Abstract—Recently, Capsule Networks (CapsNets) have shown improved performance compared to the traditional Convolutional Neural Networks (CNNs), by encoding and preserving spatial relationships between the detected features in a better way. This is achieved through the so-called Capsules (i.e., groups of neurons) that encode both the instantiation probability and the spatial information. However, one of the major hurdles in the wide adoption of CapsNets is their gigantic training time, which is primarily due to the relatively higher complexity of their new constituting elements that are different from CNNs.

In this paper, we implement different optimizations in the training loop of the CapsNets, and investigate how these optimizations affect their training speed and the accuracy. Towards this, we propose a novel framework FasTrCaps that integrates multiple lightweight optimizations and a novel learning rate policy called WarmAdaBatch (that jointly performs warm restarts and adaptive batch size), and steers them in an appropriate way to provide high training-loop speedup at minimal accuracy loss. We also propose weight sharing for capsule layers. The goal is to reduce the hardware requirements of CapsNets by removing unused/redundant connections and capsules, while keeping high accuracy through tests of different learning rate policies and batch sizes. We demonstrate that one of the solutions generated by the FasTrCaps framework can achieve 58.6% reduction in the training time, while preserving the accuracy (even 0.12% accuracy improvement for the MNIST dataset), compared to the CapsNet by Google Brain [25]. Moreover, the Pareto-optimal solutions generated by FasTrCaps can be leveraged to realize trade-offs between training time and achieved accuracy. We have open-sourced our framework on GitHub.

Index Terms—Machine Learning, Capsule Networks, Training, Accuracy, Efficiency, Performance, Weight Sharing, Decoder, Batch Sizing, Adaptivity.

I. INTRODUCTION

The development of Deep Neural Networks (DNNs), especially the Convolutional Neural Networks (CNNs), has experienced a dramatic increase in the past decade, leading to many different architectures [16][25]. A key problem is the optimization of CNNs and their hyper-parameters, for which, most of the fine-tuning optimizations (e.g., choosing the training policy, the learning rate, the optimizer, etc.) are repetitive and time-consuming, because every change must be tested with many epochs of training and repeated many times to have certain statistical significance. This is significantly worsened when increasing the CNN complexity, which leads to a more demanding compute effort to find the right set of optimizations.

Different methods have been explored in the literature to reduce the training time of CNNs, such as one-cycle policy [31], warm restarts [13], and adaptive batch sizing [6][7][8]. However, CNNs have a key limitation: they do not retain information on the spatial correlation between the detected features. This effect causes poor network performances in terms of accuracy when the object to be recognized is rotated, has a different orientation, or presents any other geometrical variation. Currently, this problem is solved by training CNNs on expanded large-sized datasets, that include also transformed and modified objects. However, wider datasets lead to much longer training times, which not only pose delays in the DNN development cycle, but also require (1) super-costly training machines like Nvidia’s DGX-2 or DGX Superpod rendering it unaffordable for many developers/organizations, or (2) outsourcing to third party cloud services that can compromise privacy and security requirements [27][34]. Hence, both advanced DNN architectures and fast training techniques are necessary to mitigate the above challenges.

Capsule Networks (CapsNets) by Google [25] aim at overcoming the limitations of CNNs w.r.t. preserving the spatial correlation between the detection features through the following two means: (1) by substituting single neurons with the so-called capsules (i.e., groups of neurons); and (2) using the dynamic routing algorithm between the capsule layers that provides the ability to encode both the instantiation probability of an object and its instantiation parameters (width, orientation, skew, etc.). However, one of the major hurdles in the wide-scale adoption of CapsNets is their gigantic training time, which is primarily due to the higher complexity of their new constituting elements.

Motivational Case Study and Target Research Problem:
Our analyses in Fig. [1] show the available improvement potential and opportunities for the CapsNets, when different optimizations like Exponential Decay (ExpDecay), One-Cycle-Policy (OCP) [31], Warm Restarts (WR) [13] and Adaptive Batch Size (AdaBatch) [6] are applied. These optimizations result in a higher accuracy and a lower number of epochs.
to reach the maximum accuracy, w.r.t. the baseline (Fixed) learning rate policy. This motivates us to tailor these optimizations towards the new features of CapsNets, and implement them in an integrated framework to train CapsNets in a fast yet accurate manner, while also reducing the number of CapsNet parameters, for instance, through weight sharing.

