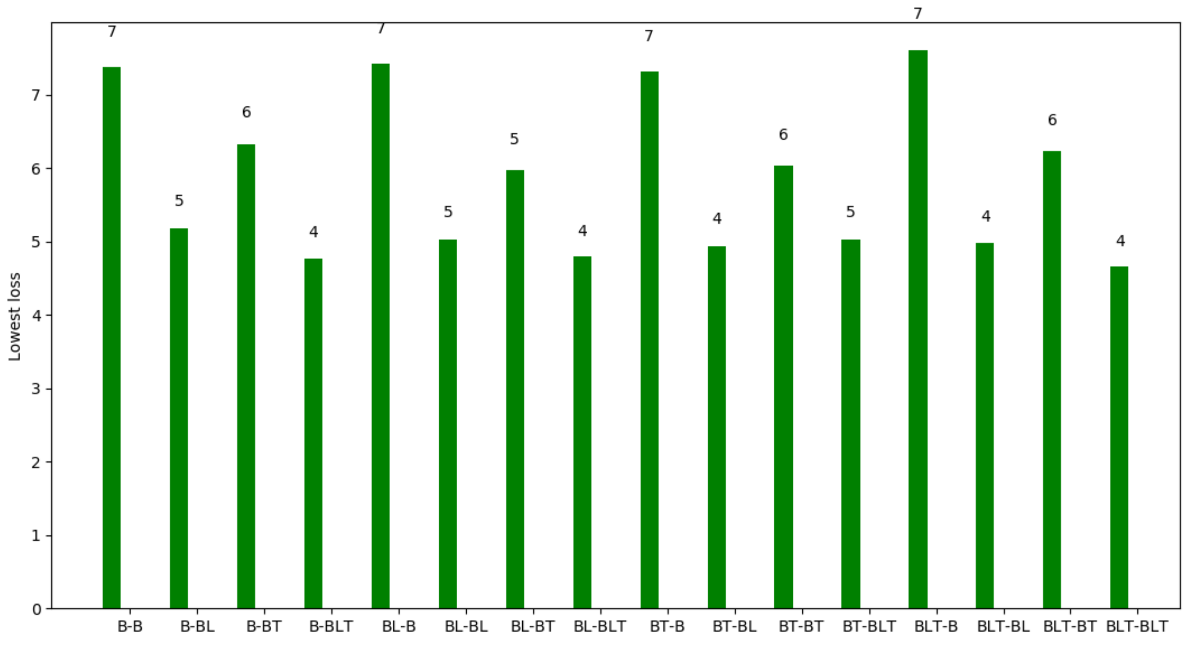

FIGURE 3.13: Comparison Bar Plot : generalization loss on added-border2 data

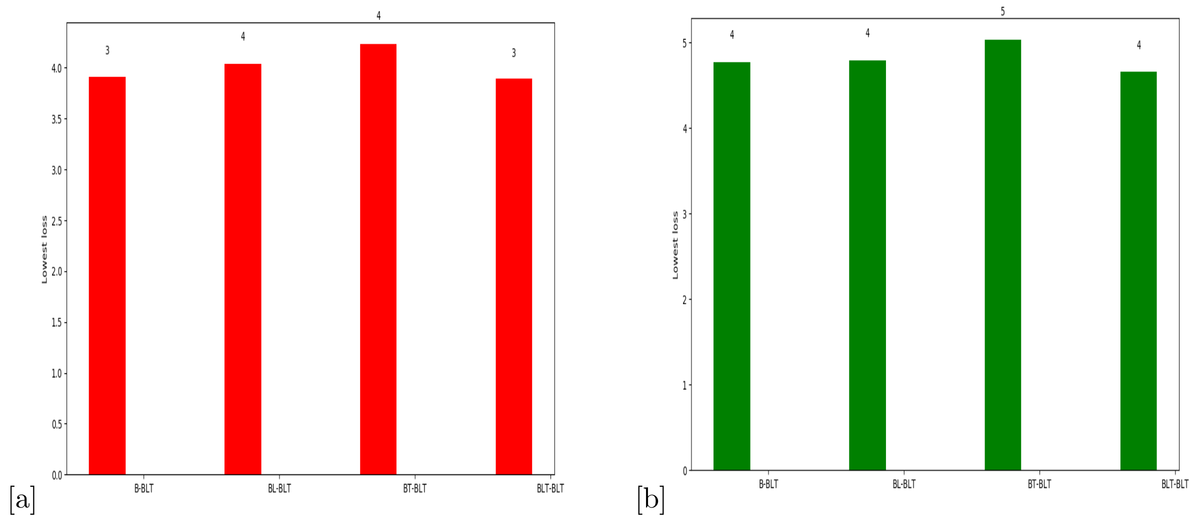

FIGURE 3.14: Encoder is varied and Decoder is fixed as BLT : (a) shows the train loss (b) shows the generalization loss

