UNIVERSITY OF CALIFORNIA SAN DIEGO

Better Cardiac Image Segmentation by Highly Recurrent Neural Networks

A thesis submitted in partial satisfaction of the
requirements for the degree of Master of Science

in

Computer Science

by

Jiaxin Li

Committee in charge:

Professor Garrison Cottrell, Chair
Ilkay Altintas
Julian McAuley

2020

The thesis of Jiaxin Li is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California San Diego

2020

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank Dr. Mai H. Nguyen of the San Diego Supercomputer Center, with whom I have been working for the past three years, who have afforded me countless opportunities to learn and grow my skills, and whose guidance and care helped me reach new heights in my career. I also had the pleasure to work under the supervision of Dr. Ilkay Altintas, who offered me the opportunity to work at San Diego Supercomputer Center and have since provided invaluable support. Dr. Nguyen and Dr. Altintas afforded me the chance to advance my studies and skills at a crucial time in my life, and without them I wouldn't be where I am today.

I have been fortunate to study under Professor Gary Cottrell, whose academic brilliance and ingenuity continues to inspire me, and whose caring counsel helped me set a firm foot into the academia. It was in Professor Cottrell's courses that I have truly discovered my passion for artificial neural networks. I was also given the chance to meet many brilliant minds when Professor Cottrell introduced me to his research group, where I made new friends and became more in love with my work. I thank Professor Cottrell for his kind words, his scrutinizing eyes, and his many nudges that made me both a better scholar and a better person.

I would like to thank Professor Julian McAuley, whose generosity afforded me support and guidance as I worked hard toward this thesis; I had the pleasure to take several of Professor McAuley's graduate courses, which furthered my love for big data and artificial intelligence. I would also like to thank Drew Linsley at Brown University, who have provided invaluable feedback and support while I studied and refined the computer models for this work, and who have kindly made available a large amount of essential code and data.

At last, I would like to thank my friends for supporting me and my work, and for being there for me especially when I need it the most. I would like to thank my parents, who have given me great freedom in exploring what I love the most, and supporting me both financially and as the greatest family ever. I would like to thank Amy, whose care and affection for the past nine years have continued to push me forward, and whose presence have given me faith that the future is, after all, prosperous and bright.

ABSTRACT OF THE THESIS

Better Cardiac Image Segmentation by Highly Recurrent Neural Networks

by

Jiaxin Li

Master of Science in Computer Science

University of California San Diego, 2020

Professor Garrison Cottrell, Chair

Cardiac magnetic resonance (CMR) image segmentation has been a crucial tool for medical professionals to diagnose cardiovascular diseases (CVDs), which are the leading causes of death throughout the world. Segmenting CMR images is very time consuming and increases the cost of CVD diagnoses and treatment, making them inaccessible to many. Automated CMR image segmentation models strive to lower the cost of CVD diagnosis, but such models must be efficient and accurate in such failure-sensitive domains as human medicine. This thesis proposes to apply γ-Net, a recurrent extension of the popular U-Net, to automatically perform high-quality CMR image segmentation. γ-Net is a recent development by Linsley et al. of Brown University, and has exhibited the ability to outperform U-Net on very small datasets, which is beneficial

given the very limited amount of patient CMR data available to the scientific community. γ-Net leverages biological principles backed by anatomical evidence as well as attention mechanisms in order to achieve its high efficiency.

In this thesis, we examine the following topics: (a) γ-Net's resilience to smaller training set sizes, which is cruicial when little patient data is available; (b) resilience to variation in training and validation data, which is shown to significantly degrade performance in state-of-the-art models; and (c) the ability to transfer to new datasets with minimal fine tuning, which saves training cost for practical applications. We have found that (a) γ-Net significantly outperforms an equivalent U-Net in validation performance when trained using a reduced training set; (b) γ-Net is much more resilient to input variations than U-Net; and (c) γ-Net generalizes to new datasets better than comparable U-Nets.

Chapter 1

Introduction

1.1 CMR Segmentation

In today's age of advanced medical technologies, cardiovascular diseases (CVDs) still remain the leading cause of death in the United States and around the world, beating cancer by a wide margin in number of deaths, according to the World Health Organization [1]. Notably, the worldwide deaths caused by CVD is almost double the number of that caused by cancer. With on-time diagnosis and proper care, it is possible to reduce the fatality rate of CVDs [1, 33]. Thus, improving the tools and techniques for identifying and analyzing CVD pathologies is paramount for furthering human medicine.

For decades, cardiovascular magnetic resonance (CMR) imaging, also known as cardiac MRI, has been one of the principle ways to diagnose CVDs, and to aid medical professionals in providing healthcare and treatments [1, 33]. Crucially, CMR images are often used by professionals to infer the Left Ventricle Ejection Fraction (LVEF), which is a measure of the heart's ability to circulate blood inside the body, and an important predictor of various major CVDs, such as myocardial infarction and dilated cardiomyopathy [23]. LVEF is parameterized by two values: the end diastole volume (EDV), which is the volume of the left ventricle (LV) when fully expanded; and the end systole volume (ESV), when the LV is fully contracted. However, inferring EDV and ESV usually requires medical professionals to manually segment CMR images, which is a very laborious process. Manual segmentation of the CMR data for

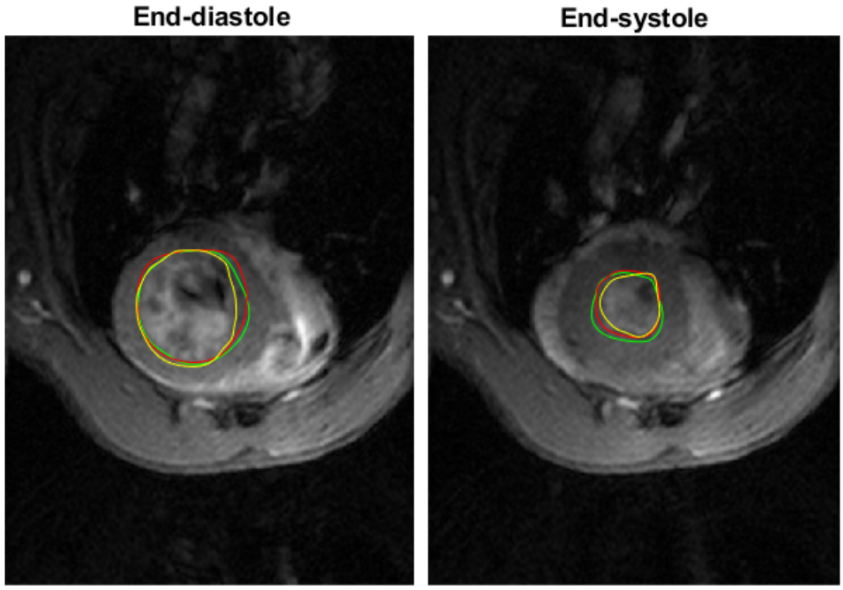

Figure 1.1. The CMR Segmentation Task And Inter-Observer Variability [32]

a single patient will take a professional around 30 minutes, making it infeasible to calculate LVEF for large numbers of patients. Furthermore, previous work has found that, due to the large variance in image quality and visibility of segmentation borders, manual segmentations of CMR images exhibit low reliability and high inter-observer variability (see Figure 1.1) [23].

To address these problems, we propose a robust and lightweight recurrent neural network (RNN) model to perform LV segmentation on horizontal (short-axis) CMR images. We will compare our model with feed forward fully-convolutional networks, which are currently the state-of-the-art for image segmentation tasks. We will also demonstrate our model's interpretability by examining its convergence on a "best-effort" output through several time steps.

1.2 Neural Networks

Artificial neural networks (ANNs) are a class of machine learning models inspired by the brain, in which very large layers of specialized brain cells, called neurons, leverage the weighted connections between them to represent and process complex signals. It has been shown that artificial neural networks are capable to mathematically to approximate Borel-measureable

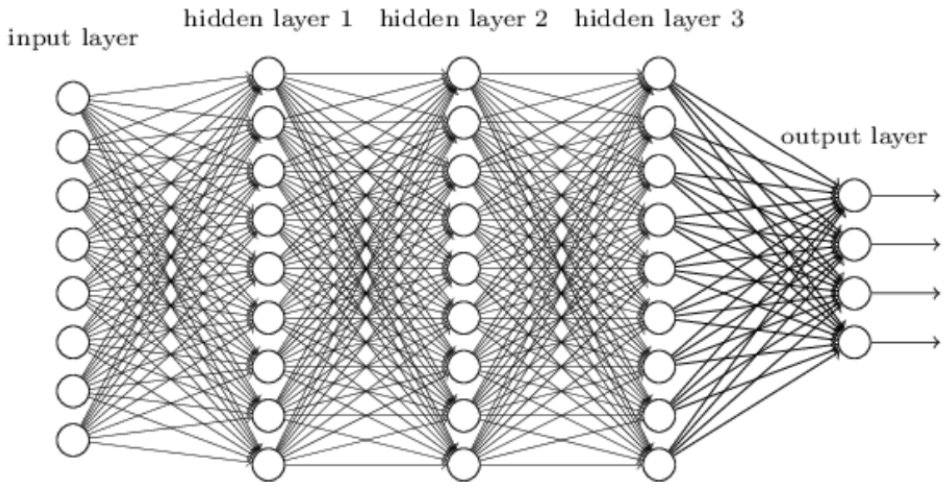

Figure 1.2. A Fully-Connected Neural Network [19]

functions to an arbitrary degree of accuracy, and optimize a wide range of traditionally difficult machine learning problems without requiring manual parameter selection or feature engineering.

Like many other machine learning models, neural networks require training data to learn the correct representations of the proposed problem, and make informed predictions. Unlike many other models, the inference process of neural networks is fully differentiable, and learning for the neural network can be simplified to backward error propagation through network layers via linear operations. Not only does this simplify model development and allow an unprecedented level of creativity for machine learning researchers, the learning process is also computationally efficient and highly parallelizeable, which makes it possible to construct and run very complex neural network models on limited hardware. This is ideal in both the scientific community, where computation resources are also limited; and also production environments, where it is desired to maximize computation efficiency for minimum cost.

1.2.1 Neural Network Basics

Neural networks achieve their impressive learning capacity by propagating input information through multiple layers of individual processing units, or neurons, wherein each neuron calculates a weighted sum from all its connections to neurons in the previous layer, then applies

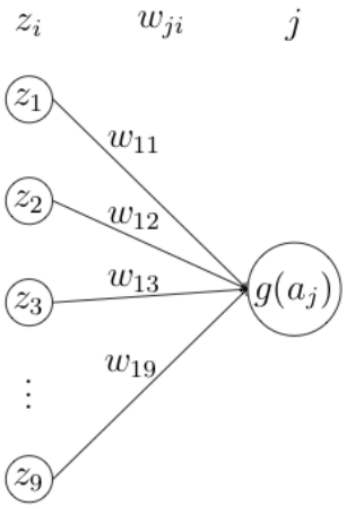

Figure 1.3. Interaction At A Single Neuron In A Fully-Connected Neural Network

a non-linear function to the sum to produce an output. This process is known as the forward pass, where the model accepts an input and returns its prediction.

The network's output is then compared against the known ground truth the output that we want the model to learn. The result of this comparison is used to obtain a loss metric, which represents the current performance of the neural network, or rather, a measure of the model's error. To minimize the loss, we need to calculate the gradient of the loss value with regard to every weight in every layer of the neural network. This is done via a process called back-propagation, where the loss is propagated iteratively back through the network. Apart from a few, relatively unpopular neural network models such as the Spiking Neural Network, all modern neural networks are trained via back propagation.

Back-propagation starts by calculating the loss with regard to the output of each individual neuron in the output layer. Using the chain rule, we then calculate the loss gradient with regard to the internal weights and current activity at each neuron, and calculate the gradient with regard to both the inputs and the weights of the final layer. The gradient with regard to the layer inputs is our back propagated loss.

After we have propagated loss through the output layer, we can proceed to calculate the gradient at each neuron in the layer before the output layer in the same manner, and repeat for

the next layer, until we have calculated the loss gradients for neurons in the input later. Once we have calculated the gradients with regard to every parameter in the network, we can update the parameters using gradient descent algorithms such as RMSProp and Adam.

So far, we have illustrated neural networks which are fully connected, where each neuron accepts connections from all neurons in the previous layer. This is, however, not the optimal neural network architecture for most tasks, which require specific topological constraints in the network to learn more efficiently. We will illustrate neural networks that are more optimized to specific tasks below.

1.2.2 Convolutional Neural Networks

In deep learning for image-related tasks, Convolutional neural networks, or CNNs, have been found to be wildly successful. CNNs apply a form of spatial colocation constraint to the general neural network, by forcing network layers to only learn a small "kernel" with which to convolve on comparatively large images. These kernels are also known as "features," which intuitively correspond to their role in CNN layers: generating activation maps over the input image that correspond to specific local image features. In other words, a pixel value in an activation map represents the similarity between the corresponding kernel and the local patch of image centered at said pixel. Similar mechanics have been found to exist within early human vision pathways.

