FasTrCaps: An Integrated Framework for Fast yet Accurate Training of Capsule Networks

Alberto Marchisio[1,*], Beatrice Bussolino[2,*], Alessio Colucci[1,*], Muhammad Abdullah Hanif [1],
Maurizio Martina[2], Guido Masera[2],Muhammad Shafique[1]
[1]*Technische Universität Wien, Vienna, Austria*
[2]*Politecnico di Torino, Turin, Italy*
Email: {alberto.marchisio,alessio.colucci,muhammad.hanif,muhammad.shafique}@tuwien.ac.at
{beatrice.bussolino,maurizio.martina,guido.masera}@polito.it

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[1].

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][26]. 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

*These authors contributed equally to this work.

[1] https://github.com/Alexei95/FasTrCaps

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

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*.

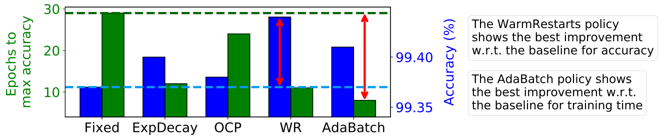


Fig. 1: Optimization potentials when considering CapsNet.

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[2].

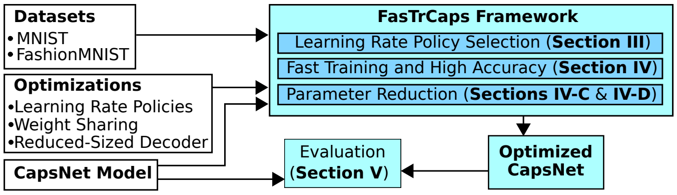


Fig. 2: An overview of our novel contributions in this paper.

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.

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 local minima or diverge, while a low learning rate can lead to a very slow convergence [2][3][4]. Given the difficulty of the choice of the best value for a constant learning rate, dynamic learning rate policies are often adopted, consisting in varying the learning rate during the training [32].

[2] A previous version of this work is available in [15].

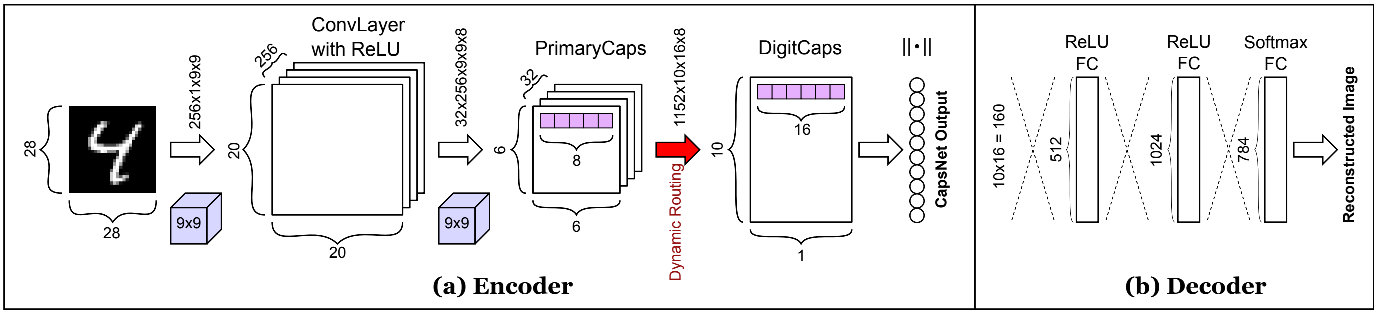
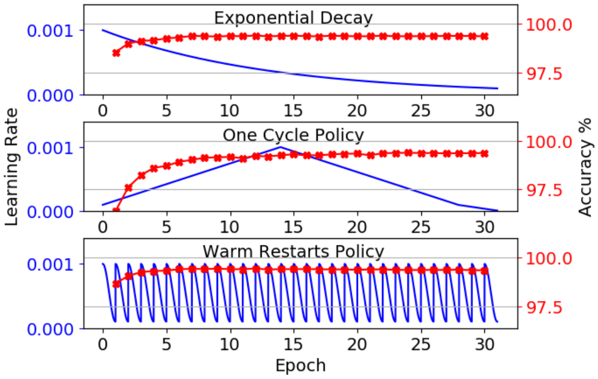


Fig. 3: Architectural view of a CapsNet illustrating both the Encoder and Decoder parts.