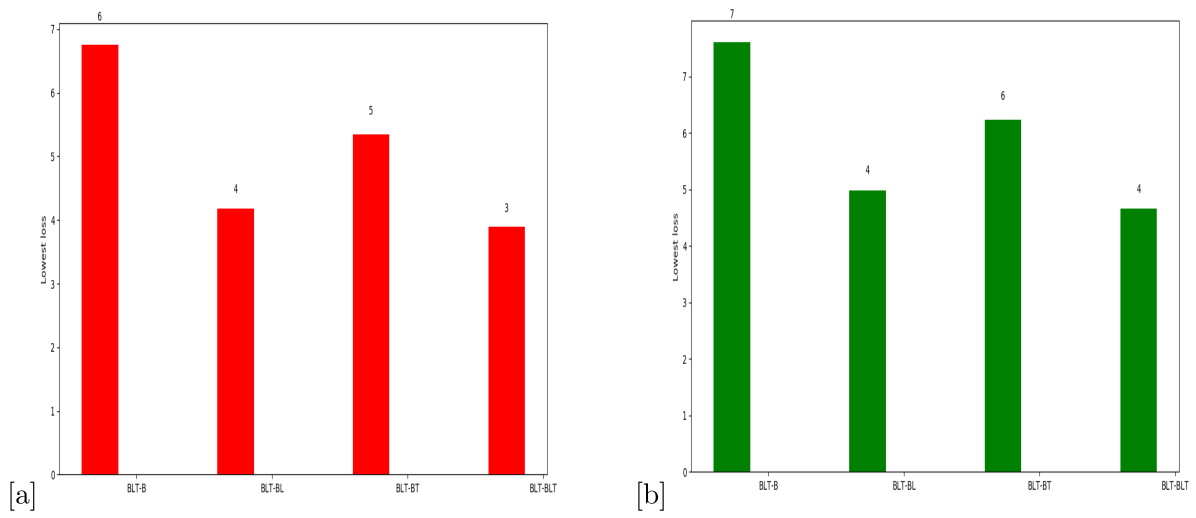

FIGURE 3.15: Decoder is varied and Encoder is fixed as BLT : (a) shows the train loss (b) shows the generalization loss

Changes in background (caused due to blurring) also affects the error. So, the error may not completely justify a model's performance.

3.1.4 Higher Levels of Occlusion

In order to push the model further, I increased the level of occlusion in the image by adding one more digit into the image. The motivation to do this analysis is because two is a special case where a digit is either an occluder or being occluded, but never both. So I would like to rule out that the network has learnt a special solution for this special case that doesn't generalize. In figure 3.16, I have shown a typical input and the expected target. I shall term this dataset as 'border3' from now on.

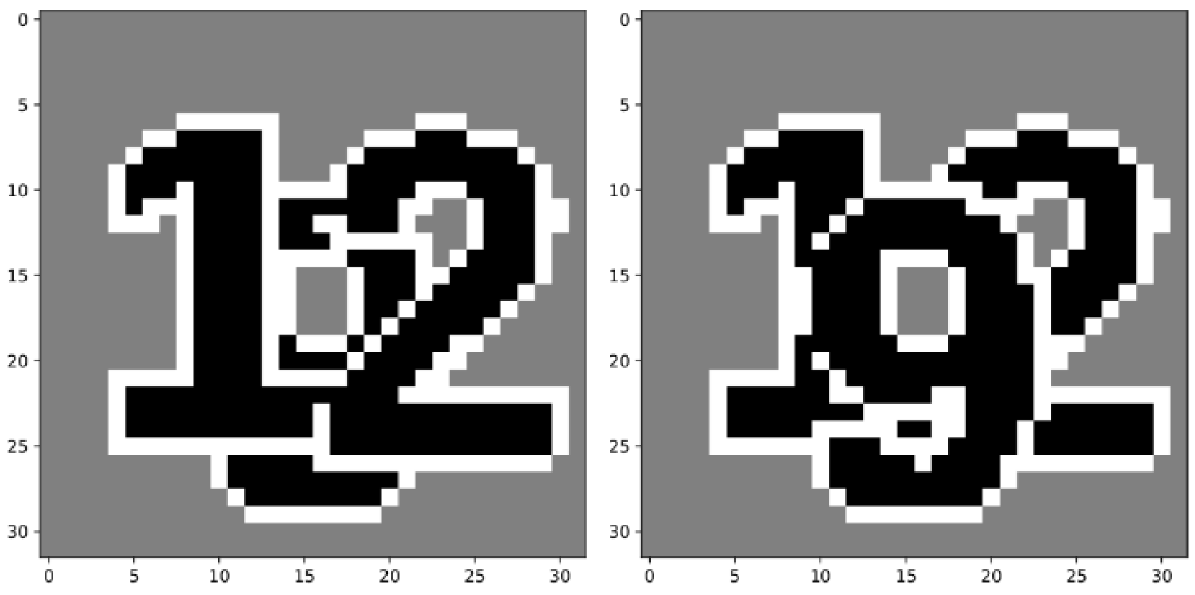

Figure 3.16: Input and its Expected target output

I have only tested the BLT-BLT autoencoder on border3 data set, wherein there are three digits in one image. This is because, BLT-BLT autoencoder is the best performing autoencoder among all models and I would like to see if how it performs in this task. Here the model has to understand that there are three digits and when inverting the order of the digits, the front digit has to become the third, the last one has to become the first digit and the middle digits has to remain at the same place. This task is more challenging. As we are not improving the model's capacity, it would be too ambitious to expect a performance as good as what we achieved for the two digit case.
In order to compare with the two digit case, I have maintained the number of time steps at four and the code layer size at 24. The model was trained on 100000 training examples and tested on 1000 images. I have shown a few reconstruction outputs in Figure 3.17.

In the first two images, we can see that the model identified three digits in the input and also has inverted the order. This ascertains that the model is able to generalize to higher number of digits in the input as well. But, we can also see that in the last but one image, the model fails to get the right identity of digit '9'. In the last image, it completely fails to identify any digit.

It can be seen that the model is not doing as well as in the two digit occlusion case.This can be because the model capacity is not enough to do the task. The model capacity can be increased by either increasing the number of code layer units or by increasing the number of time steps or both. I have plotted the learning curve, as shown in Figure 3.18.
I have done a comparative analysis to visualize the changes in the performance by varying the code size and the number of time steps. I have taken the example in Figure (where all the