At a high level, convolutional neural networks model a hierarchical information processing framework, where each layer or functional block in the network represents one more level of abstraction from its input; the farther the layer is from the network input, the more abstract the information is in the layer. For example, the first convolutional layer in a CNN often learns features that match to object edges or corners, while the last layer in the same CNN may learn features that will match to the picture of a cat's ear or a dog's muzzle. To facilitate this learning, it's common for CNNs to utilize pooling layers, which aggregate activation with regard to spatial patches in the input activity, in order to decrease the spatial dimension of the

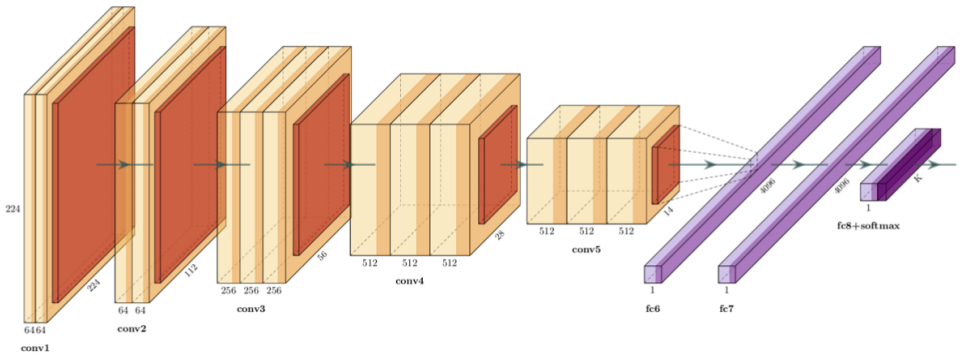

Figure 1.4. Example Convolutional Neural Network (VGG 16 [28])

activity maps; CNNs also tend to increase the number of features learned in their later layers, as features become increasingly abstract and global farther down the network. The pooling layers (in particular, max-pooling layers) also allow the CNN to exhibit translation invariance to an extent, although recent studies have shown that this is not entirely true [34].

The receptive field (RF) is an important concept with regard to CNNs. In neuroscience, the receptive field of a certain neuron represents the set of neurons at the bottom layer of the network, whose excitation will result in the excitation of said neuron. In a convolutional neural network, the receptive field of a particular output unit is the set of input units, from which there is a path to said output unit in the computation graph. For a CNN to effectively learn global features, i.e. large features in the image that may span the entire image, the CNN must be deep enough such that the receptive field in every unit in the output layer is at least as large as the input image.

In recent years, a large number of image processing neural networks are constructed by stacking a large number of convolutional layers, achieving impressive processing depth, hence the invention of the term Deep Learning. Deep CNN networks have achieved superhuman performance in previously difficult problems such as image classification, and even in hard time-series problems such as speech modeling. Fully convolutional networks such as the FCN, Faster R-CNN and U-Net have been shown to perform excellently on image segmentation tasks. In this work, we will use state-of-the-art fully convolutional networks, specifically the U-Net, as

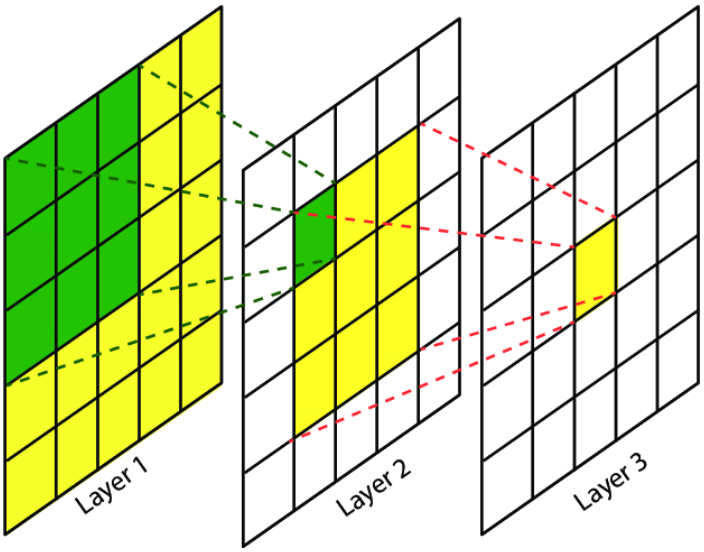

Figure 1.5. CNN Receptive Field [13]. The yellow regions are the receptive field of the single cell in layer 3, and the green regions are the receptive field of the single cell in layer 2.

the baseline model to compare to.

1.2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of artificial neural networks where neurons may receive input from its own layer at previous timesteps, and not necessarily from layers before it. This allows self- and within-layer connections, and introduces persistent activity states to the network model, enabling the network to become stateful, i.e. able to "remember" its own processing history, unlike the stateless feed-forward networks. Stateful models are often found to perform well in time series modeling, where the output of the model depends not only on the current input, but also on inputs from the previous timestep and beyond. RNN models such as the gated recurrent unit (GRU) and the long-short term memory (LSTM) architectures have been shown to excel at time-series modeling, and are able to achieve excellent performance in tasks such as simple language modeling and music generation [20].

Another use case of recurrent networks, aside from time-series modeling, is increasing effective receptive field size with minimal increase in the number of network parameters. This is especially useful in convolutional neural networks, where receptive field sizes of output units

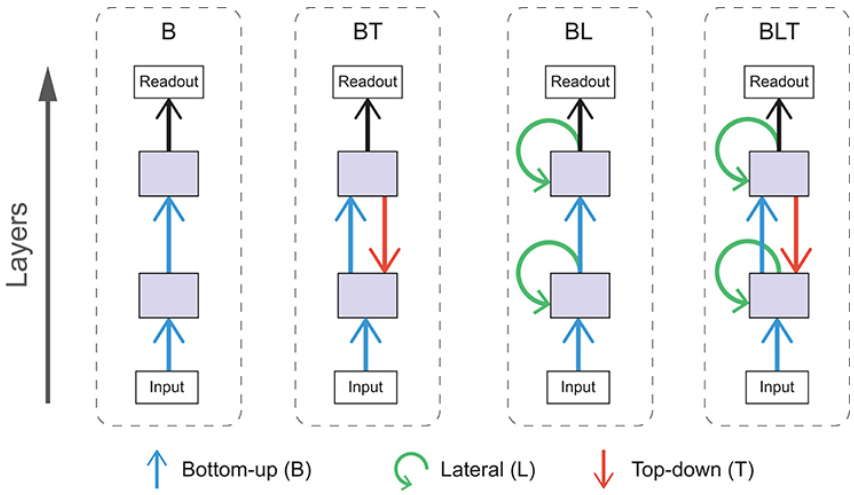

Figure 1.6. Different RNN Connectivity Schemes [30]. Shown are the three major connectivity schemes in recurrent networks: bottom-up (feed-forward), lateral (horizontal), and top-down.

are constrained by network depth, filter sizes and spatial pooling. In feed forward CNNs, large receptive fields are only achievable by increasing the size of convolution filters in each layer or the number of convolutional layers in the network, both of which significantly increase the number of parameters in the network. In contrast, recurrent convolutional networks are able to reuse the convolution filters in the same layer by passing its own output back as its input, thus eliminating the need to make the filters larger or increase the number of layers. In addition, such recurrent connections can enable lateral, or horizontal contextual information processing, which is supported by anatomical evidence [31, 26]. It has been shown that recurrence in convolutional neural networks significantly increases the effective depth of the recurrent layers, thus enabling efficient learning of global, context-rich features [15].

Compared to most feed-forward neural networks, modern recurrent neural network architectures are often established on the basis biological knowledge, building on discoveries from fields such as neuroscience, psychology and cognitive science. As a result, these networks can exhibit behaviors similar to those found in the human brain [14].

Because the weights of recurrent neural networks depend not only on its input data, but also on its own persistent activity states during the forward pass, a special back-propagation

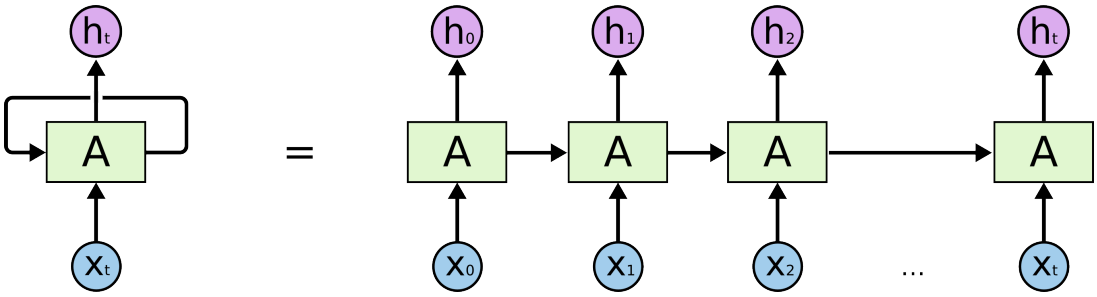

Figure 1.7. Unrolling A Recurrent Neural Network [20]

method called back-propagation through time (BPTT) must be used to train them. In back propagation, the recurrent network is ”unrolled,” such that network weights and activites at each time step are treated separately, and the computation graph becomes a direct acyclic graph. Back-propagation is then performed normally, taking care to calculate loss gradients with regard to the same model weights at each time step. After back propagation is complete, the gradients for the same weights at different time steps are aggregated into single values, and the gradient optimizer updates the weight.

Due to the need to unroll the network through many time steps during the training process, training recurrent neural networks can become prohibitively expensive in terms of processing and storage costs, so special care must be taken to make efficient use of network parameters. This is especially the case with large recurrent networks.

Chapter 2

Related Works

2.1 Image Segmentation

Image segmentation is the task of producing pixel-wise labels of an image in one of several possible categories, segmenting the image into groups of pixels that represent different regions of interest. In recent years, the image segmentation task is becoming incredibly important with the rise of advanced computer vision, finding applications from automated driving to large-scale medical imaging processing.

Early attempts at image segmentation using neural networks often involved using sliding windows to classify image patches centered at each pixel, thus producing labels for each individual pixel [7]. This process is not only incredibly expensive, as each window must be individually classified, but it also faces the dilemma of optimizing for either spatial resolution versus contextual information: small windows offer good spatial resolution but little spatial context, while large windows provide more context but poor resolution.

More recent approaches to the image segmentation problem often rely on fully convolutional networks, which do away with many of the above mentioned tradeoffs.

2.1.1 Fully Convolutional Networks

It has been argued that fully convolutional networks (FCNs) are the most important development with regard to the image segmentation task. Unlike the above mentioned methods,

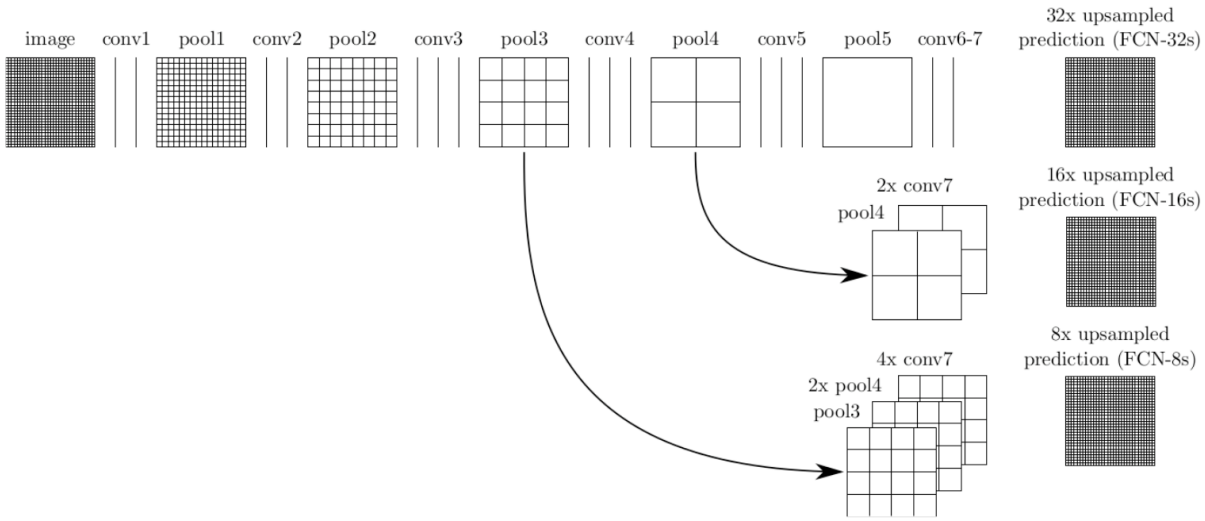

Figure 2.1. FCN Architecture proposed by Long et al. [17]

FCNs are capable of performing a single-pass computation on the image and produce pixel-wise labels, instead of iterating over a large number of image patches. Relying purely on convolution operations and foregoing all fully-connected layers, FCNs directly compute a non-linear mapping in the original image space, which not only significantly accelerates the segmentation process, but are also generalizable to arbitrary image sizes, owing to FCN's fully convolutional architecture [17].

Crucially, early FCNs such as that proposed by Long et al. produce segmentations by directly combining highly abstract feature maps in the downsampling path, as shown in 2.1, thus collapsing high-level abstract features with low-level, concrete features without additional processing. We note that there is still a conflict between spatial resolution and and contextual information in this early fully convolutional network, depending on the design of upsampling paths [32].

2.1.2 DenseNet

Another notable fully convolutional network is the Densely Connected Convolutional Network (DenseNet), proposed by Huang et al in 2017 [10]. A major departure from the early FCNs, the DenseNet is formed by a mirrored series of downsampling and upsampling blocks.

