# L46: Investigations into Compressing Closed Form Continuous Time Networks

**Sharan Agrawal (sa2140)**

## Abstract

In Continuous time neural networks, state is a continuous function of time and its change is represented by differential equations. Typically, such networks require complex and slow ODE solvers to be applied at each step to calculate the next state. Closed Form Continuous Time (CfC) networks eliminates this need by using a closed form approximate solution to the state flow ODEs, and showed faster training speed and higher accuracy compared to state of the art RNNs. This study explored a number of approaches towards compressing such networks, adapting technqiues such as quantization and pruning to CfC architecture specific issues, and carried out empirical studies of the efficacy of various compression techniques and its effect on the CfC published benchmarks. Analytical insights about the quantization of CfC networks were also drawn.

## 1 Introduction

### 1.1 Objective

Closed Form Continuous Time networks have recently been proposed by Hasani (1) as extensions to Liquid Time Constant (LTC) networks that use closed form approximations to calculate state flow in continuous time networks, rather than ODE solvers. The technique devised showed extraordinarily fast training times vs. traditional discrete RNNs such as LSTMs, as well as LTC networks. It also showed better-than state of the art performance across a wide variety of benchmarks vs. all the neural architectures studied, all while using a relatively smaller parameter set.

The smaller training time and greater expressivity of the network already has promising implications on creating efficient and accurate smaller networks for resource-constrained compute environments. However, a study hasn't yet been done on the application of network compression techniques to this novel architecture. This study will apply compression techniques to the novel architectural elements used by the CfC network (such as 3 separate embedded neural networks) and carry out an empirical investigation of which approaches yielded the best result in maintaining accuracy whilst reducing the network's footprint.

### 1.2 Continuous Time Neural Networks

In Continuous Time Recurrent Neural Networks, state is a continuous function of time, and has state-dynamics represented by equation 1 (2).

$$\dot{\mathbf{x}}(t) = -\frac{1}{\tau}\mathbf{x}(t) + W\sigma(\mathbf{x}) + \mathbf{I}(t) \tag{1}$$

Here $\mathbf{x} \in \mathbb{R}^N$ are the states of each of the units, $\sigma(x_i(t))$ is the output of the $i$-th unit passed over an activation function ((2) uses a sigmoid function), $\tau$ is a time-constant impacting the equilibrium state the stable solution of the ODE reaches, $W \in \mathbb{R}^{M \times M}$ is a matrix of discrete weights connecting each unit, and $\mathbf{I}(t)$ is the input into the network. The latter term was can be more generally represented, as per (3), as some non-linear function of state and inputs, possibly a neural network with parameters $\theta$, as $f(\mathbf{x}(t), \mathbf{I}(t), \theta)$.

This model was further improved by (3) into a more expressive form called Liquid Time Constant networks, where the time constant becomes dependent on the input, allowing the capture of a wider class of features. Taking inspiration from the neural dynamics from certain smaller species, (3) recasts equation 1 as:

$$\dot{\mathbf{x}}(t) = -\frac{1}{\tau}\mathbf{x}(t) + f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)(A - \mathbf{x}(t)) \tag{2}$$

Similarly to equation 1, a neural network $f(\mathbf{x}(t), \mathbf{I}(t), t, \theta)$ is used to determine the gradients of the internal unit states. An ODE solver is used at each training step of the network to determine the states of the units from equation 2 since the equation does not have a known closed form solution yet (1). This use of the ODE solver is a major hurdle in the implementation of LTC networks, with drawbacks including its difficulty in scaling to large numbers of dimensions, or highly complex data and complexity of discretization techniques needed, and need for repeated solves during training reducing the fidelity with which the solver can be used at each step. Several optimizations exist such as those discussed in (4) to reduce the complexity in this step. A new, novel, approach was recently published by Hasani (1) which creates closed form approximations to the solutions of equations like equation 2 with verifiable bounds on accuracy. The use of closed form approximate solutions for the state flow equations means that ODE solvers no longer need to be used at each step in training the neural ODE, permitting a far greater training speed, whilst maintaining all the benefits of continuous time networks.

## 2 Related Work

Liebenwein, Hasani, et al (5) carried out an empirical investigation into pruning a particular kind of neural ODE called Continuous Normalizing Flow. They apply pruning using a variety of techniques to the neural network representing $f$ in equations 1 and 2. Their experiments found neural ODEs with pruning ratios of up to 98% without the loss of accuracy (5).

They carried out both structured and unstructured pruning and logged the loss changes from each independently, noting that structured pruning would be expected to and did perform worse than unstructured pruning in many examples due to it being a more constrained problem.

The authors also found that neural ODEs showed much better generalization properties. They found that pruning helped avoid modal collapse, and flattened the loss surface (5) allowing the neural ODE to find flatter minima, leading to better generalization. Conversely, the authors noted a decrease in the robustness of the neural ODE with increasing pruning.