![Graph showing optimization potentials when considering CapsNet.](image)

**Fig. 1: Optimization potentials when considering CapsNet.**

### Our Novel Contributions and Concept Overview [Fig. 2]:
We present FasTrCaps, a framework which employs different optimization methods for significantly reducing the training time and the number of parameters of CapsNets, while preserving or improving their accuracy.

![Flowchart showing an overview of our novel contributions in this paper.](image)

**Fig. 2: An overview of our novel contributions in this paper.**

The key contributions are:
- Tailoring different learning rate policies (like one-cycle policy or warm restarts) to specialize them for the CapsNet structure, and analyzing their efficiency in the CapsNet training loop vs. the corresponding training time (Section III).
- A novel training framework, FasTrCaps, which accelerates the training of CapsNets by integrating the above-described optimizations (like warm restarts, adaptive batch size and weight sharing) in an automated flow specialized to the structure and training flow of the CapsNets (i.e., considering the capsules and the coupling between capsule layers) (Section IV).
- Parameter reduction via weight sharing and reducing the size of the decoder for CapsNets by removing its unused connections. These optimizations reduce the number of parameters by more than 15% (Section IV-C & IV-D).
- Evaluation of the FasTrCaps framework on the MNIST and Fashion-MNIST datasets to stay compliant with the experimental setup of [25] (Section V).
- Open-source release of our FasTrCaps framework, for reproducibility and to facilitate further research and development in the area of accelerated training of CapsNets, on [GitHub](https://github.com).

Another key benefit of our FasTrCaps framework is that it provides multiple Pareto-optimal solutions that can be leveraged to trade-off training time reductions without (or in some cases with a user-tolerable) accuracy loss. For instance, in our experimental evaluations, FasTrCaps provides a solution that can reduce the training time of CapsNets by 58.6% while providing a slight increase in its accuracy (i.e. 0.12%). Another solution provides 15% reduction in the number of parameters of the CapsNet without affecting its accuracy. Note, our methodology may also be beneficial for other complex CNNs, as it enables integration of available optimizations for training.

Before proceeding to the technical sections, we present an overview of CapsNets and the learning rate policies in Section II to a level of detail necessary to understand our contributions.

### II. BACKGROUND AND RELATED WORK

#### A. CapsNets

The Capsule introduced by Hinton [10] denotes a group of neurons encoding both the instantiation probability of an object and the spatial information. The works in [25] and [9] introduced two architectures based on capsules, tested on the MNIST dataset [12], with a classification accuracy aligned to the state-of-the-art traditional CNNs for the same application. The CapsNet that we use in our work corresponds to the model of [25], as shown in Fig. 3. The layers constituting the encoder are:

- **ConvLayer**: an initial convolutional layer.
- **PrimaryCaps**: a layer that transforms the scalar numbers of ConvLayer in vectors.
- **DigitCaps**: a layer that performs dynamic routing and computes the output probabilities.

The encoding network is followed by a decoder, composed of three fully-connected layers, which output the reconstructed image. The loss computed on the reconstructed image (i.e., the Reconstruction Loss) directs the capsules of the DigitCaps layer to encode the instantiation parameters of the object.

Compared to traditional CNNs, CapsNets have shown high robustness against adversarial attacks [22] and affine transformations [18]. These results contribute towards employing CapsNets in safety-critical applications. Moreover, from the computational perspective, CapsNets can be efficiently executed on IoT/Edge devices with the usage of specialized hardware [14] [17]. Further energy savings can be achieved by applying quantization [19] or approximate multipliers [20].

#### B. An Overview of Two Key Learning Rate Policies

The learning rate (LR) is a relevant hyperparameter for the fast convergence during the training of a neural network. With a wide learning rate the optimization process may stop in a wide learning rate the optimization process may stop in a

- **ConvLayer**: an initial convolutional layer.
- **PrimaryCaps**: a layer that transforms the scalar numbers of ConvLayer in vectors.
- **DigitCaps**: a layer that performs dynamic routing and computes the output probabilities.

The encoding network is followed by a decoder, composed of three fully-connected layers, which output the reconstructed image. The loss computed on the reconstructed image (i.e., the Reconstruction Loss) directs the capsules of the DigitCaps layer to encode the instantiation parameters of the object.