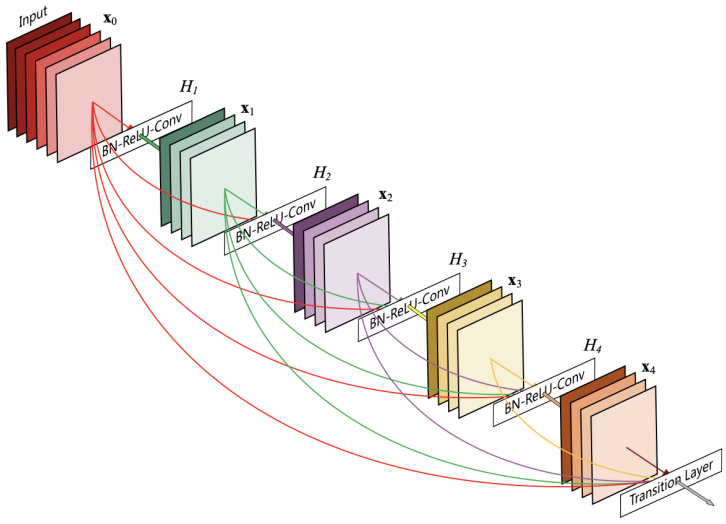

Figure 2.2. DenseNet Architecture, 5 functional blocks with all connections shown. [10]

Each functional block is composed of a series of convolutional layers and an up- or down-sampling layer, and receives input from all blocks preceding it via feature map rescaling and concatenation.

While originally applied to image classification, the DenseNet architecture can be easily adapted for image segmentation, as studied by Zhu et al [35] and Krešo et al [11]. Compared to early FCNs, DenseNet integrates feature maps at different levels of abstraction at every processing block via feature map concatenation, and is shown to produce segmentations that are both more detailed and more contextually accurate. Krešo et al. noted, however, that DenseNet is difficult to train due to overfitting in its upsampling blocks, likely due to the densely connected nature of the network. It has been shown that variety in the training data composition is crucial to obtaining better performance in DenseNets [11].

2.1.3 U-Net

U-Nets, named after their topology, are a class of Fully Convolutional Networks utilizing both a symmetric, bottlenecked auto-encoder architecture, and skip connections between layers of the same downsampling depth in the network [27]. This is in contrast to previous FCNs, which have much shorter upsampling paths and no skip connections to ground the upsampled outputs.

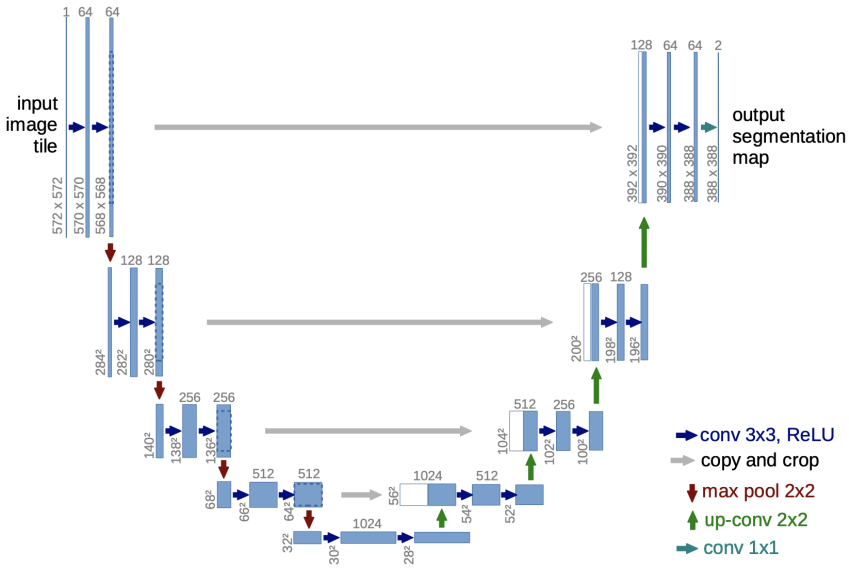

Figure 2.3. U-Net Architecture, 5 depth levels. [27]

Similar to the later DenseNet, U-Nets combine feature maps from the upsampling path and skip connections via feature map concatenation. U-Nets are known to achieve high segmentation performance using less training data than previous fully convolutional networks.

A unique feature of the U-Net is the two diverging processing pathways, which focus on contextual information and spatial resolution, respectively. The downsampling-upsampling path, the part of the network shaped like a ”U,” allows the network to process spatial relationships and global context, the extent of which depends on the network’s depth. At the same time, the skip connections at each depth level provides local spatial information which is integrated with the upsampled feature maps. By recombining information from the diverged pathways, U-Nets are able to produce both spatially detailed and context-aware segmentations, thus averting the context-vs-localization issues that plagued earlier models.

2.1.4 γ-Net

The γ-Net, developed by Linsley et al., is a recent recurrent extension to the popular U-Net, which employs recurrent horizontal and top-down processing pathways, as well as self-attention. This allows γ-Net to converge through several time steps onto an optimal output for

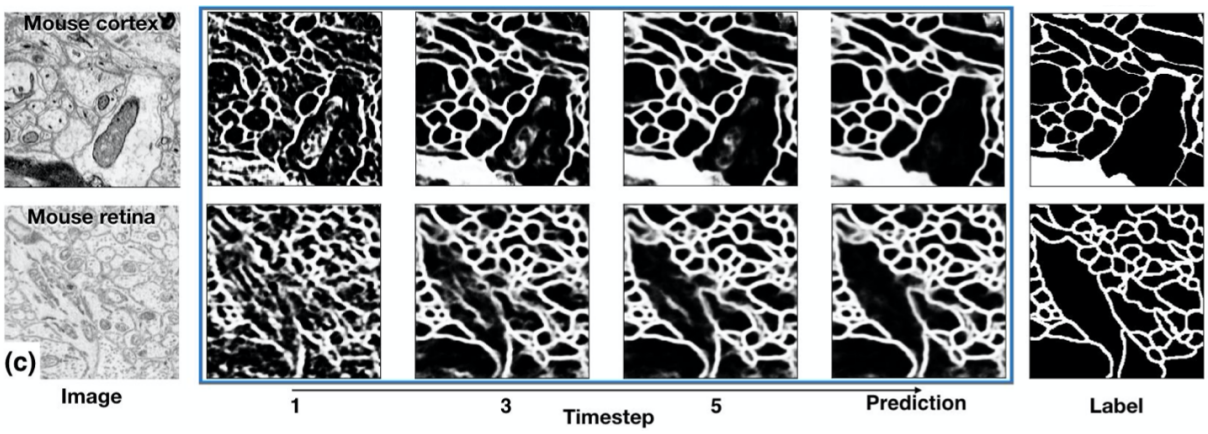

Figure 2.4. γ-Net converging to a "best-effort" output through several timesteps. [14]

each input image, and requiring much less training data than comparable feed-forward models to reach the same performance. We explain the architecture of the γ-Net in detail below.

Horizontal Gated Recurrent Unit

The Horizontal Gated Recurrent Unit (hGRU) is γ-Net's most important addition to the standard U-Net [14, 15]. The hGRU emulates biological neural networks, specifically connections in the mammalian visual cortex, which are known to exhibit a range of contextual phenomena. By using a gated, recurrent convolutional architecture, the hGRU allows information to "spread" horizontally in its activity map for several time steps, which gives the network the ability to learn long-range spatial dependencies.

In the traditional Gated Recurrent Unit (GRU), a "update" gate and a "forget" gate are implemented as part of the recurrent processing, and these gates have been shown to be critical for maintaining information over long time intervals. Notably, the hGRU explicitly separates inhibitory and excitatory connections, which is commonplace in biological neural networks. This explicit separation adds additional constraints to the hGRU architectre, allowing it to learn more effectively. Linsley et al. have demonstrated that this inhibition-excitation implementation allows the network to emulate the effect of lateral inhibition, which is an important phenomenon in early human vision that enhances edge detection and increases feature salience.

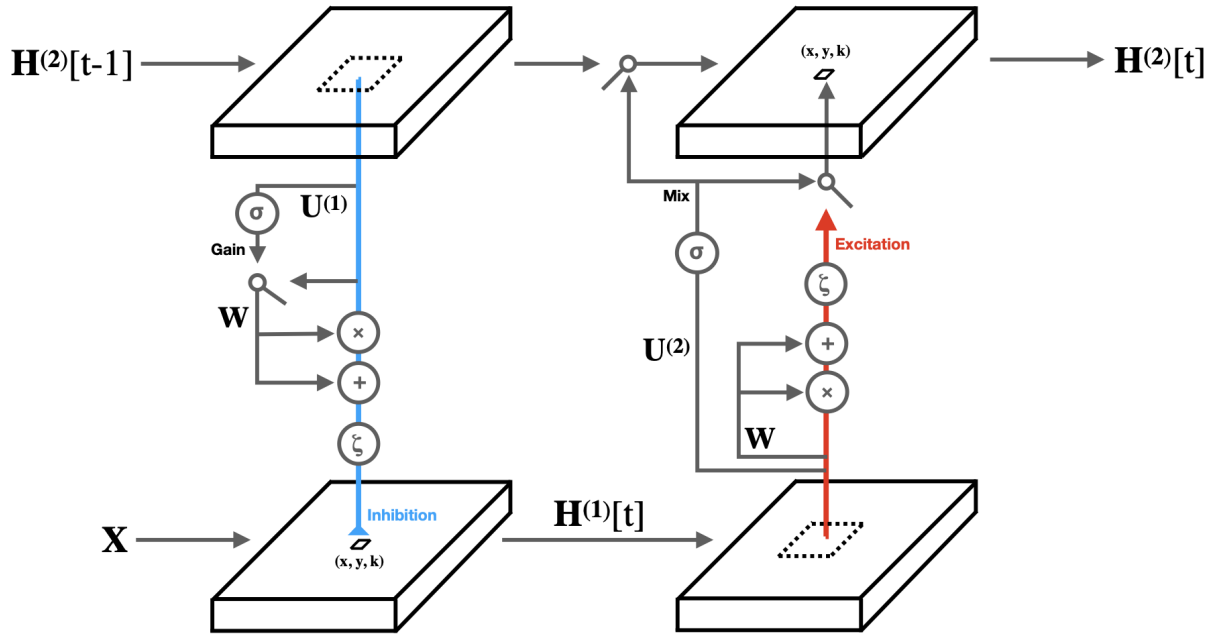

Figure 2.5. hGRU circuit. The hGRU explicitly separates the inhibitory and excitatory stages of the recurrent processing, constraining the model further and increasing training effectiveness. [15]

The hGRU was inspired by the recurrent neural model of contextual interactions originally developed by Mély et al [18]. The original neurological model can be expressed as the following differential equations:

$$\eta \dot{H}^{(1)}_{xyk} + \varepsilon^2 H^{(1)}_{xyk} = [\xi X_{xyk} - (\alpha H^{(1)}_{xyk} + \mu) C^{(1)}_{xyk}]_+ \tag{2.1}$$

$$\tau \dot{H}^{(2)}_{xyk} + \sigma^2 H^{(2)}_{xyk} = [\gamma C^{(2)}_{xyk}]_+ \tag{2.2}$$

where

$$C^{(1)}_{xyk} = (\mathbf{W}^I * \mathbf{H}^{(2)})_{xyk}$$

$$C^{(2)}_{xyk} = (\mathbf{W}^E * \mathbf{H}^{(1)})_{xyk}$$

Here, $X \in \mathbb{R}^{W \times H \times K}$ is the forward drive which may come from earlier feed-forward convolutional layers, $H^{(1)} \in \mathbb{R}^{W \times H \times K}$ is a intermediate internal state known as "circuit input", and $H^{(2)} \in \mathbb{R}^{W \times H \times K}$ is the output activity of the hGRU. The $\mathbf{W}^I$ and $\mathbf{W}^E$ convolution kernels model

hypercolumn activity in early human vision, and have been found consistent with anatomical data. $C^{(1)}_{xyk}$ and $C^{(1)}_{xyk}$ are the inhibition and excitation activities calculated during each time step. η, τ, σ and ε are time constants.

From the above, Linsley et al derived the following difference equation in order to quantize the learning process for computer models, by applying Euler's method to the differential equations. Here we assume $\eta = \tau$ and $\sigma = \eta$, and set $\Delta t = \eta/\varepsilon^2$.

$$H^{(1)}_{xyk}[t] = \varepsilon^{-2}[\xi X_{xyk} - (\alpha H^{(1)}_{xyk}[t-1] + \mu)C^{(1)}_{xyk}[t]]_+ \tag{2.3}$$

$$H^{(2)}_{xyk}[t] = \varepsilon^{-2}[\gamma C^{(2)}_{xyk}[t]]_+ \tag{2.4}$$

where $\cdot[t]$ denotes the activities at the t-th time step. This gives us a set of usable operations for training the hGRU.

Linsley et al further improved the above model by: (1) adding learnable 1×1 convolutional gates at each time step, represented by $\mathbf{U}$; (2) asserting equality between inhibition and excitation kernels, i.e. $\mathbf{W} = \mathbf{W}^I = \mathbf{W}^E$; and (3) applying squashing non-linearity to intermediate activities.