## 3 Closed Form Continuous Time Networks

### 3.1 Overview of CfCs

The CfC Network proposed by Hasani et al (1) is built around using a tight closed form approximation for equation 2. This avoids the need for ODE solvers to calculate state flow over time. The approach adopted by LTCs and CfCs is inspired by biological neurons, shown in 1.

Figure 2 shows a network with 2 neurons, and one input $I(t)$, in ODE form, and solution form using the CfC approximation. Each edge represents an additive input into the state calculation of each neuron, with recurrent connections also being allowed, as seen in edge $S_{22}(t)$.

Another way to represent this network is to use a bit mask $m_i \in \mathbb{R}^K$ over each edge multiplied by the cell-state contribution from each connected neuron, transformed using $vec(.)$ to a $\mathbb{R}^K$ vector, in equation 3:

$$\begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{pmatrix} m_1 \\ m_2 \end{pmatrix} \times vec \begin{pmatrix} S_{00}(t) & S_{01}(t) & S_{02}(t) \\ S_{10}(t) & S_{11}(t) & S_{12}(t) \\ S_{20}(t) & S_{21}(t) & S_{22}(t) \end{pmatrix} \tag{3}$$

Formally, (1) proposed a solution to the following differential equation specifying state-flow in LTC networks from (3), which is a restatement of equation 2:

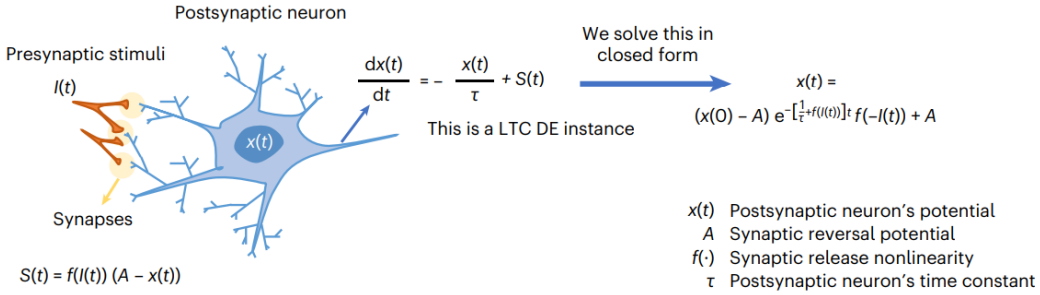


Figure 1: Synaptic dynamics inspiring LTC equation 2, figure from (1). Stimuli $I(t)$ drive the synaptic current $S(t)$ through a non-linear function $f(.)$ and a constant $A$. The dynamics of the state are given by the LTC DE, for which the authors provided an approximate solution.

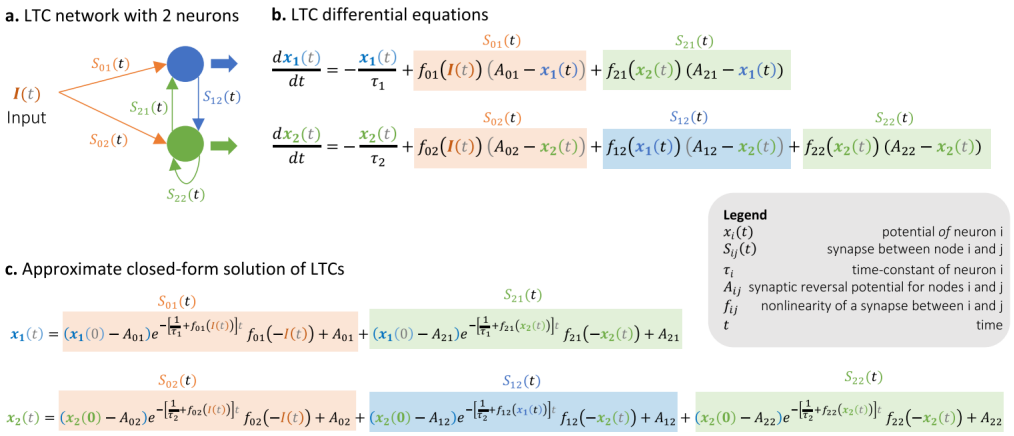


Figure 2: Illustration of a LTC network with 2 neurons and 5 edges, from (1)

$$\dot{\mathbf{x}} = -[w_\tau + f(\mathbf{x}, \mathbf{I}, \theta)] \odot \mathbf{x}(t) + A \odot f(\mathbf{x}, \mathbf{I}, \theta) \tag{4}$$

The approximate solution for a single dimensional time series input $I(t)$ with no self-connections was given (1) as:

$$x(t) \approx (x_0 - A)e^{-[w_\tau + f(I(t),\theta)]t} f(-I(t), \theta) + A \tag{5}$$

Hasani et al propose converting this into a practical network architecture with $f$ being a neural network, and $\mathbf{x}(t) \in \mathbb{R}^D$ hidden units, and $\mathbf{I}(t) \in \mathbb{R}^M$ as the input as:

$$\mathbf{x}(t) = B \odot e^{-[w_\tau + f(\mathbf{x}(t), \mathbf{I}(t); \theta)]t} \odot f(-\mathbf{x}(t), -\mathbf{I}(t); \theta) + A \tag{6}$$

The inclusion of $\mathbf{x}$ into the neural network $f$ denotes the possibility of having self-connections (1).