Compared to traditional CNNs, CapsNets have shown high robustness against adversarial attacks [22] and affine transformations [18]. These results contribute towards employing CapsNets in safety-critical applications. Moreover, from the computational perspective, CapsNets can be efficiently executed on IoT/Edge devices with the usage of specialized hardware [14] [17]. Further energy savings can be achieved by applying quantization [19] or approximate multipliers [20].

- **ConvLayer**: an initial convolutional layer.
- **PrimaryCaps**: a layer that transforms the scalar numbers of ConvLayer in vectors.
- **DigitCaps**: a layer that performs dynamic routing and computes the output probabilities.

The encoding network is followed by a decoder, composed of three fully-connected layers, which output the reconstructed image. The loss computed on the reconstructed image (i.e., the Reconstruction Loss) directs the capsules of the DigitCaps layer to encode the instantiation parameters of the object.

Compared to traditional CNNs, CapsNets have shown high robustness against adversarial attacks [22] and affine transformations [18]. These results contribute towards employing CapsNets in safety-critical applications. Moreover, from the computational perspective, CapsNets can be efficiently executed on IoT/Edge devices with the usage of specialized hardware [14] [17]. Further energy savings can be achieved by applying quantization [19] or approximate multipliers [20].
The cosine annealing function is given in Equation 2, where it is again set to the maximum value, realizing a step function.

Warm Restarts \[13\] (aka warm restarts) the formulas of the three phases of the one-cycle policy value in a small fraction of the last steps. Equation 1 reports in phase-3, the learning rate must be annealed to a very low range. In phase-2, the learning rate is symmetrically decreased.

In phase-1, the learning rate is linearly increased from a minimum to a maximum value in an optimal training time, where the cycle period needs to be set properly. Equation 1 reports

\[ \begin{align*}
\text{lr} &= \text{lr}_{\text{min}} + ts \cdot \frac{\text{lr}_{\text{max}} - \text{lr}_{\text{min}}}{T_i} & 0 < ts < 0.45 \cdot T_i \quad \text{phase-1} \\
\text{lr} &= \text{lr}_{\text{min}} + (ts - 0.9 \cdot TS) \cdot \frac{\text{lr}_{\text{max}} - \text{lr}_{\text{min}}}{0.45TS < ts < 0.9TS} & \text{phase-2} \\
\text{lr} &= \text{lr}_{\text{min}} - 0.9 \cdot \frac{\text{lr}_{\text{max}} - \text{lr}_{\text{min}}}{T_i} \cdot (ts - 0.9 \cdot TS) & 0.9 \cdot TS < ts < T_i \quad \text{phase-3}
\end{align*} \tag{1} \]

**Warm Restarts**: In the Stochastic Gradient Descent with Warm Restarts \[13\] (aka warm restarts), the learning rate is initialized to a maximum value and then it is decreased with cosine annealing until reaching the lower bound of a chosen interval. When the learning rate reaches the minimum value, it is again set to the maximum value, realizing a step function. The cosine annealing function is given in Equation 2 where $lr_{\text{min}}$ and $lr_{\text{max}}$ are the learning rate range boundaries, $ts$ is the training step, $T_i$ is the number of training steps for each cycle. When $ts = T_i$, $ts$ is set to 0 and the cycle starts again. This process is repeated cyclically during the whole training time, where the cycle period needs to be set properly to optimize the training time and accuracy. Increasing the learning rate step-wise emulates a warm restart of the network and encourages the model to step out from possible local minima or saddle points.

\[ lr = lr_{\text{min}} + \frac{1}{2} \left( lr_{\text{max}} - lr_{\text{min}} \right) \left( 1 + \cos \left( \frac{\pi \cdot ts}{T_i} \right) \right) \tag{2} \]

Recently, a novel warm restart technique was proposed in \[23\], where the learning rate is decreased with a polynomial function after each restart.