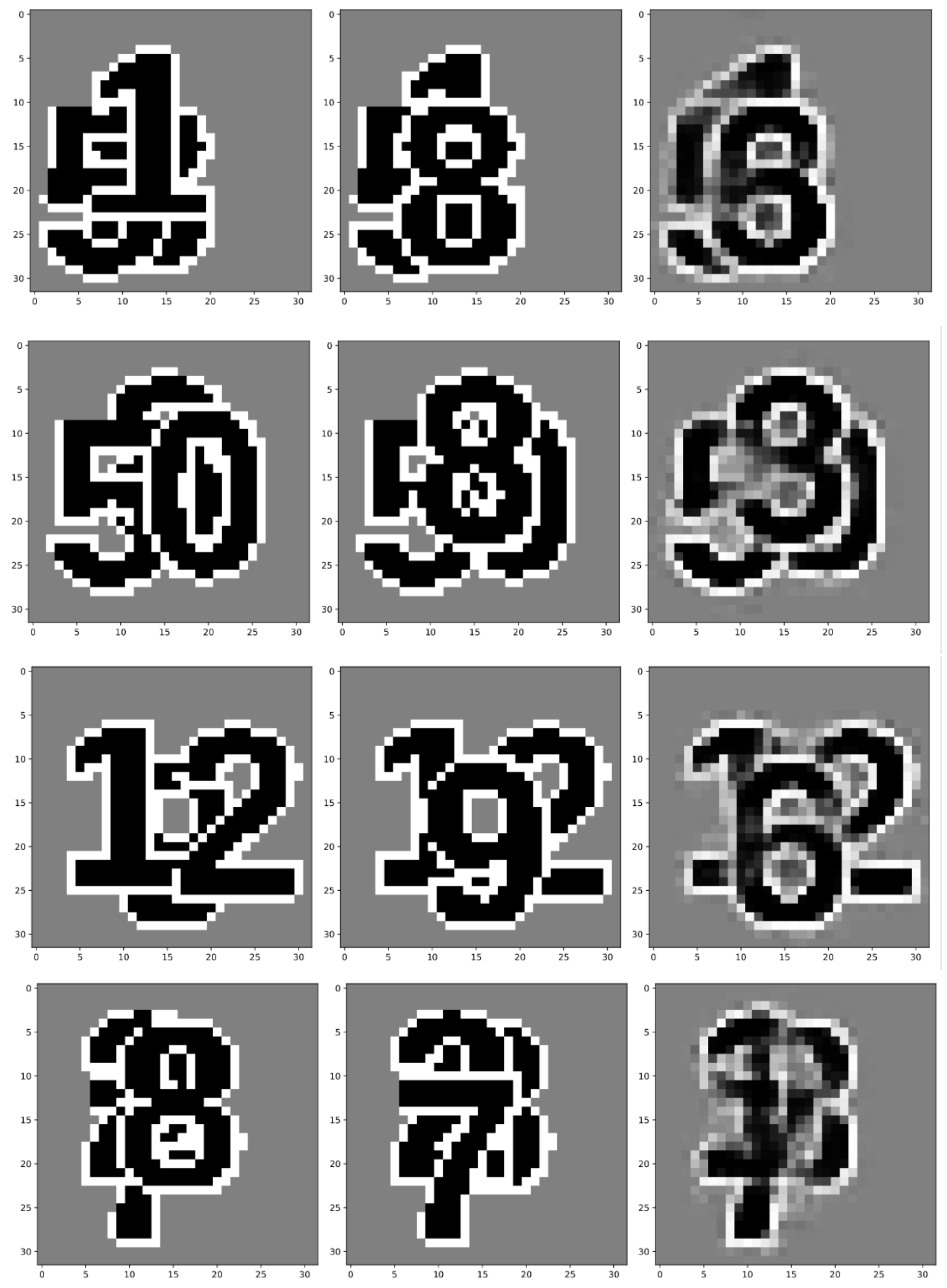

FIGURE 3.17: Input, Expected Output, Reconstruction Output

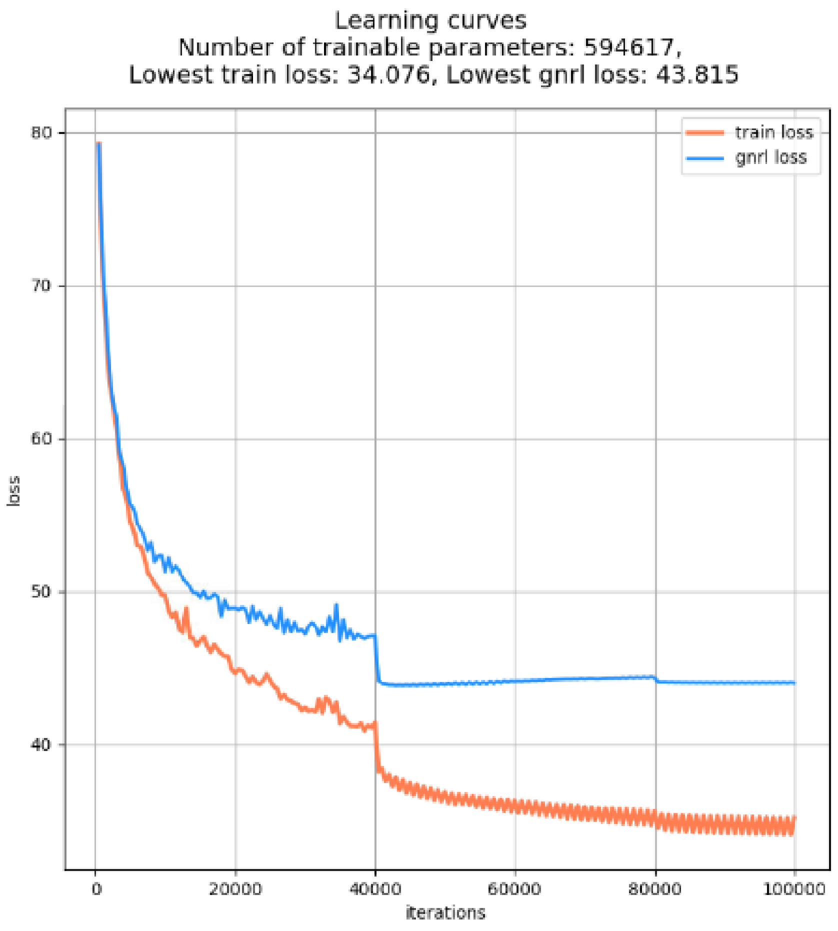

FIGURE 3.18: Learning Curve for BLT-BLT autoencoder : border3

three digits are visible and previously shown that the model failed to reconstruct) and shown the reconstruction outputs across different model parameters. I have run the model for values number of time steps (nrep) = 2,4,8 and code layer sizes z = 18,24,32. I have shown the results in Figure 3.19.

I have plotted the train-generalization loss against different code layer sizes and different number of time steps in Figure 3.20 and Figure 3.21 respectively.