As can be seen, this model is subtly different to the model proposed by figure 2. Under the model in figure 2, the connections between CfC neurons occur explicitly through the bit masks applied to each edge. In the model proposed by (1), the state flow equation calculates the new hidden state of the network, and the neural network backbone maps not to an output per edge, but an output per cell-state.

Hasani et al (1) note that the approach in equation 6 has issues with gradient vanishing, driven by the negative exponential term. This was handled by replacing the exponential term with a sigmoid

transformation instead. It also has issues with hyperparameter specification given the number of bias terms present. To combat that, $B$ and $A$ were replaced with new neural networks $g(.)$ and $h(.)$. Finally the authors also noted that the sigmoid function transformation can be used to play a gating role with appropriate transformations.

Given these, the authors propose a final architecture:

$$\mathbf{x}(t) = \sigma(-f(\mathbf{x}, \mathbf{I}; \theta_f)t) \odot g(\mathbf{x}, \mathbf{I}, \theta_g) + [1 - \sigma(-f(\mathbf{x}, \mathbf{I}; \theta_f)t)] \odot h(\mathbf{x}, \mathbf{I}; \theta_g) \tag{7}$$

Here $f$, $g$ and $h$ represent neural networks underlying the state flow equation that must be trained. The authors propose to do this using a "backbone" network with network heads providing each of the neural network outputs (1). This allows shared learning of representations across the networks.

### 3.2 CfC Networks Analyzed

The above model will be the main focus of model compression techniques, and will be referred to as the **CfC** Network. The second architecture that will be examined will be the implementation of Neural Circuit Policy based wired CfC networks from Lechner et al (6). These are **CfC** networks without the backbone network, and more similar to the network in equation 3. This will be referred to as the **CfC-NCP** network. The third architecture, **CfC-Deep**, adds more layers to the backbone in **CfC** networks.

## 4 Model Compression Approaches

This section details how the model compression techniques can be applied to the networks defined above. All compression techniques will be carried out using PyTorch (7).

### 4.1 Quantization

Quantization reduces the memory footprint of a network, and increases the training and inference speed, by using lower precision numbers to represent the weights and activations of the network (8). There are numerous different design considerations that affect the quantization process, such as whether to retrain the network, which affect the applicability of the scheme. Uniform quantization will be used as the primary scheme. Other quantization schemes detailed in (9) but not used here include binarization (setting weights to their signs), product quantization where quantization is done on disjoint subspaces of the vectorized weights, scalar quantization using k-means clustering where weights are clustered and a quantized reference to the cluster is used instead.

#### 4.1.1 Analytical Insights

Given the analytical structure of the CfC network's state, under some simplifying assumptions, analytical calculations on quantization error can be carried out to determine optimal approaches to quantization. This uses approaches inspired by (10)'s approach to optimal quantization through analysis of MSE. Assume that a numerical value, $W$, can be quantized to $\tilde{W}$. The quantization error, or machine epsilon $\epsilon$, can be defined as $\epsilon = (\tilde{W} - W)/W$, which implies that $\tilde{W} = W(1 + \epsilon)$.

A simplifying assumption is made that this can be scaled to tensors $\boldsymbol{W}$ with a constant, scalar $\epsilon$, which is true if the quantized tensor has the same range as the original. Under this assumption, applying quantization to the weights of a linear, fully connected network defined as $f(\boldsymbol{X}; \boldsymbol{W}) = \boldsymbol{W}_L(\boldsymbol{W}_{L-1}(...\boldsymbol{W}_1\boldsymbol{X}))$ yields an expression for the quantized network output, where $p^L(\epsilon)$ is a polynomial of order $L$ excluding the constant term:

$$\tilde{f}(\boldsymbol{X}; \tilde{\boldsymbol{W}}) = (1 + p^L(\epsilon))f(\boldsymbol{X}; \boldsymbol{W}) \tag{8}$$

In reality the quantization error should exist for each element in $\boldsymbol{W}$, where similar logic could also be applicable but hasn't been explored yet. Now let's examine the simplified setting of equation 6 assuming that the embedded network $f = f(\boldsymbol{I}(t); \theta)$ is a linear, fully connected network with no self connections, then the quantized cell state can be written as:

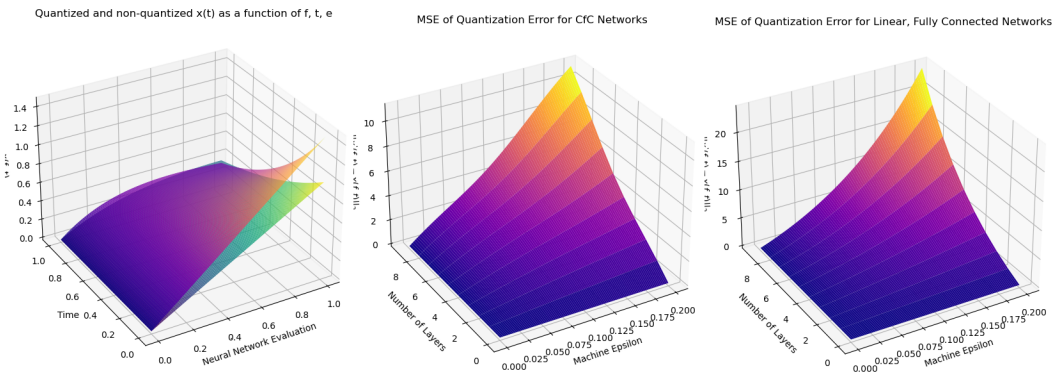


Figure 3: [Left] $x(t)$ with quantized (red) and non-quantized (green) models for a constant $\epsilon$. [Middle, Right] The MSE between the surfaces over layers and $\epsilon$

$$\tilde{\boldsymbol{x}}(t) = \boldsymbol{x}(t) + (\boldsymbol{x}(t) - A) \odot (p^L(\epsilon) e^{-p^L(\epsilon)\boldsymbol{f}t} + e^{-p^L(\epsilon)\boldsymbol{f}t} - 1) \tag{9}$$

Derivations are given in the appendix. The LHS of figure 3 shows the state surfaces for one-dimensional values of $f$ and $t$ with quantized and regular state. The MSE between $x(t)$ and $\tilde{x}(t)$ is given on the RHS. This clearly shows that the quantization error increases super-linearly with the machine precision $\epsilon$ and the number of layers $L$. This motivates one approach for heavy quantization: to reduce the number of hidden layers in $f$ when applying heavy quantization. Another observation is that for some configurations of CfC networks with non-zero $w\tau$, CfC networks seem to have less error than linear fully connected networks in this model, so CfC networks could good candidates for quantization.

The MSE error formulation can be used in combination with a uniform quantization scheme with 0 offset, where a quantized tensor can be represented as $\tilde{\boldsymbol{W}} = \alpha \hat{\boldsymbol{W}}$ and $\hat{\boldsymbol{W}}$ is the projection of $\boldsymbol{W}$ into the quantized space. Given the definition of $\epsilon$, this implies that $\epsilon = \frac{\alpha \hat{W} - W}{W}$, which can be used within equation 9 to determine the scale factor that minimizes the MSE.

### 4.1.2 Computational Implementation

PyTorch supports 3 different modes of quantization, all aimed at compressing the precision of weight and activation matrices to INT8 instead of FP32 (11). Two of those will be used in this study:

1. **Static**: all weights and activations are quantized to INT8 statically post-training for inference
2. **Quantization Aware Training**: all weights and activations are quantized to INT8 statically and stored as such

Static quantization, and quantization aware training will be applied to minimize the model size, and the impact on accuracy and inference time will be assessed. Operator fusing will also be used between the linear and activation layers of the embedded networks where possible.

## 4.2 Pruning

Pruning removes neurons or connections that don't significantly contribute to overall accuracy (12). Pruning techniques can be broadly categorized as unstructured and structured (5). Unstructured pruning (12) prunes individual weights from the global model or a local layer. This can be done using techniques like $l_1$ norm filtering, where all weights $||W||_1 <= \alpha$ are removed from the model at each step. Structured pruning, as described in (13) removes all weights associated with a filter or neuron based on the $l_1$-norm of all weights associated with the neuron. Numerous studies have shown that pruning often not only maintains the accuracy of the trained model (13), but can also lead to better generalization out of sample (5), making pruning a very powerful technique for network compression.

| Baseline Network Results | | | | |
|---|---|---|---|---|
| Benchmark | Network Type | Accuracy | Time per Epoch (s) | Number of RNN Parameters |
| Person Activity | **CfC** (1) | 82.8% $\pm$0.7% | 7.6 $\pm$0.4 | 151k |
| IMDB | **CfC** (1) | 88.2% $\pm$0.8% | 2.8 $\pm$0.3 | 75k |
| IMDB | **CfC-Deep** (1) | 88.1% $\pm$0.4% | 3.2 $\pm$0.4 | 551k |
| IMDB | **CfC-NCP** (6) | 84.4% $\pm$0.6% | 2.6 $\pm$0.4 | 37k |