**C. Adaptive Batch Size**

Training a DNN with a small batch size can provide a faster convergence \[11, 21\], while a larger batch size allows to have an higher data parallelism and, consequently, high computational efficiency. Hence, many authors have studied methods to increase the batch size with fixed policies \[11, 5\] or following an adaptive criterion, with the so-called Adaptive Batch Size \[6, 7, 8\].
III. ANALYZING THE EFFICIENCY OF LEARNING RATE POLICIES FOR CAPSNETS

The techniques described in Section II have been tailored for traditional CNNs, to improve their performances in terms of accuracy and training time. This section aims to customize different learning rate policies and batch size selection for training the CapsNets considering the multidimensional capsules and their cross-coupling, and to study whether and how much these policies are effective. Since the traditional neurons of the CNNs are replaced by capsules in the CapsNets, the number of parameters (weights and biases) to be trained is huge.

For this purpose, we implemented different state-of-the-art learning rate policies for the training loop of the CapsNet, such that these techniques are enhanced for the capsule structures and relevant parameters of the CapsNet (see discussion in Section IV). Fig. 4 (a) shows the learning rate changes for different techniques and how the accuracy of the CapsNet for the MNIST dataset varies accordingly. More detailed results of our analyses, including the comparisons with the LeNet5, are reported in Fig. 4 (b).

From this analysis, we derive the following key observations:
1) The warm restarts technique is the most promising one because it allows to reach the same accuracy (99.37%) as the CapsNet with a fixed learning rate, while providing a reduction of 79.31% in the training time.
2) A more extensive training with warm restarts leads to an accuracy improvement of 0.07%.
3) The adaptive batch size shows similar improvements in terms of accuracy (99.41%) and training epochs.
4) The first epochs with smaller batch sizes execute relatively longer when compared to the ones with bigger batch sizes.

IV. FasTrCaps: OUR FRAMEWORK TO ACCELERATE THE TRAINING OF CAPSNETS

Training a CapsNet consists of a multi-objective optimization problem, because our scope is to maximize the accuracy, while minimizing the training time and the network complexity. A comprehensive processing flow of our FasTrCaps framework is shown in Fig. 5. Before describing how to integrate different optimizations in an automated training methodology and how to generate the optimized CapsNet at the output (Section IV-E), we present how these optimizations have been implemented with enhancements for the CapsNets, which is necessary to realize an integrated training framework.

A. Learning Rate Policies for CapsNets

The first parameter analyzed to improve the training process of CapsNets is the learning rate. The optimal learning rate range is evaluated within the range boundaries 0.0001 and 0.001. For our framework, we use the following parameters in these learning rate policies:

- **Fixed learning rate**: 0.001
- **Exponential decay**: starting value 0.001, decay rate 0.96, decay steps 2,000: \( lr = lr_0 \times 0.96^{current \_ step/2,000} \)
- **One cycle policy**: lower bound 0.0001, upper bound 0.001, annealing to \( 10^{-5} \) in the last 10% of training steps (see Algorithm 1)
- **Warm restarts**: lower bound 0.0001, upper bound 0.001, cycle length = one epoch (see Algorithm 2)

Algorithm 1: One Cycle Policy for CapsNet

```plaintext
1: OCP(\( l_{min}, l_{max}, TotalSteps, T_{curr} \))
2: procedure OCP(\( l_{min}, l_{max}, TotalSteps, T_{curr} \))
3: \( \text{lr} \leftarrow 0.45 \times TotalSteps \)
4: \( m \leftarrow l_{max} - l_{min} \)
5: \( m_{ann} \leftarrow 9 \times \frac{l_{min}}{TotalSteps} \)
6: if \( T_{curr} \leq l_{min} \) then
7: \( \text{lr} \leftarrow mx + l_{min} \)
8: else if \( l_{min} < T_{curr} < 2t_{min} \) then
9: \( \text{lr} \leftarrow \text{lr} \cdot (x - 2t_{min}) + l_{min} \)
10: else
11: \( \text{lr} \leftarrow -m \cdot (x - 2t_{min}) + l_{min} \)
12: end if
13: end procedure
```

Fig. 5: FasTrCaps processing flow: the CapsNet at the input goes through the different stages of optimization in parallel, to search for the right learning rate policy, batch size and complexity reduction, obtaining at the output the Optimized CapsNet, based on the optimization criteria chosen by the user.

Fig. 6: Learning rate policy for our WarmAdaBatch.
multiply

Out DigitCaps
10
16
are set to zero (Fig. 7 left). Thus, the decoder receives $10 \times 16$ inputs, which are linked to the capsule that outputs the highest magnitude. The outputs of the DigitCaps layer are fed to the decoder: the highest valued vector (capsule) at the accuracy of the CapsNet. Indeed, the absence of a decoder would result in a lower accuracy worlds.