To obtain the activities at each stage of the hGRU for one timestep, the following computations are performed. First we obtain the circuit input, $H^{(1)}[t]$ by the following:

$$\mathbf{G}^{(1)}[t] = \sigma(\mathbf{U}^{(1)} * \mathbf{H}^{(2)}[t-1] + \mathbf{b}^{(1)}) \tag{2.5}$$

$$C^{(1)}_{xyk}[t] = (\mathbf{W}^I * (\mathbf{G}^{(1)}[t] \odot H^{(2)}[t-1]))_{xyk} \tag{2.6}$$

$$H^{(1)}_{xyk}[t] = \xi\,(X_{xyk} - C^{(1)}_{xyk}[t](\alpha_k H^{(2)}_{xyk}[t-1] + \mu_k)) \tag{2.7}$$

where ξ is the non-linearity function, $\mathbf{U}^{(1)}$ is similar to the "forget" gate in GRU, $\mathbf{b}^{(1)}$ is the bias of the "forget" gate, α_k and μ_k are scalar parameters controlling the linear and quadratic components of the inhibition activity. Importantly, we use ReLU as the non-linearity function as opposed to the tanh function proposed in the original hGRU paper, as ReLU has been shown to

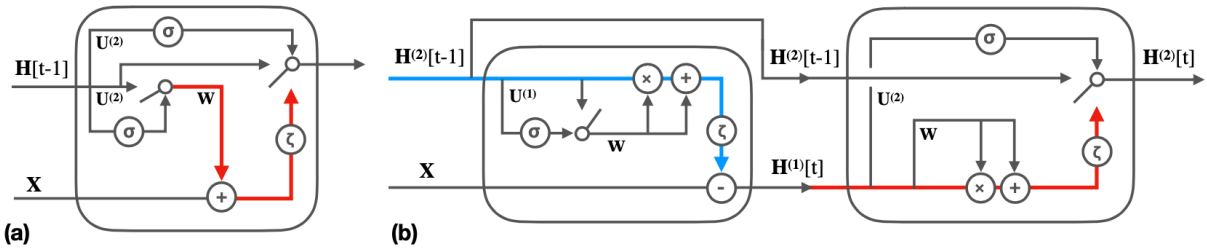

Figure 2.6. Comparison between GRU (a) and hGRU(b). The hGRU explicitly separates inhibitory and excitatory processing, further constraining the training process [15]

improve hGRU's learning efficiency [15].

The circuit output is computed as following:

$$G^{(2)}_{xyk}[t] = \sigma((\mathbf{U}^{(2)} * \mathbf{H}^{(1)}[t])_{xyk} + b^{(2)}_k) \tag{2.8}$$

$$C^{(2)}_{xyk}[t] = (\mathbf{W} * \mathbf{H}^{(1)}[t])_{xyk} \tag{2.9}$$

$$\tilde{H}^{(2)}_{xyk}[t] = \xi(\kappa_k H^{(1)}_{xyk}[t] + \beta_k C^{(2)}_{xyk}[t] + \omega_k H^{(1)}_{xyk}[t] C^{(2)}_{xyk}[t]) \tag{2.10}$$

$$H^{(2)}_{xyk}[t] = \eta_t(H^{(2)}_{xyk}[t-1](1 - G^{(2)}_{xyk}[t]) + \tilde{H}^{(2)}_{xyk}[t] G^{(2)}_{xyk}[t]) \tag{2.11}$$

where $\mathbf{U}^{(2)}$ is similar to the "update" gate in GRU, $\mathbf{b}^{(2)}$ is the bias of the "update" gate, ξ is the non-linearity function, κ, β and ω are learnable scalar parameters controlling the quadratic and linear components of both $H^{(1)}[t])$ and $C^{(2)}[t]$, and η is the set of scalar parameters controlling the weight of each time step. $\tilde{H}^{(2)}[t]$ is the "candidate" activity, which is integrated with the previous activity $H^{(2)}[t-1]$ into the output activity for the current time step, $H^{(2)}[t]$.

It has been shown that modern feed-forward convolutional networks need considerable depth to achieve large receptive field (RF) sizes, which makes processing global contextual information incredibly expensive in terms of model size (number of parameters), as well as computation and memory costs. Compared to traditional feed-forward networks, the hGRU achieves high effective receptive field (RF) area with minimal network depth and efficient use of weights, by reusing the save horizontal processing kernels through all its execution time steps. It has been shown that a very small hGRU can achieve the same performance as very large

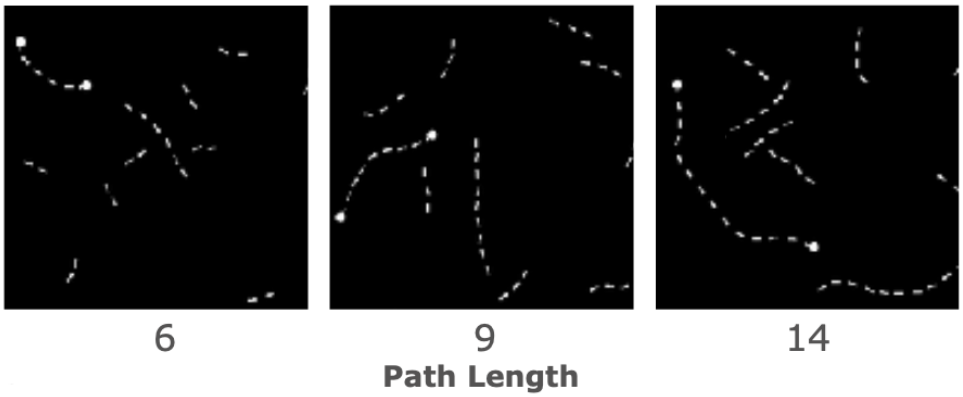

Figure 2.7. The Pathfinder Task requires the model to integrate global contextual information for the correct solution. Traditional feed-forward CNNs either struggle with inadequate RF size or inefficient use of parameters, while the hGRU solves this problem with minimal network depth and parameter count [15].

feed-forward networks in tasks that require global context, a prime example of which is the Pathfinder Task, a synthetic task where models are trained to determine if two dots in the input image are connected by a dashed path.

γ-Net Architecture

The γ-Net Architecture extends the U-Net architecture, adding both horizontal and top-down information pathways during recurrent processing. γ-Net does not remove any component of the base U-Net architecture, although it does decrease the block sizes at each layer of the network to allow more parameter-efficient learning through recurrence.

The horizontal and top-down connections are implemented by incorporating a modified version of the hGRU, the Feedback Gated Recurrent Unit (fGRU), into each layer of the U-Net. We note that the fGRU differs from hGRU in that it does not enforce the source of its two input tensors; in hGRU, the module's outputs are always routed back to itself as previous the activity $H^{(2)}[t-1]$, and the other input is always the forward drive, but fGRU allows arbitrary contextual input in place of the previous activity, which enables a general context-integration operation between the two input tensors. Much like the hGRU, the fGRU is a anatomically-constrained method to incorporate contextual information into the forward drive in recurrent processing.

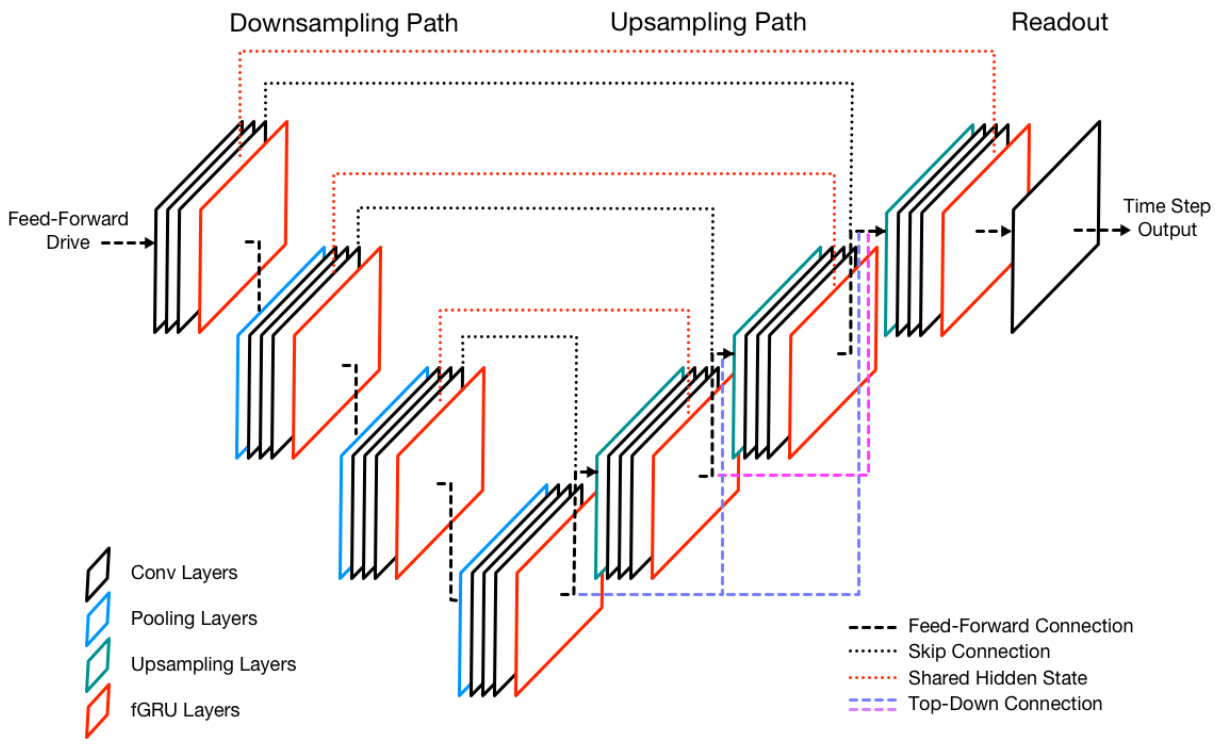

Figure 2.8. A 9-block, 4-depth level γ-Net Architecture. The black, blue and green components in the graph denote the original, feed-forward U-Net architecture, while the red components mark the γ-Net modifications. Note that the fGRU units in the graph do not accept their own hidden activities, but instead pass their hidden activities to the fGRU in the upsampling block of the same layer, and receive hidden activites from said fGRU. Additionally, top-down connectivity is implemented by concatenating the fGRU activities of each layer below the current layer along the feature dimension in the upsampling path, which are then concatenated with the skip activity before the convolutional block. The information flow at each layer of the network resembles a γ, hence the name.

Crucially, the fGRU allows for a key innovation in γ-Net: instead of treating each recurrent fGRU layer as its own self-contained module, and performing recurrent computation independently for each, γ-Net treats the entire network as one recurrent "cell," and allows data to flow through all its modules in a single time step. In addition to propagating information horizontally, this also enables top-down information flow. Through this highly interconnected architecture, the output of each functional block in γ-Net becomes a highly non-linear function of every other functional block, enabling complex computations with a relatively small number of parameters.

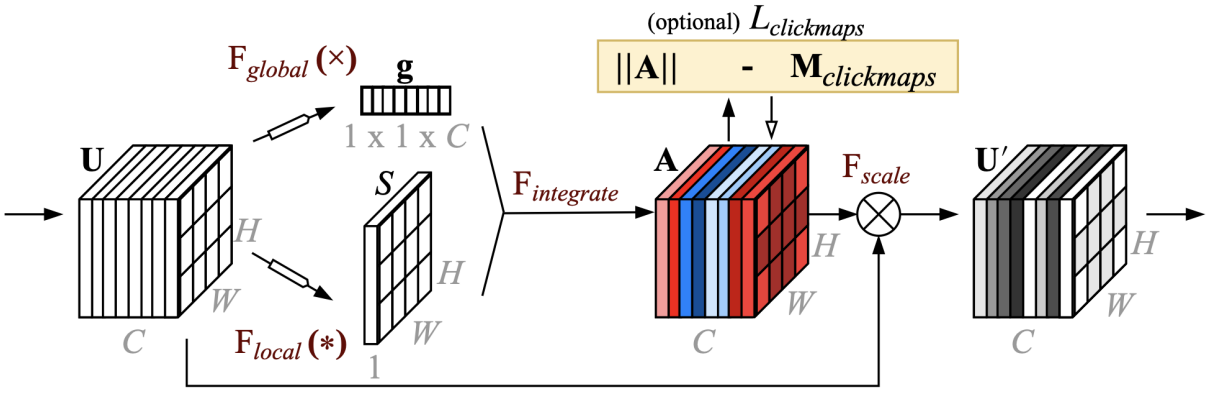

Figure 2.9. The Global and Local Attention Layer [16]. The input volume is split into two distinct pathways, the global attention and local attention pathways, each of which learns "what" and "where" to attend, respectively. The global and local attention maps are then integrated to produce a attention volume of the same shape as the input volume, which can then be used for multiplicative attention.

γ-Net also implements a form of self-attention, Global And Local Attention (GALA), in order to further constrain internal activities and improve performance [16]. The Global and Local Attention layer, as the name suggests, splits incoming information into two distinct pathways – local and global – then recombines the two forms of attention to produce a final attention volume which can be applied to the input data.

We note that the GALA self-attention mechanism essentially emulates the design philosophy of the U-Net architecture, by constructing separate information processing pathways for both context and localization, then integrating information from both pathways to produce a final result. Conceivably, this mechanism can be used as a general-purpose image processing unit, not only for estimating attention maps.

Perceptual Artifacts and Illusions in γ-Net

It has been found that γ-Net exhibits sensitivity to the center-surround tilt illusions that humans perceive, which is further evidence of the validity of the neurological model that inspired the network's architecture.