Table 1: Accuracy and training time per baseline used. Errors are taken as stddev over 5 runs.

| Category | Benchmark | Model | Baseline | Static Quantization | Operator Fusing | QAT |
|---|---|---|---|---|---|---|
| Accuracy | Person Activity | CfC (Hasani) | 87.2% | 86.4% | 86.4% | 86.5% |
| | IMDB | CfC (Hasani) | 86.1% | 86.0% | 86.1% | 86.8% |
| | IMDB | CfC (Hasani) + Deep | 88.1% | 87.2% | 87.1% | 87.6% |
| | IMDB | CfC NCP (Lechner) | 84.2% | 83.8% | - | 83.9% |
| Compression Ratio | Person Activity | CfC (Hasani) | 0.6MB | 3.1x | 3.1x | 3.1x |
| | IMDB | CfC (Hasani) | 15.6MB | 3.8x | 0.9x | 3.8x |
| | IMDB | CfC (Hasani) + Deep | 17.6MB | 3.8x | 3.8x | 3.8x |
| | IMDB | CfC NCP (Lechner) | 5.3MB | 3.8x | 3.8x | 3.8x |
| Inference Time* | Person Activity | CfC (Hasani) | 0.9s | 1.8x | 1.9x | 1.8x |
| | IMDB | CfC (Hasani) | 2.3s | 1.9x | 2.0x | 1.9x |
| | IMDB | CfC (Hasani) + Deep | 5.1s | 2.7x | 3.2x | 2.7x |
| | IMDB | CfC NCP (Lechner) | 1.5s | 1.9x | - | 1.9x |

Figure 4: Quantization Experimental Results. Inference time results from CPU backend.

For the **CfC** and **CfC-Deep** networks, pruning is done over the backbone network and network heads. In the CfC-NCP network, pruning is applied on all the individual $f, g$ and $h$ networks.

### 4.3 Knowledge Distillation

Knowledge distillation (14) and (15) uses a pre-trained large, slow network to train a smaller but more efficient network. The technique proposed by (15) involved creating cast amounts of training pseudo-data by using the trained, large model to classify fake data and use it to train the smaller model. More modern techniques train the smaller model in tandem with the larger model with frozen parameters, minimizing a shrinkage of a join and individual loss function. This method is applied here to CfC networks given by equation 7 by training a model with a large backbone network, and then distilling the shared representation learnt across the network heads into 3 smaller networks for $f, g$ and $h$.

## 5 Empirical Results

### 5.1 Baseline

We use 2 different benchmarks adopted and modified from (1). The Human Activity benchmark is a hard task using the RNN features of the network and is slow to train. Only the **CfC** network was used here. The IMDB benchmark (16), while not a recurrent task, was used across all 3 networks to provide a standard baseline for comparison. The network parameters are in the appendix.

### 5.2 Quantization

Static weight and activation quantization to int8 tensors were applied across all assessed networks. Operator fusing amongst some of the deep layers of the network were explored for its impact on inference speed. Finally, Quantization Aware Training was explored to improve accuracy whilst maintaining compression gains. The impact on accuracy, compression ratio and inference time was explored. The results are in 4.