It can be observed that the error does decrease by improving the capacity of the model.

3.1.5 Can a shallow network do the task?

Since recurrent connections don't seem to be important for occlusions as such in the deeper network, we want to see whether this is because those deeper networks have so much computational capacity that adding recurrence doesn't do much.

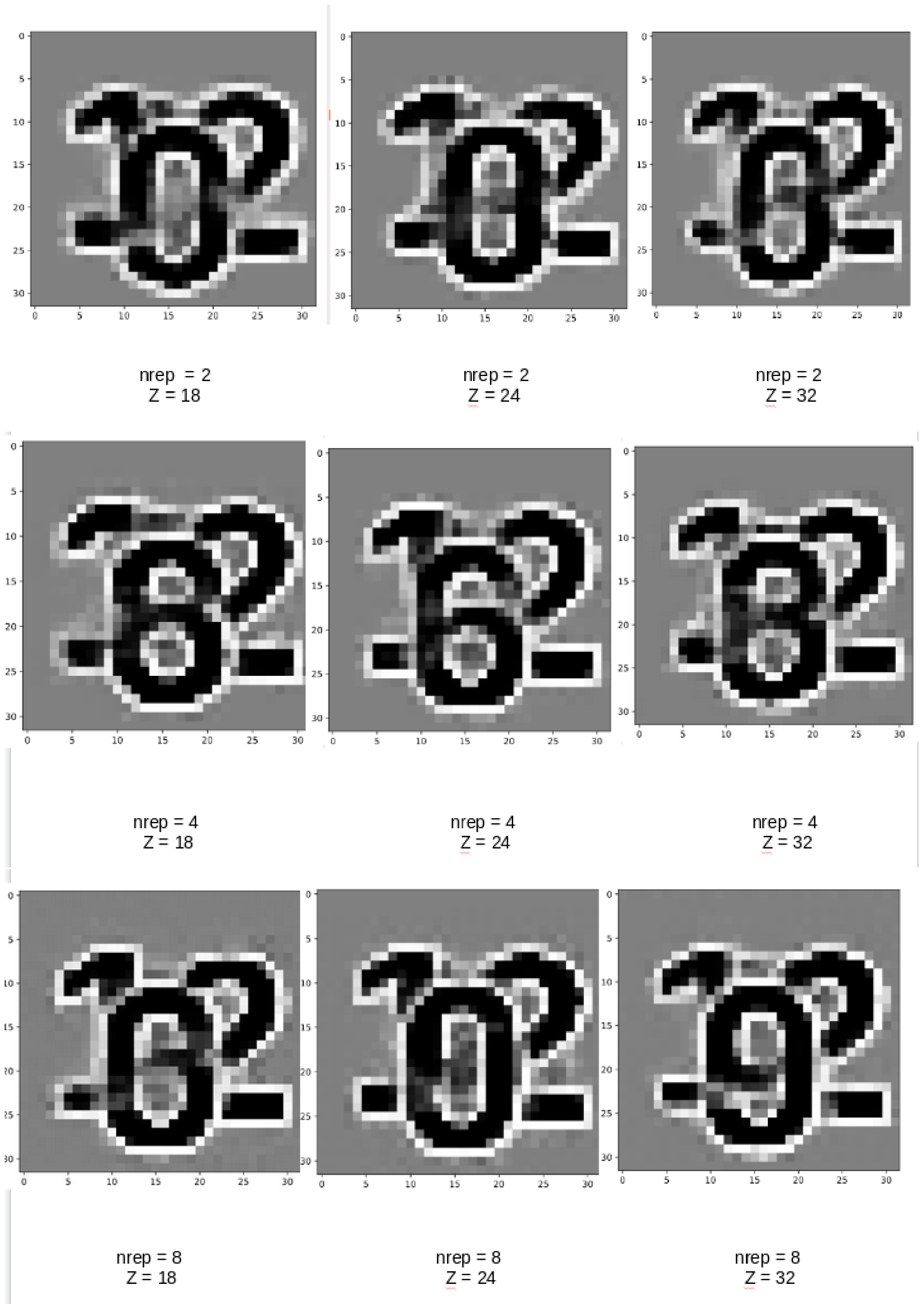

FIGURE 3.19: Comparisons

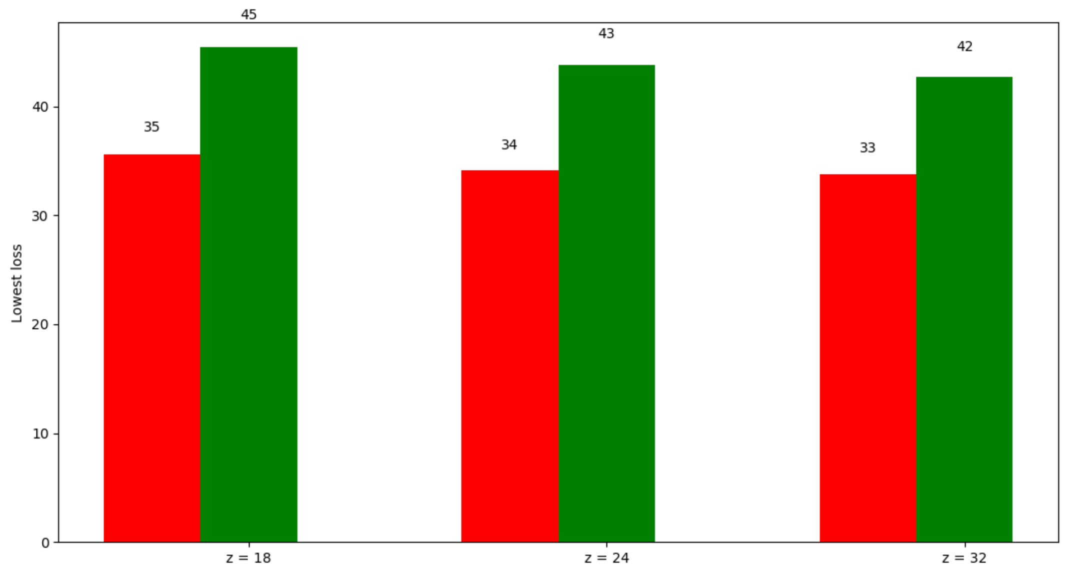

FIGURE 3.20: Here, the number of time steps is fixed at nrep = 4. The red bars show the train loss and the green bars show the generalization loss