This sensitivity to visual illusions exemplifies the importance of horizontal and spatial

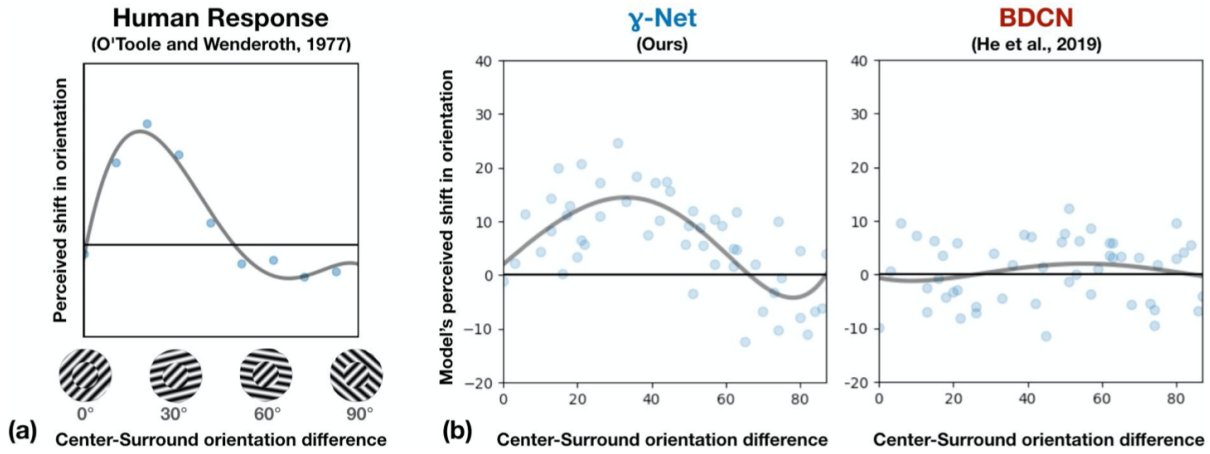

Figure 2.10. γ-Net exhibits sensitivity to the center-surround tilt illusion experienced by humans. Comparable feed-forward models fail to exhibit this sensitivity.

context in the human visual cortex's ability to process inhomogenous data and generalize to arbitrary new data sources [14], and is one of the factors that led us to choose γ-Net for the segmentation task, the performance on which is very sensitive to both global context and local information. While the U-Net architecture has already been shown to integrate global and local information efficiently, we think γ-Net's additional recurrent connections will produce more salient features, and allow the model to converge to better outputs than the U-Net can produce.

2.2 Model Complexity and Analysis in FCNs and RNNs

In recent years, advances in computer hardware has allowed increasingly complex models, with parameter counts in the range of millions to train successfully and produce state-of-the-art results. Over time, however, studies have found that the widespread trend of model overparameterization not only increases the amount of computation and storage resources needed to train models for specific tasks, but also degrades model performance in some cases when compared to more lightweight models.

A range of model optimizing and pruning techniques have been proposed to deal with model overparameterization in feed-forward and recurrent neural networks [12, 8, 9]. A previous study by Uys (2019) found that the original U-Net proposed by Ronneberger et al. (2015)

is significantly overparameterized for CMR image segmentation tasks, and investigated the most effective forms of network pruning for U-Net in CMR imaging applications [32]. Uys's paper proposed a lightweight implementation of U-Net, which reduces the network's number of parameters by 98% compared to the original U-Net, while reaching state-of-the-art performance on the Automated Cardiac Diagnosis Challenge dataset.

For the purpose of this thesis, we'll pitch our implementation of γ-Net against Uys et al.'s U-Net implementation, as the two models have similar parameter counts, and is most appropriate for demonstrating the effect of added recurrence in γ-Net.

2.2.1 Difficulties in Training Recurrent Neural Networks

Due to their highly nonlinear nature and theoretical ability to maintain information for a large number of time steps, recurrent neural networks are known to be difficult to properly train and tune for many individual tasks [4]. On one hand, recurrent neural networks face many of the same problems as deep feed forward neural networks, such as vanishing and exploding gradients during back-propagation-through-time [21]. On the other hand, recurrent networks face difficulties in tasks such as learning long-term temporal dependencies, and in correctly applying gradient descent optimizers [3]. Recurrent networks also take much longer to train than feed forward networks of similar size, due to the need to perform the same operations for multiple time steps.

Pascanu et al. provided a thorough analysis on the vanishing/exploding gradient problems in recurrent networks, building on the 1994 paper by Bengio et al. [4, 21]. Pascanu showed via formal proof that without normalization, it is trivial for recurrent networks that run for a sufficient number of time steps to have individual back-propagated gradients to either asymptote to zero or explode to infinity. As γ-Net is a large recurrent network, normalization is more pertinent, and Linsley et al. (2018) took care to apply a multitude of instance normalization and batch normalization methods throughout γ-Net [15, 14].

Bengio et al.'s 2013 paper provided an overview of the difficulty in training recurrent

networks, noting that their highly non-linear mapping made them difficult to initialize, as slightly different initializations may cause the recurrent model to learn completely different mappings; recurrent networks are also subject to underfitting due to dense activation patterns, a result of gradient diffusion which makes it difficult for units in recurrent networks (and also deep feed-forward networks) to specialize [3]. The paper proposed a range of solutions to these problems, such as applying leaky integration to time step activations, enforcing output regularization and using a more robust gradient optimizer. Due to time and resource constraints, we were not able to apply these solutions in our experiments, but we hope to so do in the future.

In addition, activities in recurrent networks are also harder to visualize and interpret, since information from multiple time steps is superimposed together, and each time step performs a non-linear map that completely changes activities from the previous timestep. Without advanced visualization and interpretation techniques, recurrent networks remain black boxes, which hinders their deployment in mission-critical fields such as medical imaging, where inferential transparency is crucial.

Chapter 3

Data

3.1 CMR Data Basics

3.1.1 CMR Imaging Procedures

As MRI imagers can only produce 2D images, CMR images must be captured along a given axis of the heart. For datasets used in this work, all images are taken along the short axis, or the rostral-caudal axis, which makes horizontal slices of the heart. This is because the left ventricle, which is the target for segmentation, is most clearly delineated by its thick, circular walls when imaged along the short-axis.

MRI machines are capable of quickly capturing a collection of images slices along the given axis of the heart, with only a few miliseconds of delay between each slice. This collection of slices is known as a frame. In one CMR imaging session, multiple frames are usually captured for a single patient, covering at least half a heart cycle starting from end-diastole (ED) and ending in end-systole (ES), between which the volume of the left ventricle ejection fraction (LVEF) can be calculated. The process results in a four-dimensional data volume for a single CMR session of a single patient: one 2-D grayscale image for each slice of each frame from the CMR session.

3.1.2 Common CMR Data Formats

We have obtained data in two common MRI data formats, the Digital Imaging and Communications in Medicine (DICOM) format and the Neuroimaging Informatics Technology

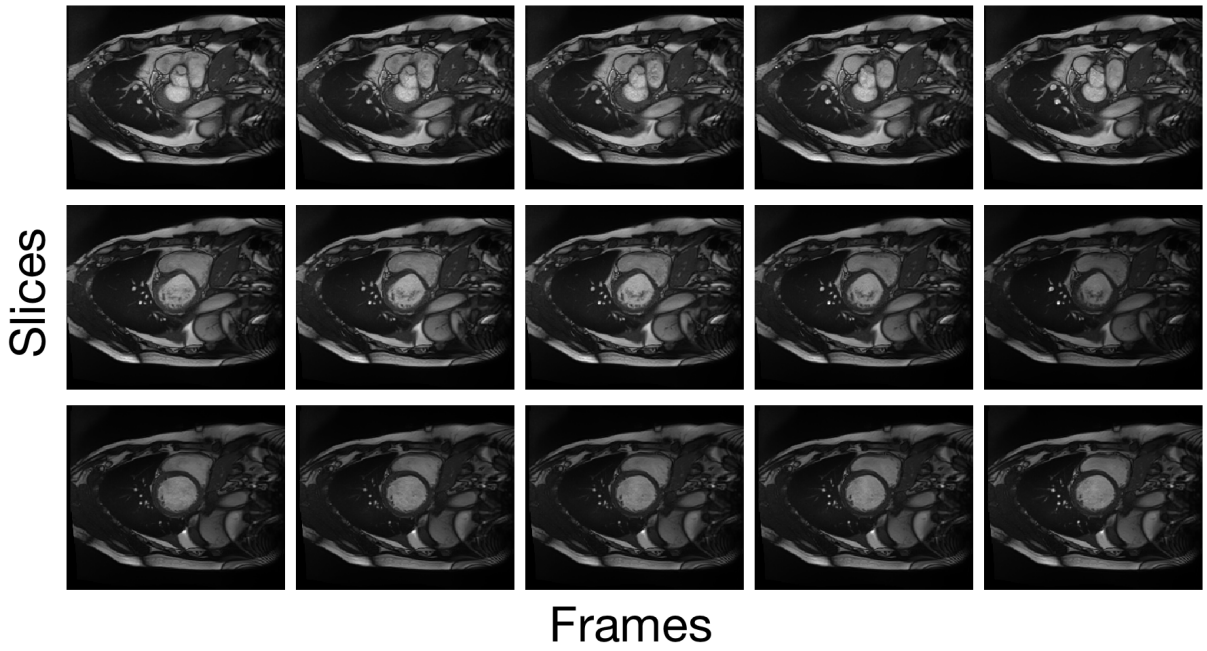

Figure 3.1. The organization of frames and slices in the ACDC data. Frames represent the temporal axis, and slices represent the spatial axis along the rostral-caudal direction.

Initiative (NIfTI) format. Of the two, DICOM is the more widely used format, as it's the format most commercial MRI machines produce, and includes a wide range of standard and custom metadata fields; the downside of the DICOM format is that it doesn't support packaging multiple frames into a single file, and that metadata fields often have missing data. The NIfTI format is specifically designed for use in medical research, and allows easy packaging of multiple frames into a single file, thus simplifying data storage and access.

3.2 Datasets

We make use of two different cardiac imaging datasets in this thesis, the Automated Cardiac Diagnosis Challenge (ACDC) dataset [5], and the Sunnybrook Cardiac Dataset [25]. The ACDC dataset, being the larger and more comprehensively labeled dataset, is used for model training and validation. The older Sunnybrook dataset is used for model generalization experiments.

3.2.1 The Automated Cardiac Diagnosis Challenge Dataset

We obtained our training data from the Automated Cardiac Diagnosis Challenge (ACDC) dataset, which was created using clinical data obtained over a period of six years and from two different MRI machines [5]. The ACDC challenge aims to specifically foster development of automated cardiac disease diagnosis, and contains a relatively large amount of segmented cardiac images. The ACDC dataset is also made available in the NIfTI format, which makes the dataset more available to medical researchers. This is the largest and best-quality publicly available human cardiac segmentation dataset that we can find. We note that the dataset also contains detailed metadata on disease conditions associated with each patient image, but we don't need these data for the purpose of this study.

The ACDC data are already split into the training and test sets, where the test data have no segmentation labels and have very little associated metadata. For this study, we use the training set exclusively to obtain credible validation performance scores, since it is difficult to quantify segmentation performance without ground truth data.

The ACDC training set is organized by patients with anonymized patient IDs, and is organized by frames subdivided by slices, as illustrated above. Of all the frames, only one end-diastole and one end-systole frame are segmented. All slices within the abovementioned frames are individually segmented.

We note that the ACDC dataset is also the most well labelled we could find, with individually segmented labels for inner border of the left ventricle, the outer border of the left ventricle, and the inner border of the right ventricle for each slice of the ED and ES frames.

For training, we performed a training-validation split by patient. Since images from the same patient are often very similar to each other, this ensure that the models will not be trained on samples that are very similar to those in validation.

3.2.2 The Sunnybrook Cardiac Dataset

The Sunnybrook Cardiac Dataset, also known as the 2009 Cardiac MR Left Ventricle Segmentation Challenge Dataset, was originally produced by Radau et al. for their 2009 paper and subsequently put into public domain [25]. The dataset consists of CMR images from 45 patients and control group members. All images are accompanied by ground truth LV segmentations, as well as metadata pertaining to basic patient information, pathologies, and left-ventricular finite element models (CAPs). For the purpose of this study, we will use only the CMR images and their LV segmentations.

For the Sunnybrook data, LV segmentation labels are available only in the form of polygon vertex coordinates, where the contour of the LV segment is traced by connecting nearby vertices while enforcing convex constraints. This makes it necessary to first convert these coordinates into pixel masks in order to train neural network models. To this end, we made use of the OpenCV library's fill_poly function to draw and fill the LV segmentation on a blank canvas using the coordinates for each set of LV label coordinates [6].

Chapter 4

Methods

4.1 Experimental Setup

All experiments were performed using the Cognitive Hardware and Software Ecosystem Community Infrastructure (CHASE-CI), and the Nautilus HyperCluster maintained by the Pacific Research Platform [2, 29]. Each experiment was trained on a single Nvidia GTX 2080Ti GPU, each of which has 12GB of VRAM. We opted not to train any model on multi-GPUs due to resource constraints.

We used the PyTorch machine learning platform for easy implementation of complex recurrent circuits as required by the γ-Net architecture. PyTorch's implementation of fast eager execution has been invaluable to accelerating our training process, and the platform's dynamic computation graphs makes sure that the smallest amount of VRAM is used during training, making it possible to train more timesteps and larger minibatches [22].