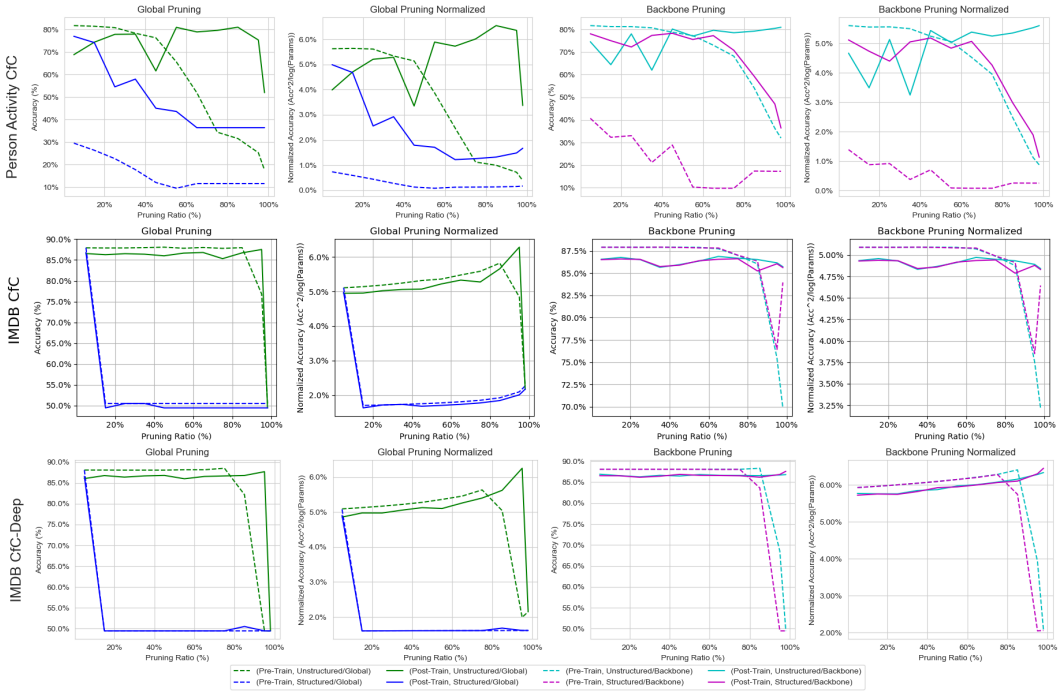


Figure 5: Results of pruning applied across CfC network architectures and benchmarks.

### 5.3 Pruning

Structured and unstructured prunining was applied both globally, as well as locally to just the backbone section of the network. The results are in figure 5.

Pruning ratios varied from 10% to 98%, and the impact on accuracy was assessed. In each case, the effect on accuracy of pruning without training was recorded, as well as post-training with 10 further epochs. Finally, in order to view the best trade-off between accuracy and compression ratio, and given that in local pruning the overall network compression ratio will be much smaller than the pruning ratio, a "normalized accuracy" metric was developed and used: $norm\_acc = \frac{accuracy^{\gamma}}{log(num\_params)}$. We use $\gamma = 2$. This shows an "accuracy$^{\gamma}$ per log parameter" metric, which is better at showing how "efficient" the pruning has been.

### 5.4 Knowledge Distillation

The **CfC** network has 3 network heads $(f, g, h)$ supported by a backbone network. This architecture allows for learning shared representations across the 3 network heads, which the CfC-NCP network cannot do. However, this leads to a bigger model. Knowledge distillation of the CfC-Deep network trained on the IMDB benchmark was carried out into a CfC-NCP network with just individual network heads $f, g, h$ with no backbone. The hypothesis is that the shared representation learnt by the backbone can be distilled into much smaller network heads once learnt and still achieve similar accuracy. Knowledge distillation was carried out using a shrinkage based loss function: $loss = 0.25 \times lossfn(student\_preds, target) + 0.75 \times lossfn(student\_preds, teacher\_preds)$. The results are shown in 6. The counterfactual networks (trained only on the data without distillation) often failed to train, or trained with substantially ( 5-10%) smaller accuracies.

## 6 Discussion

Through quantization, strong compression ratios up to 3.8x are achieved without significantly compromising accuracy across most network architectures, especially after retraining using QAT.

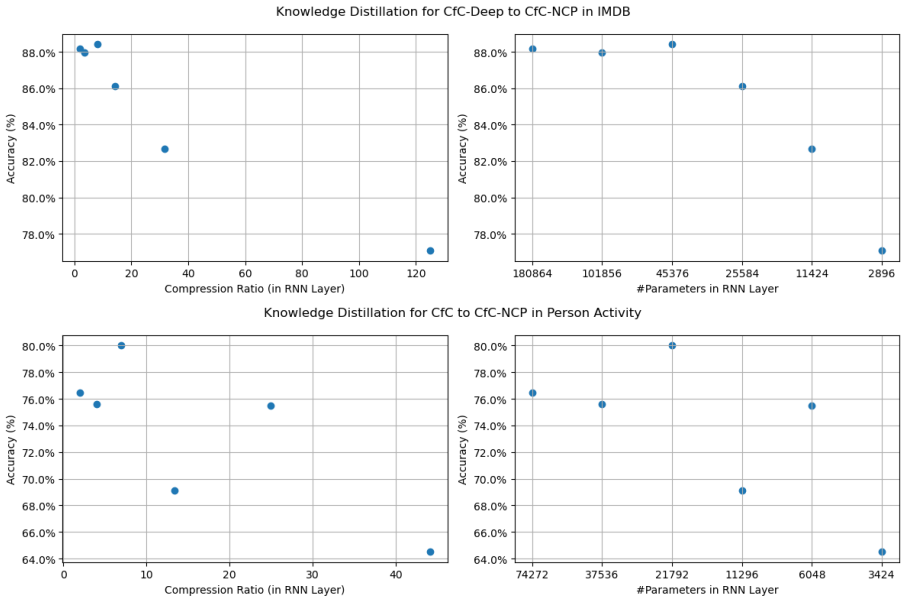


Figure 6: Results of Knowledge Distillation of CfC to CfC-NCP Networks.

This is consistent with the theoretical analysis that CfC networks should fare well under quantization. Consistent with figure 3, the **CfC** network fared better under quantization than the **CfC-Deep** network that had more layers. Operator fusing also significantly improved inference speeds without compromising accuracy.