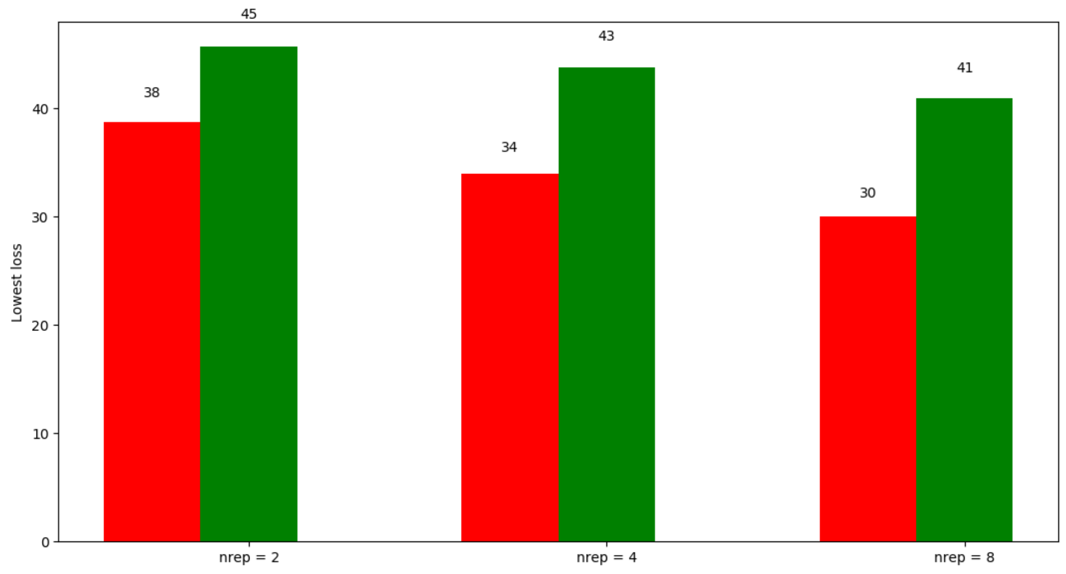

FIGURE 3.21: Here, the code layer is fixed at z = 24. The red bars show the train loss and the green bars show the generalization loss

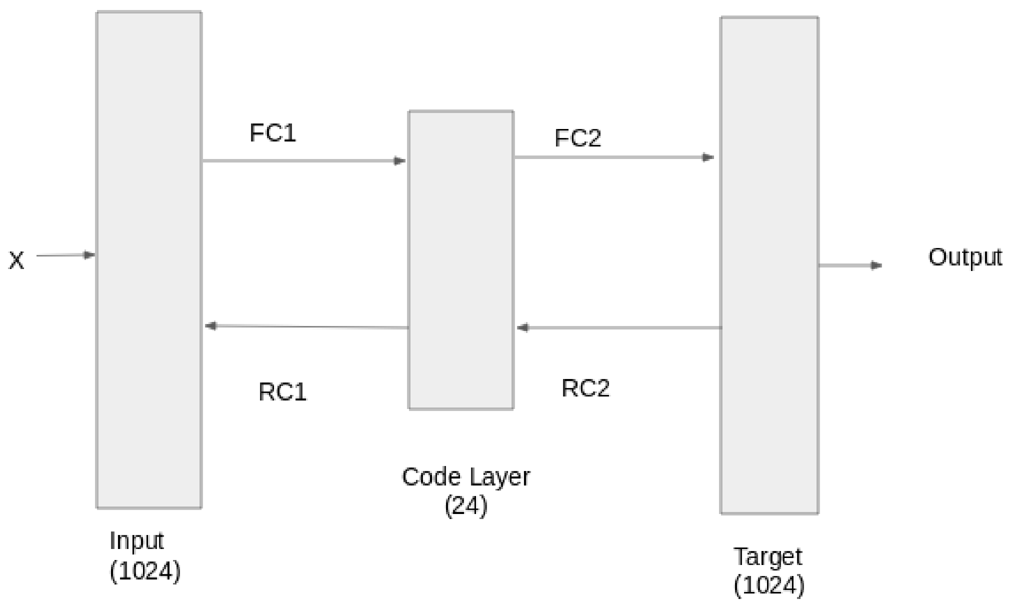

Figure 3.22: The shallow autoencoder

This model has an encoder and a decoder with one feedforward connection (B) and one feed back connection (T) each. There is no non-linearity introduced in the model.I have kept the number of time steps as 4 and the code layer size as 24. The image is first flattened into 1024 values and then fed into the model. The encoder compresses the information into 24 units. This is then converted back to 1024 output which is then reconstructed into an image. Figure 3.22 shows the model diagrammatically.

This model is trained and tested on the border2 data. As the model is too simple and does not have convolutional layers, I keep the position of digits in the data constant. This means, digits occur at the same position throughout the data.
This is a very simple network but surprisingly, it performs well on border2 data, where the position of the digits are same throughout. Figure 3.23 shows the reconstruction outputs.