We note that, compared to Linsley's implementation of γ-Net in Tensorflow 1.x, our PyTorch implementation is much more flexible and runs about two times faster, thanks to VRAM saving and an efficient implementation of eager execution.

4.2 Data Preprocessing

Multiple studies have found that input data selection and preprocessing is crucial to increasing the performance of U-Nets and similar models [32, 11]. In particular, we control for

image noise, brightness variation and pixel value range for the purpose of our experiments.

We also discuss methods of assembling mini-batches, which we have also found to be an important factor in model performance. Specifically, we test for the effect of training our models with different mini-batch sizes.

4.2.1 Gaussian Smoothing

During earlier training sessions, we noticed clear noise patterns in the learned kernels of γ-Net, which may have been magnified due to the recurrent nature of the architecture. We have thus opted to perform Gaussian Smoothing of all input images before feeding them into the network. We chose a Gaussian kernel of size 3 and standard deviation of 1, which allows noise reduction while keeping image blurring to a minimum.

4.2.2 Brightness Redistribution via CLAHE

Because CMR images often have "bright spots" which will cause regions of the image with different brightness levels to contribute differently to the prediction, we used the Adaptive Histogram Equalization (AHE) technique to redistribute brightness levels over the input images, in order to improve local contrast and decrease global variation in brightness. In particular, we performed Contrast Limited AHE (CLAHE) processing to avoid over-amplifying the noise in relatively uniform regions of the image [24].

4.2.3 MinMax Scaling

To ensure a constant range of input pixel values, we applied MinMax scaling to bring the pixel value range of each image to between zero and one. Through experimentation, we have found that the performance of U-Nets and similar architectures are very sensitive to changes in model input range, and MinMax scaling will alleviate this issue by mapping pixel values to a fixed range and allow U-Nets to generalize to new datasets more readily.

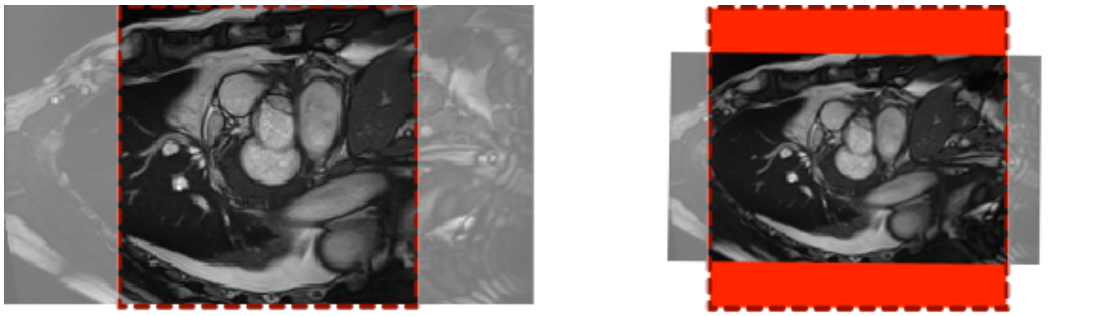

Figure 4.1. Image resizing and cropping. The original image is only downsampled if its shorter edge is larger than the crop size, otherwise the image won't be resampled; this is to preserve as much of the original image as possible if the original images are large, or preserve the pixel resolution if the original image is small. Empty regions in the result image is padded with black pixels, then a center crop is performed.

4.2.4 Image Resizing and Cropping

To correctly batch the input images for minibatch training, we need to enforce uniform dimensions for all input images. This can be challenging, as medical images (CMR images in particular) are often pre-cropped to contain only areas of interest, and come in a wide range of different dimensions.

To make sure that the resized images retain the maximum amount of information from the original images, we apply the following procedure. First, we determine a size for the resized image, which is kept square to accommodate original images of different shapes; this size must be a multiple of 2^{k-1}, where k is the depth of the U-Net or the γ-Net tested, in order to ensure that upsampling operations are performed correctly. Then we determine whether or not to downsample the image, based on the shorter axis of the original image: if the short axis of the original is shorter than that of the target size, we do not downsample the image to preserve as much information in the original as possible; if the shorter axis is longer than the target size, we downsample the original such that the shorter axis becomes equal in size to the target size. Last, we pad the image, if necessary, so that the shorter axis is the same size as the target size, and perform a center-crop to obtain the final resized image.

4.2.5 Train-Validation Split

In biomedical image segmentation datasets, medical images are often grouped by the patient, from whom the images are taken, and images from the same patient are often more similar to each other than to those from other patients. Thus we need to perform the train-validation split by patient, instead of by individual images, to ensure that our validation performance numbers truly represent the model's ability to perform well on unseen data.

To this end, we designed a data loader that identifies each image using its metadata, and group images from the same patient together. The patients are then shuffled randomly, and the train-validation split is performed according to a predetermined validation set ratio. Note that this method doesn't guarantee that the number of images in the training and validation sets exactly conforms to the predetermined split ratio, but we've determined empirically that this difference is not sufficiently significant to have an impact on validation performance.

4.3 Model Training

4.3.1 Experiment Harness

We have observed unstable behaviors while training both the U-Net and γ-Net, where model performance (in particular, the validation BCE and Dice scores) would consistently worsen for a period of time after reaching a low point, which matches criteria of overfitting, but then suddenly spike as the model's performance increases drastically. The reverse also happens, wherein model performance would increase drastically and fall back to a worse level. Because of these observations, we have decided to not employ early-stopping in our experiments, and allow the model to train for a long period of time until its performance stabilizes. We have found through experimentation that prolonged training does not lead to significant overfitting for either model, taking into account both validation loss and visual inspection of segmentation results.

To ensure that we can get the best model out of each training run, we chose to save the model regularly at short, predefined intervals measured in number of epochs; we also save the

validation scores for each validation minibatch at each training epoch to assist us in finding the best model from the training run.

We have also devised an automated pipeline to accelerate model training, by leveraging the batch job system of the Kubernetes cluster management platform, in use at SDSC's Nautilus Hypercluster. This pipeline allows for a large number of concurrent training pipelines, and takes advantage of the hundred of GPUs in use in the Nautilus Hypercluster, thus drastically decreasing the amount of time needed to perform model fine tuning and hyperparameter search.

4.3.2 Model Overviews

We outline the two main models we have tested for this work, an implementation of U-Net by Uys et al [32], and our PyTorch implementation of the γ-Net.

U-Net

We have opted to use Uys et al's lightweight implementation of the U-Net, which previous work has found to not only require very little resource to reach impressive performance compared to the standard U-Net, but also is more specifically suitable for CMR image segmentation tasks [32]. The parameter count of this lightweight U-Net is only a small fraction of that of a standard U-Net, and yet is able to achieve better performance than the latter.

γ-Net

We reimplemented Linsley et al's γ-Net code, which was originally written in TensorFlow 1.x, using the PyTorch platform. We chose to use a 5 depth-level architecture for the γ-Net, in order to stay consistent with the U-Net we are comparing it against. Most of the network's configurations were kept the same as in Linsley et al's original implementation, such as the network topology, use of normalization, sizes of the feed-forward convolution blocks, and upsampling methods. We experimented with different numbers of timesteps at which to train the network, and minibatch sizes. As the original γ-Net paper didn't specify the read out block to use, we attached a convolution layer at the end of the network to calculate the final, single-channel

prediction of the segmentation label. The convolution layer uses a single $1 \times C \times 5 \times 5$ kernel, where C is the number of output channels of the γ-Net stack.

In addition, we have made modifications to the original γ-Net architecture beyond Linsley et al's original paper, at the suggestion of the author's open source code and later papers [16]. In particuar, we added Global-And-Local Attention (GALA) layers to the fGRU components of the network, to allow the model to better focus on regions of interest during recurrent processing.

4.3.3 Pairwise Training

To make direct comparisons between U-Net and γ-Net, we construct experiments such that the two models are trained to complete exactly the same tasks. We control the training environment by performing all training tasks inside docker containers, which are constructed using the same docker image; whenever possible, we also run the training containers on identical hardware nodes. We also control the preprocessing steps for each training run, such that the two models are fed identical training and validation data; in particular, the train-validation split is performed identically for each paired training instance of the models. We seed the PyTorch random number generators in order to produce replicable results.

In addition to using the binary cross entropy loss for validation, we also use the soft Dice similarity coefficient, which is differentiable and can be used to back propagate errors. To ensure correctness, we pass the raw model outputs through a sigmoid squashing function to bring the output range to between zero and one, then calculate the dice coefficient. Both the U-Net and the γ-Net are trained for 200 epochs on the same training data for each hyperparameter configuration. Each training run is repeated five times, and performance scores are averaged from each run.

4.4 Reduced Training Set

In Linsley et al's original paper, one of the most important claims regarding the γ-Net model is that it requires only a very small dataset compared to feed-forward models in order to perform well on the validation set. We have thus decided to test this claim on the CMR

data by adjusting the train-validation split ratio between 1:9 to 8:2. For each train-validation split, we control the seed used in the random split and shuffling processes to ensure experiment repeatability. The train-validation split is made on patients instead of individual images to avoid contaminating the training set, since images from the same patient are often very similar.

We performed this experiment for both the U-Net and γ-Net models, to establish a side-by-side comparison of the two models' ability to train on a small dataset and perform well on the validation set.

We note that this experiment is important especially in medical data processing, where patient data are expensive to obtain and are often available in very small datasets. The ability to train a model successfully on a small dataset and generalize it to a large amount of unlabeled patient data is invaluable in medical applications.

4.5 Training with Temporal Context

We opted to perform an additional experiment for γ-Net only, inspired by the successful application of recurrent convolutional neural networks in video segmentation. In this experiment, we use the unlabeled frames as temporal context for the labeled frames, and stack a predetermined number of unlabeled frames before the labeled frame as the input data. This turns γ-Net into a many-to-one recurrent network, instead of the original one-to-one model.

To build the new dataset, we build a timeseries of slices for each slice in the labeled frame (ED or ES), preserving the order of the frames where the slices come from. As the ED frames are often the first frame in the available frames, we reverse the order of frames to build the timeseries for the ED slices, such that each labeled ED slice would be at the end of the time series. We also modified γ-Net to allow it to accept time series data instead of single images. Since the entire γ-Net architecture is ran as a recurrent cell, this involves only allowing the model to read a new input image at each timestep, instead of forcing the input to be the same input image.

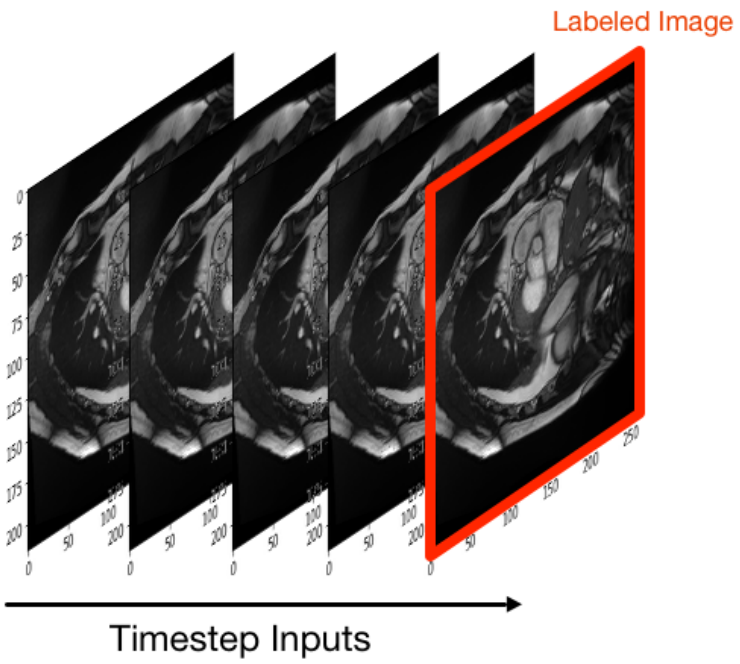

Figure 4.2. Time series for Training γ-Net with Temporal Context. Pictured is the assembled time series of frames for a single slice from a single imaging session. The model is tasked to predict the label of the last image in the image time series.

4.6 Model Evaluation

4.6.1 Model Validation

For each experiment, we use the validation set obtained from the train-validation split before training to gauge model performance. We perform model validation at the end of each training epoch, and record the average validation score over all images in the validation set.

We obtain validation scores for both binary cross entropy loss, which is the same metric for training the model, and the Dice coefficient, which is a better metric for determining segmentation performance [36]. For hyperparameter tuning, we use only the Dice coefficient and disregard cross entropy loss. We note that binary cross entropy can be seen as an approximate of the Dice coefficient, but is much more numerically stable and easily differentiable, which makes the cross entropy loss more suited for training the model compared to the Dice loss function.

4.6.2 Model Generalizability

We test for model generalizability using the Sunnybrook Cardiac Dataset, which is a much smaller dataset than ACDC and contains only LV labels.

Generalizability Without Fine Tuning

For this study, we have decided to not fine tune our trained models on the Sunnybrook training set, to simulate application scenarios where the model is deployed as a static system and cannot be further adjusted to new CMR data. It is our intention, however, to conduct transfer learning experiments on other datasets, and observe model performance after fine tuning on new data.