Unstructured pruning also showed significant promise. In the Person Activity RNN benchmark, global unstructured pruning showed the best performance, with a pruning ratio up to 95% showing similar accuracy to the originally trained model, after retraining. Just pruning the backbone showed similar performance, but less size-adjusted compression vs. global pruning. In the IMDB benchmark, structured global pruning performed poorly, but structured pruning on the benchmark performed similarly to unstructured. In both cases, compression ratios up to 95% globally and 98% on the backbone were achieved. The efficacy of backbone pruning shows that CfC networks can be resilient to lower fidelity specification of the embedded networks.

Distilling knowledge gained from the backbone network into individual network heads proved to be a very effective structure. Both across IMDB and Person Activity benchmarks, compression ratios of 8-15x were achievable without any loss in accuracy. This shows that learning the shared representation with the backbone network, and then distilling it into individual network heads is an effective strategy, and shared distillation learning across the separate networks is possible. However, compression ratios >15x led to significant falls in accuracy, showing that the individual network heads still need significant parameters to learn that additional complexity. Combining pruning and this approach to knowledge distillation could be a very effective strategy for further compression, if the pruned backbone can distill a simpler but still shared representation across features to the individual networks.

## 7 Conclusion

Overall all three compression techniques explored proved effective with CfC network variations. Consistent with theoretical insights, CfC networks did well with quantization, particularly with fewer layers in the backbone. Operator fusing in the backbone improved inference speed. Unstructured global pruning proved to be an effective strategy across all networks, with pruning ratios up to 95%, consistent with the findings in (5). Distilling shared representation learnt by a backbone network into smaller individual network heads also proved effective with ratios up to 8-10x.

# 8 Appendix

## 8.1 Quantization Analytical Insights Derivation

### 8.1.1 Linear Neural Network Quantization Error

Let's look at a fully connected network with $L$ layers with no bias and no activation layer, i.e. $f(\boldsymbol{X};\boldsymbol{W}) = \boldsymbol{W}_L(\boldsymbol{W}_{L-1}(...\boldsymbol{W}_1\boldsymbol{X}))$. Let's assume that weights $\boldsymbol{W}$ are being quantized, and make the significant simplifying assumption that the machine epsilon can be represented as a scalar, then the quantized weight tensor would be $\tilde{\boldsymbol{W}} \approx \boldsymbol{W}(1+\epsilon)$. Let's assume this approximation can be taken as exact.

Now, the output of a quantized FC network can be written as $f(\boldsymbol{X};\tilde{\boldsymbol{W}}) = \tilde{\boldsymbol{W}}_L(\tilde{\boldsymbol{W}}_{L-1}(...\tilde{\boldsymbol{W}}_1\boldsymbol{X}))$. Let's look at the 2 layer case for simplicity. Then:

$$f(\boldsymbol{X},\tilde{\boldsymbol{W}}) = \tilde{\boldsymbol{W}}_{\boldsymbol{2}}(\tilde{\boldsymbol{W}}_1\boldsymbol{X}) = \boldsymbol{W}_{\boldsymbol{2}}(1+\epsilon)(\boldsymbol{W}_{\boldsymbol{1}}(1+\epsilon)\boldsymbol{X}) = \boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{X} + 2\epsilon\boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{X} + \epsilon^2\boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{X}$$

Extending this to $L$ layers, let $p^L(\epsilon)$ be the $L$ degree polynomial in $\epsilon$ minus 1, then

$$f(\boldsymbol{X},\tilde{\boldsymbol{W}}) = (1+p^L(\epsilon))f(\boldsymbol{X};\boldsymbol{W})$$

.

### 8.1.2 Expanding to Multidimensional $\epsilon$

There is an avenue to expanding this to multidimensional $\epsilon$ with matrix polynomials. In this case, the output of the quantized network can be written as:

$$f(\boldsymbol{X},\tilde{\boldsymbol{W}}) = \boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{X} + \boldsymbol{W}_2(\boldsymbol{W}_1 \odot \boldsymbol{\epsilon_1})\boldsymbol{X} + (\boldsymbol{W}_2 \odot \boldsymbol{\epsilon_2})\boldsymbol{W}_1\boldsymbol{X} + (\boldsymbol{W}_2 \odot \boldsymbol{\epsilon_2})(\boldsymbol{W}_1 \odot \boldsymbol{\epsilon_1})\boldsymbol{X}$$

The original network terms can be factored out:

$$f(\boldsymbol{X},\tilde{\boldsymbol{W}}) = \boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{X} \odot (1 + \boldsymbol{W}_2\boldsymbol{\epsilon}_1\boldsymbol{X} + \boldsymbol{\epsilon}_2\boldsymbol{W}_1\boldsymbol{X} + \boldsymbol{W}_2\boldsymbol{\epsilon}_1\boldsymbol{X} \odot \boldsymbol{\epsilon}_2\boldsymbol{W}_1\boldsymbol{X} \odot \boldsymbol{\epsilon}_2\boldsymbol{\epsilon}_1\boldsymbol{X})$$

It is not certain that this can be easily factored given there are still terms dependent on weights and inputs in the polynomial side, but if it can it can yield a more accurate approximation and calibration for the scaling factor.