This implies that a linear mapping from image to code layer is enough to reconstruct input image where the digits are occluding each other and the digits are in same position.
In order to see if the shallow model is identifying the digits and inverting them, I ran the model on the border2 data and tested on data with the grey bar as before. I have shown the reconstruction outputs in Figure 3.24. It can be seen that the grey bar is not reflected in the output but now the model is no longer able to decipher the depth order from the input image though it is able to identify the identity of the digits. One reason could be that, the shallow autoencoder identifies the depth of the digits depending on which digit is fully visible and which

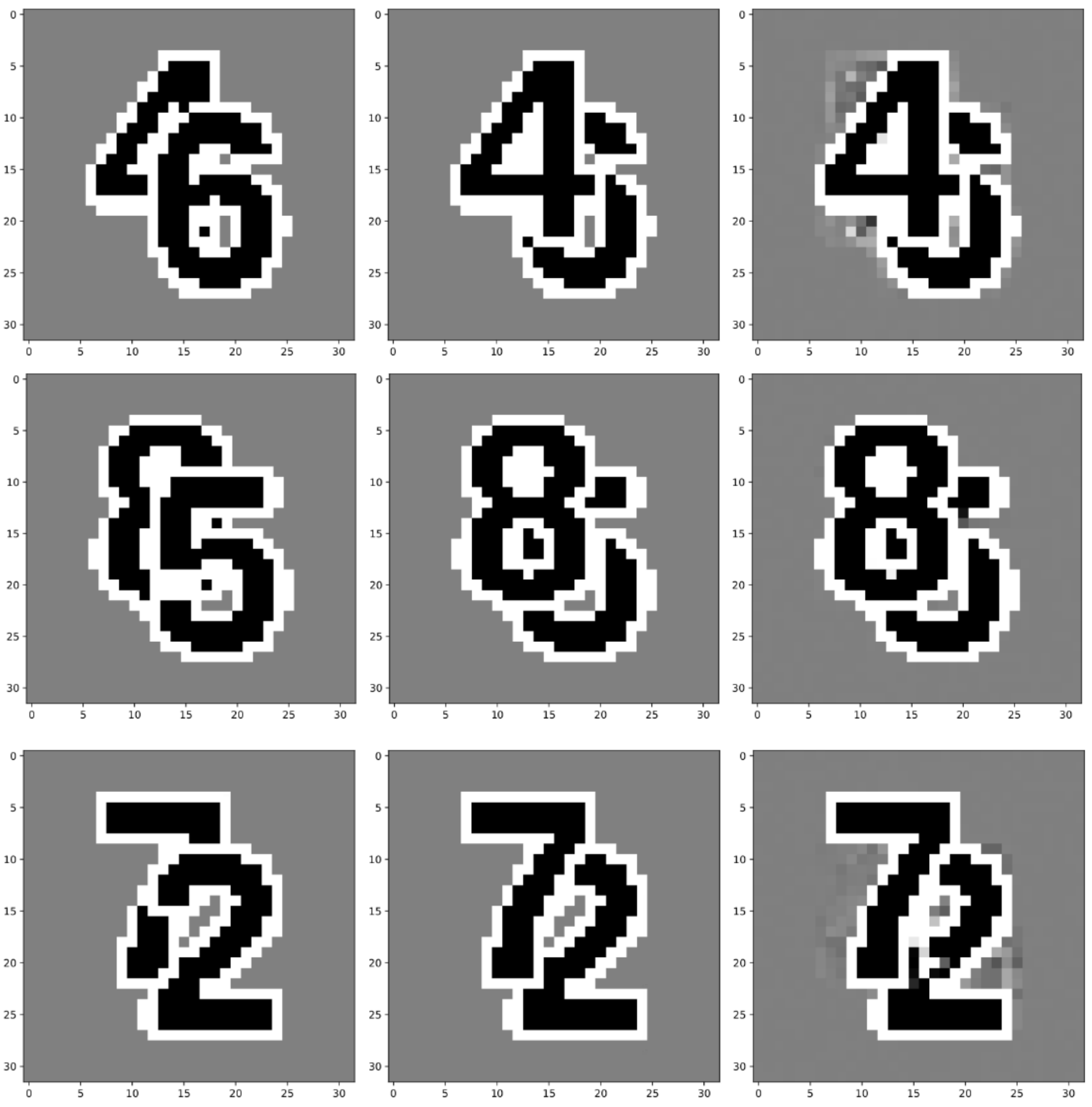
FIGURE 3.23: Input, Expected target output and the Reconstructed ouput

is not. With the grey bar in place, there is no longer a digit that is fully visible, hence it is not able to understand the depth from the input image.

As the size of the bottleneck layer determines how well the model can compress the input information, similar to previous cases, I have compared the performance of the shallow autoencoder with code layer sizes 10, 18 and 24. Figure 3.25 shows the reconstruction errors and Figure 3.26 shows how the reconstruction output varies when I vary the code layer size. We can observe that increasing the code layer size drastically improves the performance of the model.