Generalizability Using Reduced Training Set

We have also performed generalizability experiments using models trained with smaller training sets, to observe the models' ability to generalize when given less training data in the first place. With this experiment, we intend to push the U-Net and γ-Net models to their limit, and fully expect them to show severe performance degradation when given only a fraction of the training data.

Chapter 5

Results

5.1 Model Performance

We have obtained our baseline performance scores using the following model configurations and training setup. The γ-Net was trained for 2 time steps for each input image and used a mini-batch size of 2, which was determined empirically. The U-Net was trained with a mini-batch size of 8, in accordance to Uys et al's conclusions [32]. Each model was trained for 5 times, and the resulting scores were aggregated.

At the baseline configuration, our γ-Net implementation showed minimal performance increase over our lightweight U-Net implementation. This is expected, as previous studies [32] have found that the lightweight U-Net implementation performs very well with this training configuration.

The means and standard deviations of the reported Dice scores are calculated by aggregating the dice scores obtained after each training epoch of each training run; i.e. for each experiment configuration, we have calculated the mean and standard deviation over all training

Table 5.1. Performance γ-Net vs U-Net. The γ-Net exhibits minimal performance improvement over the U-Net when trained with sufficient data and standard data preprocessing.

Model	γ-Net	U-Net
Validation Dice Avg	0.882	0.884
Validation Dice Stdev	0.021	0.005

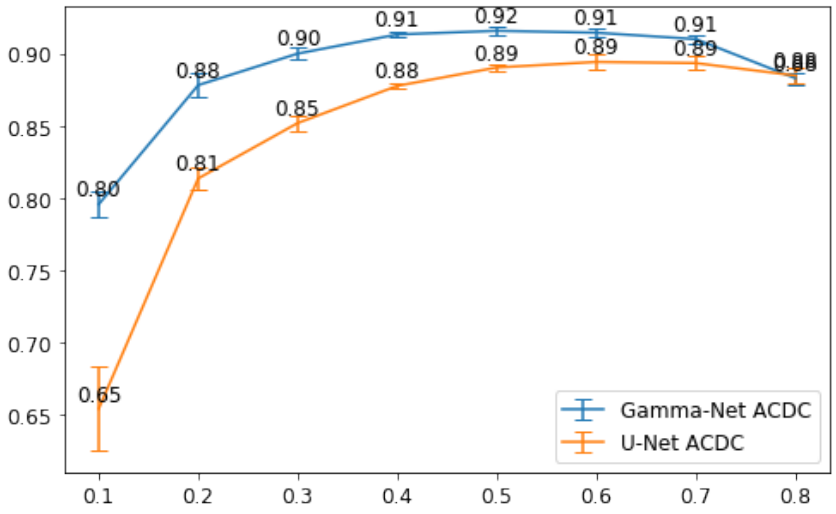

Figure 5.1. Validation Dice scores vs training set ratio. Validation is run using the ACDC validation set from the train-validation split. We note that γ-Net shows increasing peformance gain over U-Net as the training set ratio is reduced.

epochs of all training runs. This is due to the highly unstable dice score curves we have observed during earlier training runs. We will elaborate on this observation in the discussion section below.

5.2 Reduced Training Set

We obtained performance scores for our reduced training set experiments using the same baseline setup as above, and varying only the train-validation split ratio. Here, our γ-Net implementation has exhibited performance improvement over our baseline U-Net model when trained on reduced training sets, and validated on the complementary validation set. For each train-validation split we have experimented on, with the exception of the original 8:2 split, the γ-Net's performance is significantly better than that of U-Net with $p < 0.001$.

We have also observed increasingly large performance improvements for the γ-Net over the U-Net while reducing the train-validation ratio, which shows that the γ-Net is much more resilient to smaller datasets compared to the U-Net. This corroborates Linsley et al's original claim for γ-Net, that the architecture trains much better on smaller datasets than comparable feed-forward models.

Table 5.2. Generalization γ-Net vs U-Net. Obtained using the Sunnybrook LV segmentation dataset. γ-Net performs significantly better than U-Net with $p < 0.01$.

Model	γ-Net	U-Net
Generalization Dice Avg	0.757	0.721
Generalization Dice Stdev	0.052	0.011

5.3 Model Generalization

5.3.1 Generalizability

We obtained Dice scores on the Sunnybrook LV segmentation dataset using our baseline models with configurations specified in Section 5.1, and aggregating scores for 5 training runs. The models were not fine tuned to the Sunnybrook dataset, but were used to infer the labels directly from the new data after image preprocessing.

Our γ-Net implementation outperforms the U-Net by over 3%. This is a significant improvement, at $p < 0.01$. We have expected this result, since the γ-Net's recurrent architecture effectively allows it to train for longer on the training data, and reusing weights via recurrence mitigates the effects of overfitting.

5.3.2 Generalizability Using Reduced Training Set

We tested model generalizability using γ-Net and U-Net models trained with reduced training sets. The models tested are all trained with the baseline configuration, with only varying train-validation split ratios during model training.

We observed generalizability improvements for our γ-Net implementation over the U-Net when trained with all train-validation split ratios. In general, γ-Net exhibits more generalizability improvements over U-Net with smaller training set sizes, but we note that when trained with larger training sets, the U-Net catches up quickly with the γ-Net in generalizability, and the performance gap becomes insignificant.

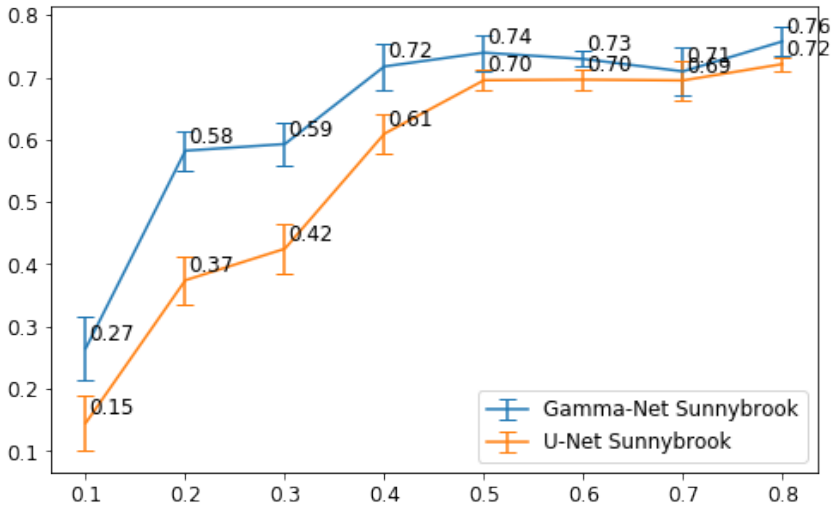

Figure 5.2. Generalization Dice scores vs training set ratio. Generalization experiments are run using the Sunnybrook validation set. We note that γ-Net shows increasing generalization peformance gain over U-Net as the training set ratio is reduced.

5.4 Sensitivity to Input Data Quality

We have also examined model performance when given poorly preprocessed data, which may have meaningful impacts in realistic applications where data sources are heterogeneous and input images may not be properly processed. We test this by attempting to take out each of the three preprocessing steps: Gaussian smoothing, CLAHE brightness redistribution, and MinMax scaling of pixel values.

5.4.1 Sensitivity to Image Noise

By removing the Gaussian filter specified in section 4.2.1, we test the models' resilience to image noise. We trained the γ-Net and U-Net models in their baseline configuration with only the Gaussian filter removed from the preprocessing steps, and compare validation performance against the baseline models. All scores are aggregated from 5 training runs.

We have found no significant validation performance degradation after removing the Gaussian filter, which indicates that image noise is not a significant performance bottleneck in our current experiment setup. In fact, there is a noticeable (but not significant) performance

Table 5.3. Sensitivity to image noise γ-Net vs U-Net. There is no significant performance impact in the baseline models when Gaussian filtering is removed.

Model	γ-Net	U-Net
Validation Dice	0.882 (0.021)	0.884 (0.005)
Generalization Dice	0.757 (0.052)	0.721 (0.011)
Validation Dice w/o Gaussian Filtering	0.892 (0.020)	0.885 (0.005)
Generalization Dice Avg w/o Gaussian Filtering	0.701 (0.049)	0.709 (0.020)

Table 5.4. Sensitivity to global brightness variation γ-Net vs U-Net. There is no significant performance impact in the baseline models when CLAHE preprocessing is removed, except for γ-Net generalization.

Model	γ-Net	U-Net
Validation Dice	0.882 (0.021)	0.884 (0.005)
Generalization Dice	0.757 (0.052)	0.721 (0.011)
Validation Dice w/o CLAHE	0.887 (0.022)	0.881 (0.005)
Generalization Dice w/o CLAHE	0.665 (0.060)	0.708 (0.034)

increase in our γ-Net model after removing Gaussian filtering.

There is a noticeable (but not significant) decrease in generalization performance for both $\gamma - Net$ and U-Net, indicating that the Sunnybrook dataset is possibly more noisy than the ACDC dataset.

5.4.2 Sensitivity to Global Brightness Variation

We test the models' resilience to brightness variation by removing the CLAHE brightness redistribution process from the preprocessing pipeline. We trained the γ-Net and U-Net models using the baseline configuration, taking out only CLAHE during image preprocessing. Scores are aggregated from 5 training runs.

We have not observed significant performance degradation after removing CLAHE preprocessing for both models, An exception is the γ-Net generalizability experiment, where there is a significant performance decrease with $p < 0.05$. This indicates that γ-Net is more sensitive to global brightness variation when generalizing to new datasets, which is surprising.

Table 5.5. Sensitivity to pixel value range γ-Net vs U-Net. U-Net is significantly impacted when pixel value range is not controlled.

Model	γ-Net	U-Net
Validation Dice	0.882 (0.021)	0.884 (0.005)
Generalization Dice	0.757 (0.052)	0.721 (0.011)
Validation Dice w/o MinMax Scaling	0.887 (0.017)	0.863 (0.003)
Generalization Dice w/o MinMax Scaling	0.742 (0.046)	0.003 (0.003)

5.4.3 Sensitivity to Pixel Value Range

We examine the models' resilience to varying pixel value ranges by removing MinMax scaling, noting that the average pixel value between the ACDC and Sunnybrook datasets differ by almost two orders of magnitude. Models are trained using the baseline configuration, with only MinMax scaling removed from preprocessing steps. Scores are aggregated over 5 training runs.

We observe no significant performance degradation after removing MinMax scaling for γ-Net, even for generalization without fine-tuning. This indicates that γ-Net is resilient to pixel value ranges in the input. On the other hand, U-Net exhibits significant performance degradation ($p < 0.01$) for both validation and generalization tests, exhibiting poor resilience to different input pixel value ranges. In particular, U-Net completely fails the segmentation task when generalizing to a new dataset with a different pixel value range, and tends to predict all negatives everywhere in the image.

5.5 Training with Temporal Context

We investigated whether adding temporal context while training our γ-Net implementation offers performance improvement. The following γ-Net models were trained for 1 training run, with varying mini-batch sizes and sequence length of added temporal context. Performance scores were taken using early-stopping, and selecting the best score in a single training run.

We note a slight improvement in γ-Net's validation performance, which signals that

Table 5.6. γ-Net validation performance with temporal context. In the configuration labels, "ts" stands for the sequence length (time steps), while "bs" stands for batch size. There is a slight but noticeable improvement when temporal context is applied.

Model	Baseline	bs=2,ts=4	bs=2,ts=6	bs=4,ts=4	bs=4,ts=6	bs=6,ts=4
Validation Dice	0.8927	0.8967	0.8942	0.9049	0.8917	0.9108

providing temporal context offers information that the recurrent architecture can take advantage of during training. There is also a noticeable difference between different mini-batch size and sequence length configurations. For this experiment, we found that a mini-batch sizes of 6 and sequence length of 4 yielded the best performing model.

While inspecting model inputs and outputs, we noticed that images in the same input sequence do not vary significantly, with only marginal change between each frame. It's possible that training γ-Net with longer temporal sequences will yield better performance scores, but we have not done so due to time constraints on the project.

5.6 Discussion

5.6.1 Model Stability with regard to Dice Coefficients

While training the U-Net and γ-Net models, we calculated dice coefficients after each training epoch for both models, and were able to plot dice coefficient curves similar to the training and validation loss curves. Contrary to the relatively stable validation binary cross-entropy loss curves, we observed very unstable dice coefficient curves, where dice scores can vary significantly (sometimes by over 30%) between two neighboring epochs. This is particularly the case with our reduced training set experiments, where smaller training sets lead to more unstable Dice curves. Because of this observation, we have chosen to not deploy early stopping for training our models, as this would very likely yield dice scores that are not representative of the model's best possible performance.