B. Batch Size
To realize adaptive batch size, the batch size is set to 1 for the first 3 epochs, and then increased for 3 times every 5 epochs. That is, the user can choose a value $P$ for the first $3$ epochs, and then increased for $3$ times every $5$ epochs. The learning rate will assume the values, where $9$ $16$.

Algorithm 3 AdaBatch for CapsNet: the Batch Size is increased during training.

```
1: procedure ADA-BATCH($P$, CurrentEpoch)
2: if CurrentEpoch \leq 3 then
3: BatchSize \leftarrow 1
4: else if 4 \leq CurrentEpoch \leq 8 then
5: BatchSize \leftarrow 2^P
6: else if 9 \leq CurrentEpoch \leq 13 then
7: BatchSize \leftarrow 2^P+1
8: else
9: BatchSize \leftarrow 2^P+2
10: end if
11: end procedure
```

C. Complexity of the CapsNet Decoder
The decoder is an essential component of the CapsNet. Indeed, the absence of a decoder would result in a lower accuracy of the CapsNet. The outputs of the DigitCaps layer are fed to the decoder: the highest valued vector (capsule) at the output is left untouched, while the remaining 9 vectors are set to zero (Fig. 7 left). Thus, the decoder receives $10 \times 16$ inputs, where $9 \times 16$ are null. Therefore, we optimize the model by using a reduced-sized decoder (Fig. 7 right) with only the $1 \times 16$ inputs, which are linked to the capsule that outputs the highest probability. Overall, the original decoder has 1.4M parameters (weights and biases), while the reduced decoder provides a 5% reduction, with 1.3M parameters.

D. Complexity Reduction through Weight Sharing
The Algorithm 4 illustrates how to share the weights between the PrimaryCaps and the DigitCaps layers, by having a single tensor weight associated to all the 8-element vectors inside each 6x6 sparse. Using this method, it is possible to reduce the total number of parameters by more than 15%, from 8.2 millions to 6.7 millions. However, the accuracy drops by almost 0.3%, when comparing it to the baseline CapsNet.

Algorithm 4 Weight Sharing for CapsNet, applied only to the DigitCaps layer.

```
1: BatchSize is the dimension containing single elements
2: Input size is $[\text{BatchSize}, 32, 36, 8]$
3: procedure DIGITCAPS(input, BatchSize)
4: Weight size is $[\text{BatchSize}, 32, 1, 10, 16, 8]$
5: initialize weight
6: Bias size is $[\text{BatchSize}, 1, 10, 16, 1]$
7: initialize bias
8: We move along the dimension with 36 elements
9: S here stands for Step
10: for $S = 1, 36$ do
11: Result size is $[bs, 32, 36, 10, 16, 1]$
12: We use the same weight, instead of cycling
13: $u[S] \leftarrow \text{matrix_multiply}(weight[1], input[S])$
14: end for
15: Output size is $[\text{BatchSize}, 1, 10, 16, 1]$
16: $v \leftarrow \text{routing}(u, bias)$
17: return $v$
18: end procedure
```

E. WarmAdaBatch
Among the explored learning rate policies, the warm restarts guarantees the most promising results in terms of accuracy, while the adaptive batch size provides a good trade-off to obtain fast convergence. We propose WarmAdaBatch (see Algorithm 3), a hybrid learning rate policy to expand the space of the solutions by combining the best of the two worlds. For the first three epochs, the batch size is set to 1, then it is increased to 16 for the remaining training time. A first cycle of warm restarts policy is done during the first three epochs, and a second one during the remaining training epochs. The learning rate variation of the WarmAdaBatch is shown in Fig. 6.

F. Optimization Choices
Our framework is able to automatically optimize CapsNets and its training depending on the parameters that a user wants.
Algorithm 5: Our WarmAdaBatch for CapsNet.