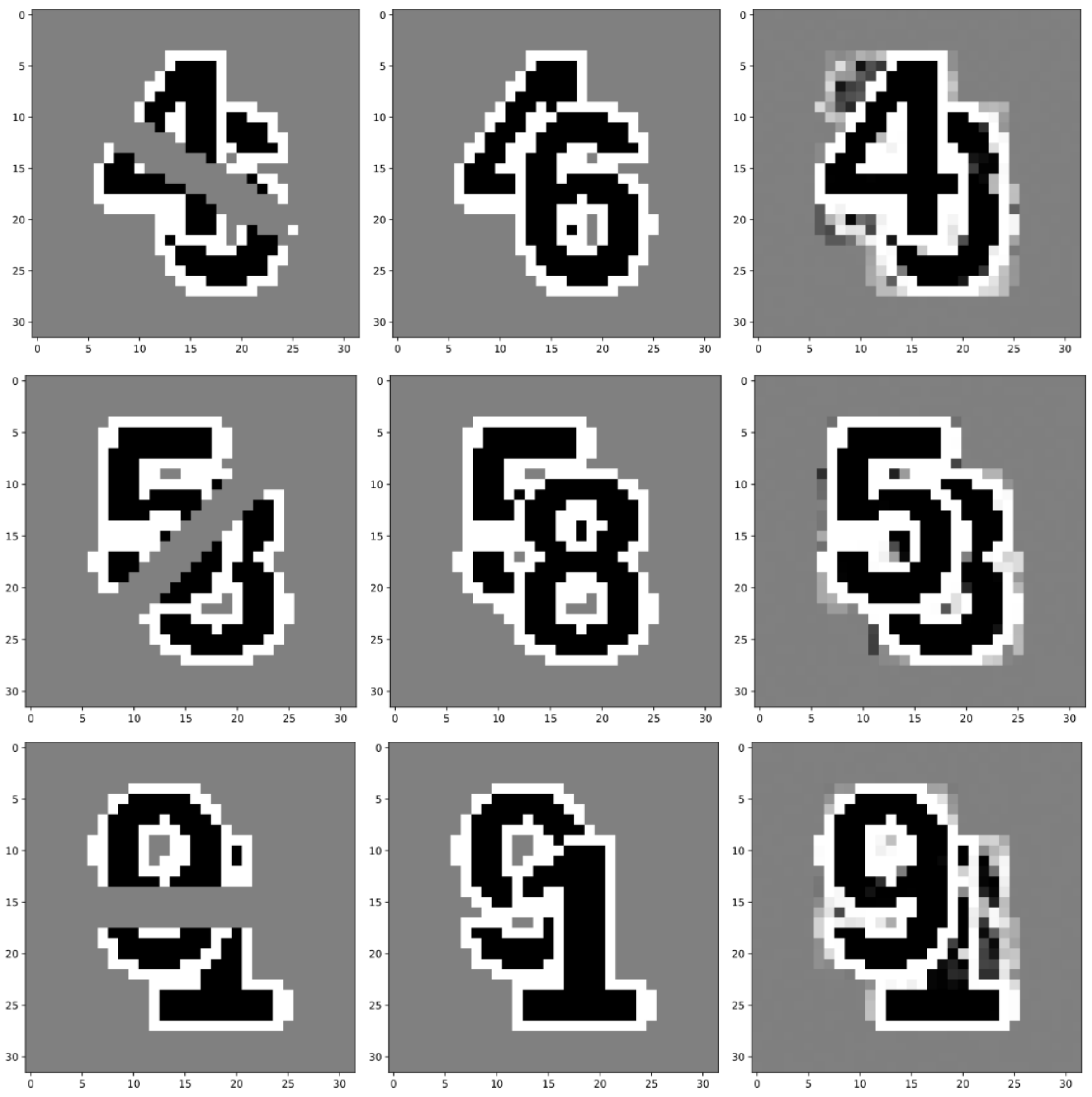

Figure 3.24: Input, Expected target output and the Reconstructed ouput

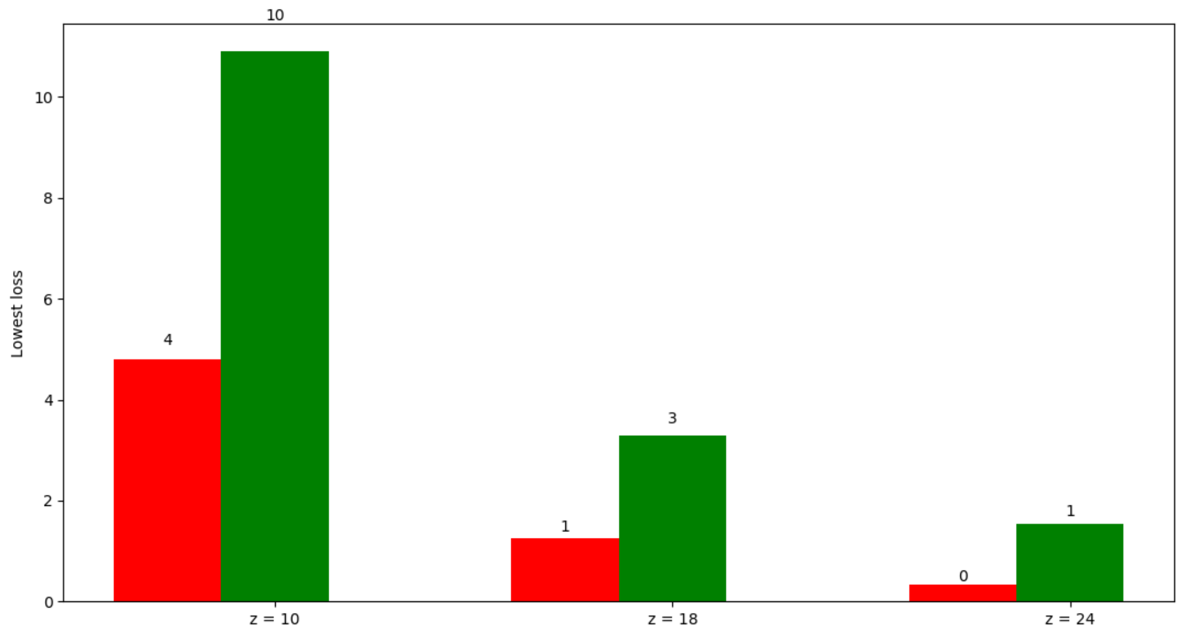

FIGURE 3.25: Comparison Bar Plot : code layer size is varied from 10 to 24, train loss is shown in red and generalization loss is shown in green