We have tried a variety of criteria for selecting portions of the dice curves to count as

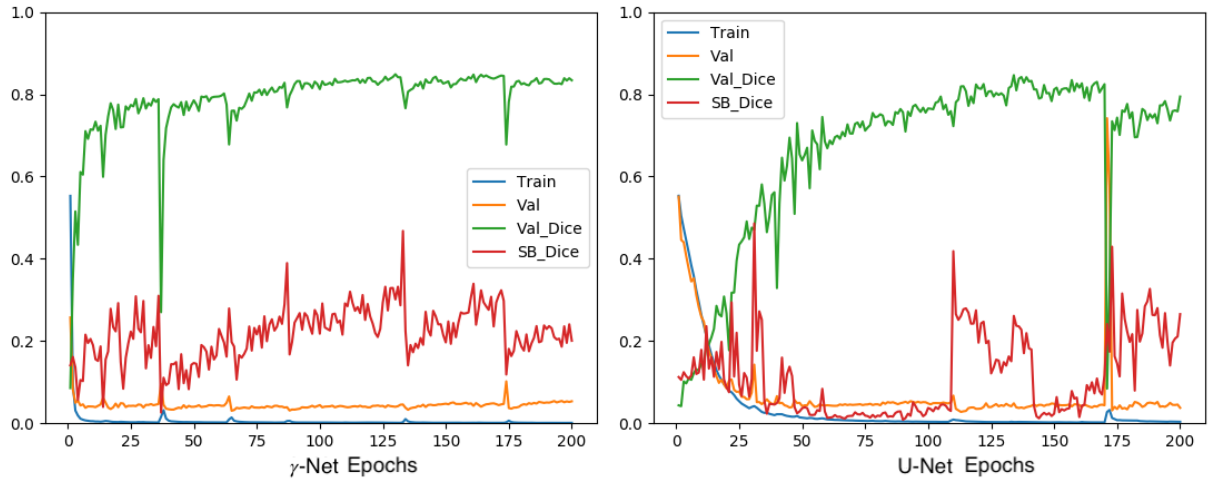

Figure 5.3. BCE and Dice curves for both U-Net and γ-Net, showing training instability. Both models were trained on 40% of the training set, and exhibit unstable curves for dice scores during the pictured sample training run, with U-Net being exceptionally unstable and varying erratically between epochs. Green graphs in the above plots represent validation dice scores, while red graphs represent generalization dice scores to the Sunnybrook Dataset.

the model's final dice scores, but failed as the curves have simply too much variation. In the end, we chose to take the mean of the model's dice performance over the entire training run, which offers an acceptable measure of the model's average performance during training, and one which allows fair comparison between the γ-Net and U-Net models. We note that by averaging dice scores over the entire training session, the resulting performance scores does not reflect the final performance of the model, but rather a combination of performance metrics including the speed and quality of convergence, as well as model stability; we only chose to calculate our performance scores in lieu of better options to obtain repeatable performance scores.

Additionally, we note that the dice curves for γ-Net are more stable than that of U-Net, and do show a general trend of improvement despite local variations and abrupt changes. We again attribute this to γ-Net's recurrent architecture and weight reuse, which constrains weight changes across epochs and prevent the model from moving too erratically on the error plane.

It is still not clear to us why our dice curves show such drastic changes while the binary cross entropy losses remain relatively low and stable, though we hypothesize that the sigmoid squashing function used to calculate the Dice scores (which is standard practice when calculating

Dice) may have magnified performance differences in otherwise similar outputs. Further work is required to investigate the true cause of these erratic dice score curves, and to improve the training pipeline to make model performance more consistent.

5.6.2 Sensitivity to Input Image Preprocessing

We have discovered that compared to the feed-forward U-Net, the γ-Net is very resilient to input pixel value range, which is surprising, considering recurrent architectures do not intrinsically deal with drastically different input ranges between training and testing.

As both our γ-Net and U-Net implementations heavily employ batch- and/or instance-normalization layers, we cannot attribute this observation to normalization. We hypothesize that γ-Net's observed resilience to pixel value ranges arises from its horizontal and top-down information pathways, which over several time steps can increase local contrast in its persistent activity maps, and as a result enhance local features.

5.6.3 Effect of Batch Sizes

We have noticed that mini-batch size is an important factor in improving model performance for both our γ-Net and U-Net implementations. In many cases, simply changing the size of the training mini-batch can produce performance gains far outshining complex model changes or image preprocessing techniques. Notably, smaller mini-batch sizes often yielded better results than larger ones for both γ-Net and U-Net.

We attribute this observed effect of mini-batch sizes to the complex nature of the error plane with regard to the cardiac image segmentation task. While short-axis cardiac MRI images don't typically have as much global variation as typical scenery images, there is a lot of subtlety in cardiac features, and it can often be difficult for even medical professionals to properly label heart tissues in cardiac images. In particular, heart chambers can vary greatly in apparent size and location in the image, heart shapes can be highly irregular, and tissue boundaries can be blurry or missing altogether. As such, it's likely that segmentation models would have to learn a

large number of very specific cases of the segmentation task, and since learning gradients from mini-batches are aggregated, gradients from drastically different cases can get mixed together and cause the model to effectively "skip over" the correct descent path in the error plane.

This hypothesis is supported by our best empirically determined batch size of 2 for our baseline γ-Net model, which is smaller than 4 for the U-Net. As γ-Net reinforces its learning by iterating on the same input images for several time steps, learning gradients are aggregated over all time steps in addition to all samples in the mini-batch, exacerbating the gradient mixture problem stated above. We do take note that we are far from reaching a conclusion about the true nature of the effects of mini-batch sizes in segmentation tasks, and will need to investigate the issue further.

Chapter 6

Conclusion

Through extensive experimentation, we compared Linsley et al.'s γ-Net model to Uys's lightweight U-Net model for cardiac image segmentation tasks, where the two models have comparable size and parameter counts and identical training conditions, and performance comparisons were made using the empirically determined best hyperparameters for each model. While γ-Net does not outperform the lightweight U-Net when given sufficient training data, γ-Net does significantly outperform the lightweight U-Net when trained using reduced training sets, yielding more comparative performance improvement for smaller training set sizes. γ-Net outperforms the lightweight U-Net when generalizing to a new dataset without additional training, and also shows more generalization performance increase when trained with smaller training set sizes compared to the lightweight U-Net.

In our experiments, we observed a variety of unexpected phenomena, such as unstable dice coefficient scores during model training, and significant impact of mini-batch sizes on model performance. Further work is needed to investigate these observations.

In the interest of eventually deploying γ-Net for cardiac image segmentation in the field, in the future we'll also need to develop proper model inspection, debugging and visualization techniques to make the model more explainable and intuitively easy to understand, as γ-Net's recurrent nature and highly-interconnected architecture obscures meaning from its activity maps and intermediate outputs.

Appendix A

Network Architectures

A.1 U-Net

Throughout our experiments, we used a lightweight U-Net heavily inspired by Uys's lightweight U-Net [32], which was designed and tuned specifically to the cardiac image segmentation task. Our implementation of the network's architecture is as follows.

Table A.1. Lightweight U-Net implementation inspired by Uys et al's 2019 study.

Depth	Conv Blocks	Conv Filters	Conv Size	Conv Stride	Pool Size	Pool Stride
1	3	8	3x3	1x1	2x2	1x1
2	3	16	3x3	1x1	2x2	1x1
3	3	32	3x3	1x1	2x2	1x1
4	3	64	3x3	1x1	2x2	1x1
5	3	128	3x3	1x1	2x2	1x1

Table A.2. Additional U-Net Hyperparameters

Network Depth	Normalization	Upsampling Method	Padding
5	Batch Normalization	Bilinear Rescaling	Same

A.2 γ-Net

We reimplemented Linsley et al's 2018 γ-Net using the PyTorch platform. The network's architecture details is as follows.

Table A.3. γ-Net reimplementation from Linsley et al's 2018 paper. Note that the architecture is outlined by functional blocks instead of network depth, with blocks 1-4 being the downsampling path, block 5 the bottleneck, and blocks 6-9 the upsampling path.

Block	Conv Blks	Conv Filt	Conv Size	Conv Strd	Pool/Ups Size	Pool/Ups Strd	fGRU Filt	fGRU Size	fGRU Attn Size
1	3	24	3x3	1x1	2x2	1x1	24	9x9	5x5
2	3	28	3x3	1x1	2x2	1x1	28	7x7	5x5
3	3	36	3x3	1x1	2x2	1x1	36	5x5	5x5
4	3	48	3x3	1x1	2x2	1x1	48	3x3	5x5
5	3	64	3x3	1x1	N/A	N/A	64	1x1	5x5
6	3	48	3x3	1x1	2x2	1x1	48	1x1	5x5
7	3	36	3x3	1x1	2x2	1x1	36	1x1	5x5
8	3	28	3x3	1x1	2x2	1x1	28	1x1	5x5
9	3	24	3x3	1x1	2x2	1x1	24	1x1	5x5

Table A.4. Additional γ-Net Hyperparameters. Not all hyperparameter options in the original γ-Net were implemented for our study.

Network Depth	Normalization	Upsampling	Attention	Attn Layers
5	Instance Norm	Bilinear	GALA	1 per fGRU

Bibliography

[1] Heart disease facts. *Centers for Disease Control and Prevention*, Dec 2019.

[2] Ilkay Altintas, Kyle Marcus, Isaac Nealey, Scott L Sellars, John Graham, Dima Mishin, Joel Polizzi, Daniel Crawl, Thomas DeFanti, and Larry Smarr. Workflow-driven distributed machine learning in chase-ci: A cognitive hardware and software ecosystem community infrastructure. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 865–873. IEEE, 2019.

[3] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628. IEEE, 2013.

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[5] Olivier Bernard, Alain Lalande, Clement Zotti, Frederick Cervenansky, Xin Yang, Pheng-Ann Heng, Irem Cetin, Karim Lekadir, Oscar Camara, Miguel Angel Gonzalez Ballester, et al. Deep learning techniques for automatic mri cardiac multi-structures segmentation and diagnosis: is the problem solved? *IEEE transactions on medical imaging*, 37(11):2514–2525, 2018.

[6] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[8] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[9] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[10] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[11] Ivan Krešo, Marin Oršić, Petra Bevandić, and Siniša Šegvić. Robust semantic segmentation with ladder-densenet models. *arXiv preprint arXiv:1806.03465*, 2018.

[12] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[13] Haoning Lin, Zhenwei Shi, and Zhengxia Zou. Maritime semantic labeling of optical remote sensing images with multi-scale fully convolutional network. *Remote sensing*, 9(5):480, 2017.

[14] Drew Linsley, Junkyung Kim, and Thomas Serre. Sample-efficient image segmentation through recurrence. *arXiv preprint arXiv:1811.11356*, 2018.

[15] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. In *Advances in Neural Information Processing Systems*, pages 152–164, 2018.

[16] Drew Linsley, D Schiebler, Sven Eberhardt, and Thomas Serre. Learning what and where to attend. In *Seventh International Conference on Learning Representations, New Orleans, LA Google Scholar Article Locations: Article Location Article Location*, 2019.

[17] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[18] David A Mély, Drew Linsley, and Thomas Serre. Complementary surrounds explain diverse contextual phenomena across visual modalities. *Psychological review*, 125(5):769, 2018.

[19] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, USA:, 2015.

[20] Christopher Olah. Understanding LSTM networks, 2015. *URL http://colah.github.io/posts/2015-08-Understanding-LSTMs*, 2015.

[21] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[22] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[23] Peng Peng, Karim Lekadir, Ali Gooya, Ling Shao, Steffen E Petersen, and Alejandro F Frangi. A review of heart chamber segmentation for structural and functional analysis using cardiac magnetic resonance imaging. *Magnetic Resonance Materials in Physics, Biology and Medicine*, 29(2):155–195, 2016.

[24] Stephen M Pizer, E Philip Amburn, John D Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B Zimmerman, and Karel Zuiderveld. Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, 39(3):355–368, 1987.

[25] P Radau, Y Lu, K Connelly, G Paul, A Dick, and G Wright. Evaluation framework for algorithms segmenting short axis cardiac mri. *The MIDAS Journal-Cardiac MR Left Ventricle Segmentation Challenge*, 49, 2009.

[26] Kathleen S Rockland and Jennifer S Lund. Intrinsic laminar lattice connections in primate visual cortex. *Journal of Comparative Neurology*, 216(3):303–318, 1983.

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[29] Larry Smarr, Camille Crittenden, Thomas DeFanti, John Graham, Dmitry Mishin, Richard Moore, Philip Papadopoulos, and Frank Würthwein. The pacific research platform: Making high-speed networking a reality for the scientist. In *Proceedings of the Practice and Experience on Advanced Research Computing*, pages 1–8. 2018.

[30] Courtney J Spoerer, Patrick McClure, and Nikolaus Kriegeskorte. Recurrent convolutional neural networks: a better model of biological object recognition. *Frontiers in psychology*, 8:1551, 2017.

[31] Dan D Stettler, Aniruddha Das, Jean Bennett, and Charles D Gilbert. Lateral connectivity and contextual interactions in macaque primary visual cortex. *Neuron*, 36(4):739–750, 2002.

[32] Dylan Uys. *Lightweight Deep Learning for Biomedical Image Segmentation*. PhD thesis, UC San Diego, 2019.

[33] Harvey D White, Robin M Norris, Michael A Brown, Peter W Brandt, RM Whitlock, and Christopher J Wild. Left ventricular end-systolic volume as the major determinant of survival after recovery from myocardial infarction. *Circulation*, 76(1):44–51, 1987.

[34] Richard Zhang. Making convolutional networks shift-invariant again. *arXiv preprint arXiv:1904.11486*, 2019.

[35] Yi Zhu and Shawn Newsam. Densenet for dense flow. In *2017 IEEE international conference on image processing (ICIP)*, pages 790–794. IEEE, 2017.

[36] Alex P Zijdenbos, Benoit M Dawant, Richard A Margolin, and Andrew C Palmer. Morphometric analysis of white matter lesions in mr images: method and validation. *IEEE transactions on medical imaging*, 13(4):716–724, 1994.