1: ⊳ WAB stands for WarmAdaBatch
2: procedure WAB(\(l_{min}, l_{max}, MaxEpoch, MaxStep\))
3: \(T_{curr} \leftarrow 0\)
4: for Epoch \(= 1, MaxEpoch\) do
5: \(\triangleright\) Batch size update
6: \(\equiv\) Adabatch(\(l,\) Epoch)
7: if Epoch \(\leq 3\) then
8: \(\triangleright\) Steps in 3 epochs with batch size 1
9: \(T_i \leftarrow 3 \times 60,000\)
10: else
11: \(\triangleright\) Steps in 27 epochs with batch size 16
12: \(T_i \leftarrow 27 \times 3,750\)
13: end if
14: for Step \(= 1, MaxStep\) do
15: \(\triangleright\) Learning Rate update
16: \(T_{curr} \leftarrow WR(l_{min}, l_{max}, T_{curr}, T_i)\)
17: end for
18: end for
19: end procedure

V. Evaluation

A. Experimental Setup

We developed our framework using the PyTorch library \[24\], running on two Nvidia GTX 1080 Ti GPUs. We tested it on the MNIST \[12\] and Fashion-MNIST \[33\] datasets. Both datasets are composed of 60,000 samples for training and 10,000 test samples each. The MNIST is a collection of handwritten digits, while the Fashion-MNIST is a collection of grayscale fashion products. After each training epoch, a test is performed. At the beginning of each epoch, the samples for training are randomly shuffled, while the testing samples are kept in the same order. The accuracy values are computed by averaging 5 training runs. Each training run lasts for 30 epochs, with the settings for each policy equal to the ones described in Section IV. The results are computed by averaging 5 training runs. Each testing sample is kept in the same order. The accuracy is calculated as the fraction of correctly classified samples. We tested it on the MNIST \[12\] and Fashion-MNIST \[33\] datasets, applying different proposed solutions.

TABLE I: Accuracy results obtained with CapsNet for the Fashion-MNIST dataset, applying different proposed solutions.

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Epochs to reach max accuracy</th>
<th>Parameters</th>
<th>Weight Sharing</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>MNIST</td>
<td>Fashion-MNIST</td>
<td>Fashion-MNIST</td>
</tr>
<tr>
<td>90.99%</td>
<td>99.37%</td>
<td>17</td>
<td>29</td>
</tr>
<tr>
<td>91.47%</td>
<td>99.45%</td>
<td>27</td>
<td>8</td>
</tr>
<tr>
<td>90.47%</td>
<td>99.26%</td>
<td>17</td>
<td>25</td>
</tr>
<tr>
<td>90.67%</td>
<td>99.38%</td>
<td>20</td>
<td>11</td>
</tr>
</tbody>
</table>

B. Accuracy Results for the MNIST dataset

Evaluating the Learning Rate Policies: Among the state-of-the-art learning policies that we enhanced for CapsNets, the warm restarts is the most promising one, as the maximum accuracy improved by 0.074%. CapsNet with warm restarts reaches the maximum accuracy of the baseline (with fixed learning rate) in 6 epochs rather than in 29 epochs as required by the baseline, thereby providing a training time reduction of 62.07%.

Evaluating Adaptive Batch Size: Different combinations of batch sizes in adaptive batch size algorithm have been tested, since the smaller the batch size is, the faster the initial convergence. However, large batch sizes lead to slightly higher accuracy after 30 epochs and, most importantly, to a reduced training time. In fact, a CapsNet training epoch with batch size 1 lasts for 7 minutes, while with batch size 128 it lasts for only 28 seconds. Batch size 16 is a good trade off between fast convergence and short training time (i.e., 49 sec/epoch). The best results, applying adaptive batch size, are obtained using batch size 1 for the first three epochs, and then increasing it to 16 for the remaining part of the training. With this parameter selection, there is a 0.04% accuracy gain with respect to the baseline, and the maximum accuracy of the baseline is reached in 5 epochs rather than in 29 epochs as required by the baseline. However, the first three epochs take a longer time (88% longer time) because of the reduced batch size, so adaptive batch size alone is not convenient. However, the total training time is reduced by 30% as compared to the baseline.

Evaluating Weight Sharing: Among the state-of-the-art learning policies, we enhanced for CapsNets, the warm restarts is the most promising one, as the maximum accuracy improved by 0.074%. CapsNet with warm restarts reaches the maximum accuracy of the baseline (with fixed learning rate) in 6 epochs rather than in 29 epochs as required by the baseline, thereby providing a training time reduction of 62.07%.