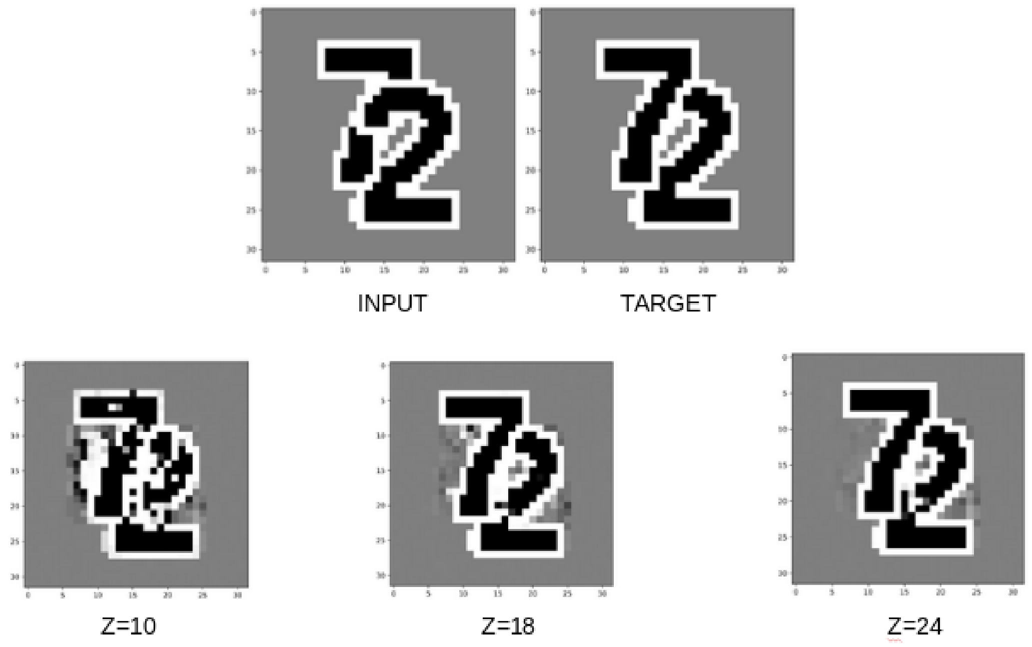

FIGURE 3.26: Input, Expected target output and the Reconstructed ouput

Chapter 4

Discussion

The entire visual pathway of human brain is shown to have recurrent neuronal dynamics. Currently, feedforward networks are ruling the object recognition domain due its exemplar performance in object detection tasks. By introducing lateral and top down connections along with feedforward connections, we hope to see an improvement in the performance of the model, and inch closer towards a more biologically plausible model.
The main aim of the thesis has been to understand the role of recurrent processing in the accomplishing the task. We have built an autoencoder with encoder and decoder being recurrent convolutional networks. We chose convolutional network as it is a parameter efficient method for large networks.
We added lateral as well as top down connections to the encoder and decoder in order to see if that improves the performance of the model.
It has already be shown by Kriegeskorte [2017] [6] that recurrent connections outperform feedforward models at object recognition in both presence and absence of occlusions. Riccardo, in his thesis has shown that recurrent connections improve the capacity of the model under supervised conditions.
I am trying to investigate how the recurrent connections help the autoencoder, which is an unsupervised learning technique. Till now, we have seen that identifying the digits and getting the order correct is not facilitated by these recurrent connections as they are achieved by the BB autoencoder as well. But, we can observe that the outputs of input images from autoencoders with recurrent connections are sharper and clearer whereas, BB architecture produces a blurry reconstruction. It is also seen that the recurrent connections in the decoder plays the significant role. This can be because, recurrent connections help in achieving sharp and crisp reconstructions of the input images.

I also tested the model on partially visible occluded images to see if the model is able to do the required task, which is to identify the digits and infer depth from the input image and it was observed that they are indeed able to do the task. It was also seen that the benefits of recurrent processing is not associated with occlusions but they work in non-occlusive cases also. Finally, it was seen that the task was acheived by a shallow autoencoder, which is pretty counter-intuitive.

4.1 Related Work

This entire project idea has drawn its inspiration from the work of Spoerer, McClure, and Kriegeskorte [2017] [6] who have thoroughly investigated the role of recurrent processing in recognising images of fully and partly drawn digits with occlusion.
This thesis work is a continuation of the work done by Riccardo, wherein he tried to understand the role of recurrent connections in inference and graphics tasks. He had concluded that BLT encoders and BLT decoders perform best at inference and graphics tasks respectively.I have instead taken up the unsupervised task, wherein the autoencoder does the inference and graphics tasks by itself. I have investigated further and have included performance of the different combinations of encoder and decoder architectures. I have also tested the model's performance on a variety of datasets and stated important observations.

4.2 Future Work

In order to analyse the model further, we can go deeper into the code layer. Currently, we are not clear how the autoencoder is encoding the input image. We should compare the code layer from both the deep and shallow networks to understand more about the results that we have got till now. How the shallow autoencoders are able to do the task is a major question.
Next step shall be trying to figure out the structure of the latent space of the shallow and deep architectures discussed in the thesis. One way to do that shall be to manipulate the code layer and see how the image changes. For an input image, the code layer size is 24 (standard throughout the thesis). We can change one or two values in this 24 valued vector and reconstruct the image from the manipulated code layer. This will allow us to find how the autoencoder encodes identity, depth, position and so on.
Another way shall be to manipulate the image and see how the representation changes. For example, we can train the network as we did previously. Then we can create another training set wherein one particular digit is present as front digit always. Let the network be trained on that. We can then compare two code layers from each of the scenarios. This allows us to understand if anything is present in the code layer specific to a digit.

Finally, our aim is to understand how brain is using the recurrent network for visual perception and produce a more biologically plausible architecture.

Appendix A

Code

The code for creating the dataset was adapted from Spoerer, McClure, and Kriegeskorte 2017 [6]. The networks were all built using PyTorch (Paszke et al. 2017[10]). The code is freely available on Github at : My-Thesis
Riccardo's thesis is available at : Riccardo-Thesis

Bibliography

[1] J. R. G. K. Hesheng Liu, Yigal Agam, *Timing, timing, timing: fast decoding of object information from intracranial field potentials in human visual cortex.* 2009.

[2] G. Hinton, *Taking inverse graphics seriously.* 2013.

[3] D. Z. DiCarlo, James J and N. C. Rust, *How does the brain solve visual object recognition? en. In: Neuron 73.3, pp. 415–434.* 2012.

[4] R. C. e. a. O'Reilly, *Recurrent Processing during Object Recognition. en. In: Front. Psychol. 4, p. 124.* 2013.

[5] S. I. Krizhevsky, A. and G. E. Hinton, *Imagenet classification with deep convolutional neural networks.* 2012.

[6] P. M. Courtney J. Spoerer and N. Kriegeskorte, *Recurrent convolutional neural networks: a better model of biological object recognition.* 2017.

[7] B. Y. Goodfellow, I. and A. Courville, *Convolutional networks. In Deep Learning,chapter 9, pages 330 – 372. MIT Press. http://www.deeplearningbook.org.* 2016.

[8] D. H. Hubel and T. N. Wiesel, *Receptive fields and functional architecture of monkey striate cortex. The Journal of physiology, 195(1):215–243.* 1968.

[9] N. Kriegeskorte, *Deep neural networks: a new framework for modeling biological vision and brain information processing. Annual Review of Vision Science, 1:417–446.* 2015.

[10] A. e. a. Paszke, *Automatic differentiation in PyTorch.* 2017.