### 8.1.3 CfC Network Quantization Error

This will proceed from equation 6. Denote $f(\boldsymbol{X}(t),\boldsymbol{I}(t);\theta)$ as $\boldsymbol{f}$, and say it is a fully connected network with no bias as per above. Then the quantized form of equation 6 (in one dimension) can be written as:

$$\tilde{x}(t) = Ce^{-(\omega_\tau + (1+p^L(\epsilon))f}t(1+p^L(\epsilon))f + A$$

Expanding this equation, we get:

$$\tilde{x}(t) = Ce^{-(\omega_\tau+f)t}e^{-p^L(\epsilon)ft}f + Ce^{-(\omega_\tau+f)t}e^{-p^L(\epsilon)ft}p^L(\epsilon)f + A$$

Note that $Ce^{-(\omega_\tau+f)t}f = x(t) - A$

Factoring out the terms dependent on the quantization error and substituting in 6, then:

$$\tilde{x}(t) = (p^L(\epsilon)e^{-p^L(\epsilon)ft} + e^{-p^L(\epsilon)ft} - 1)(x(t) - A) + x(t)$$

This can easily be expanded to higher dimensional states.

The plots in figure 3 were generated by assuming $f, t \in [0,1]$, and $w_\tau = 1, C = 1, A = 0$, for both CfC networks and FC networks.

| | Person Activity | IMDB | | |
|---|---|---|---|---|
| Parameter | CfC | CfC | CfC-NCP | CfC-Deep |
| epochs | 100 | 10 | 10 | 10 |
| clipnorm | - | 10 | 128 | 10 |
| hidden size | 448 | 192 | 192 | 192 |
| base_lr | 0.02 | 0.002 | 0.004 | 0.002 |
| decay_lr | 0.97 | 0.96 | 0.97 | 0.96 |
| backbone_activation | RELU | RELU | - | RELU |
| backbone_units | 64 | 64 | - | 256 |
| backbone_dr | 0 | 0 | - | 0.1 |
| backbone_layers | 1 | 1 | - | 5 |
| weight_decay | 3.00E-05 | 3.60E-05 | 3.60E-05 | 3.60E-05 |
| optimizer | AdamW | Adam | Adam | Adam |
| batch_size | 64 | 128 | 128 | 128 |
| tau | 10 | - | - | - |
| init | 0.84 | - | - | - |
| embed_dim | - | 192 | 192 | 192 |
| embed_dr | - | 0 | 0 | 0 |

Figure 7: Parameters for network architectures tested, mostly from (1)

### 8.2 Network Parameters

## References


[1] R. Hasani, M. Lechner, A. Amini, L. Liebenwein, A. Ray, M. Tschaikowski, G. Teschl, and D. Rus, "Closed-form continuous-time neural networks," *Nature Machine Intelligence*, vol. 4, pp. 992–1003, Nov. 2022.

[2] K.-i. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural Networks*, vol. 6, no. 6, pp. 801–806, 1993.

[3] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu, "Liquid Time-constant Networks," 2020.

[4] S. Massaroli, M. Poli, J. Park, A. Yamashita, and H. Asama, "Dissecting neural odes," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3952–3963, 2020.

[5] L. Liebenwein, R. Hasani, A. Amini, and D. Rus, "Sparse Flows: Pruning Continuous-depth Models," 2021.

[6] M. Lechner, R. Hasani, A. Amini, T. A. Henzinger, D. Rus, and R. Grosu, "Neural circuit policies enabling auditable autonomy," *Nature Machine Intelligence*, vol. 2, pp. 642–652, Oct. 2020.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and others, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[8] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit Quantization of Neural Networks for Efficient Inference," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 3009–3018, 2019.

[9] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing Deep Convolutional Networks using Vector Quantization," 2014.

[10] R. Chisholm, P. Richmond, and S. Maddock, "A Standardised Benchmark for Assessing the Performance of Fixed Radius Near Neighbours," Euro-Par 2016 Workshops, pp. 311–321, 2017.

[11] R. Krishnamoorthi, J. Reed, M. Ni, C. Gottbrath, and S. Weidman, "Introduction to Quantization on PyTorch," Mar. 2020.

[12] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both Weights and Connections for Efficient Neural Networks," 2015.

[13] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient ConvNets," 2016.

[14] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," 2015.

[15] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 535–541, 2006.

[16] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.