Evaluating Adaptive Batch Size: Different combinations of batch sizes in adaptive batch size algorithm have been tested, since the smaller the batch size is, the faster the initial convergence. However, large batch sizes lead to slightly higher accuracy after 30 epochs and, most importantly, to a reduced training time. In fact, a CapsNet training epoch with batch size 1 lasts for 7 minutes, while with batch size 128 it lasts for only 28 seconds. Batch size 16 is a good trade off between fast convergence and short training time (i.e., 49 sec/epoch). The best results, applying adaptive batch size, are obtained using batch size 1 for the first three epochs, and then increasing it to 16 for the remaining part of the training. With this parameter selection, there is a 0.04% accuracy gain with respect to the baseline, and the maximum accuracy of the baseline is reached in 5 epochs rather than in 29 epochs as required by the baseline. However, the first three epochs take a longer time (88% longer time) because of the reduced batch size, so adaptive batch size alone is not convenient. However, the total training time is reduced by 30% as compared to the baseline.

Evaluating Weight Sharing: By applying weight sharing to the DigitCaps layer, we are able to achieve a 15% reduction in the number of total parameters, decreasing from 8.2 millions to 6.7 millions. However, this reduction also leads to a slight decrease in the maximum accuracy, i.e., by 0.26%.

C. Comparison of Different Optimization Types

On the CapsNet model with the MNIST dataset, we also compare the different types of optimizations in terms of accuracy, and based on the training time to reach the maximum accuracy and the number of parameters. As we can see in Fig. 8a, we compare different optimization methods in a 3-dimensional space. This representation provides the Pareto-optimal solutions, depending on the optimization goals. We also compare, in Fig. 8b, the accuracy and the learning rate evolution in different epochs, for AdaBatch, WarmRestarts and WarmAdaBatch. Among the space of the potential solutions, we discuss the following two Pareto-optimal choices in detail, i.e., the WarmAdaBatch and the combination of WarmAdaBatch and weight sharing, which we call WAB+W.
the highest accuracy (99.45%) in the shortest time (3 epochs). Varying the batch size boosts the accuracy in the first epoch, and the restart policy contributes towards accelerating the training.

WAB+WS: The standalone weight sharing reduces the number of parameters by 15%. By a combination of it with WarmAdaBatch, the accuracy loss is compensated (99.38% vs. 99.37% of the baseline), while the training time is shorter than the baseline (18 epochs vs. 29 epochs) but longer than the simple WarmAdaBatch. Our framework chooses this solution if the reduction in the number of parameters is also included in the optimization goal.

D. Accuracy Results for Fashion-MNIST

The results for the Fashion-MNIST dataset are shown in Table II and Fig. 8c,d. However, while the WAB+WS policy is the most effective policy for reducing the network parameters while keeping a relatively high accuracy, not only the WAB policy but also the ExpDecay policy and the One-Cycle-Policy show good accuracy and training time results. The WAB policy is able to keep the same training time as for the best policies but at the cost of a slight accuracy loss. Hence, even though Fashion-MNIST and MNIST require an equivalent CapsNet architecture (i.e., without any changes), our WAB policy for Fashion-MNIST is comparable to other learning policies.

VI. CONCLUSION

In this paper, we proposed FasTrCaps, a novel framework for accelerating the CapsNet training. It integrates multiple lightweight optimizations into the training loop, to reduce the training time and/or the number of parameters, based on the requirements needed. We enhanced the different learning policies, for the first time, for accelerating the training of CapsNets. Afterwards, we discussed how an integrated training framework can be developed to find the Pareto-optimal solutions, including different new optimizations for fast training like WarmAdaBatch, complexity reduction for the CapsNet decoder, and weight sharing for CapsNets. These solutions not only provide significant reduction in the training time while preserving or even improving the accuracy, but also enable a new mechanism to provide trade-off between training time, network complexity, and classification accuracy. This enables new design points under different user-provided constraints. The source-code of our FasTrCaps framework can be found at https://github.com/Alexei95/FasTrCaps.

ACKNOWLEDGMENTS

This work has been partially supported by the Doctoral College Resilient Embedded Systems which is run jointly by TU Wien’s Faculty of Informatics and FH-Technikum Wien.

REFERENCES