(a)

	LeNet5			CapsNet		
	Max. Accuracy	Epoch of max Accuracy	Epochs to reach accuracy of fixed LR	Max. Accuracy	Epoch of max Accuracy	Epochs to reach accuracy of fixed LR
Fixed	98.86	17	17	99.37	29	29
ExpDecay	99.24	28	6	99.40	12	7
OCP	99.22	30	19	99.38	24	23
WR	99.23	20	4	99.44	11	6
AdaBatch	99.18	19	5	99.41	8	5

(b)

Fig. 4: (a) In blue, how learning rate changes when different learning rate policies like exponential decay, one cycle policy and *warm restarts* are applied. In red, the accuracy reached by CapsNet at every epoch of training with the corresponding learning rate policy applied. **(b)** A table summarizing the comparative differences between LeNet5 and CapsNet when the same learning rate policies are applied. For each architecture, different columns of the table show, from left to right, the maximum reached accuracy, the training epochs[2] to reach the maximum accuracy, and the training epochs to reach the same accuracy of the network when the fixed learning rate policy is used.

One-Cycle Policy [28][29][30]: This method consists of three phases of training. In phase-1, the learning rate is linearly increased from a minimum to a maximum value in an optimal range. In phase-2, the learning rate is symmetrically decreased. In phase-3, the learning rate must be annealed to a very low value in a small fraction of the last steps. Equation 1 reports the formulas of the three phases of the *one-cycle policy*, where ts is the training step, TS is the total number of steps in the training epochs, lr_{min} and lr_{max} are the learning rate range boundaries. Saddle points in the loss function slow down the training process, since the gradients in these regions have smaller values. Increasing the learning rate helps to faster traverse the saddle points.

$$\begin{cases} lr = lr_{min} + ts \cdot \frac{lr_{max} - lr_{min}}{0.45 \cdot TS} & 0 < ts < 0.45 \cdot TS \quad \text{phase-1} \\ lr = lr_{min} + (ts - 0.9 \cdot TS) \cdot \frac{lr_{min} - lr_{max}}{0.45 \cdot TS} & 0.45TS < ts < 0.9TS \quad \text{phase-2} \\ lr = lr_{min} - 9 \cdot \frac{lr_{min}}{TS} \cdot (ts - 0.9 \cdot TS) & 0.9 \cdot TS < ts < TS \quad \text{phase-3} \end{cases} \tag{1}$$

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_{min} and lr_{max} are the learning rate range boundaries, ts is the training step, T_i is the number of training steps for each cycle. When $ts = Ti$, 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.

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

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 [1][5] or following an adaptive criterion, with the so-called *Adaptive Batch Size* [6][7][8].

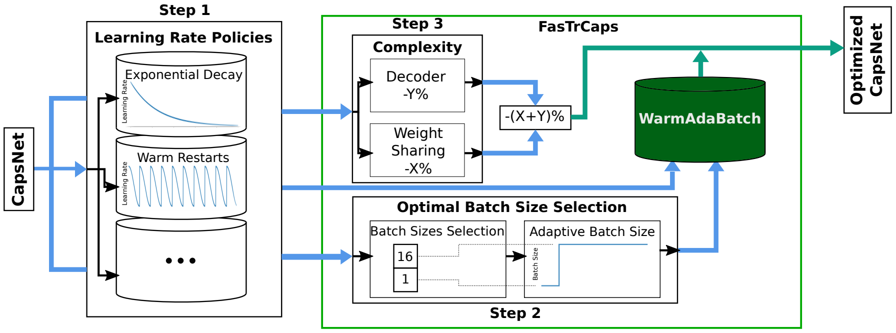


Fig. 5: FasTrCaps processing flow: the CapsNet at the input goes through the different stages of optimization in parallel, to search for the right learning rate policy, batch size and complexity reduction, obtaining at the output the Optimized CapsNet, based on the optimization criteria chosen by the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

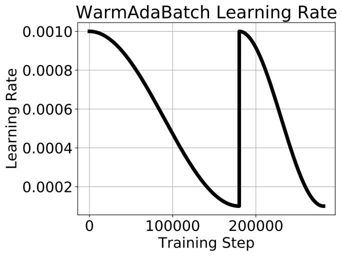


Fig. 6: Learning rate policy for our *WarmAdaBatch*.

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 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: $lr = lr_0 \cdot 0.96^{current\_step/2{,}000}$
- **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

```
▷ OCP stands for One-Cycle Policy
procedure OCP($lr_{min}$, $lr_{max}$, $TotalSteps$, $Tcurr$)
    $t_m \leftarrow 0.45 \cdot TotalSteps$
    $m \leftarrow \frac{lr_{max} - lr_{min}}{t_m}$
    $m_{ann} \leftarrow 9 \cdot \frac{lr_{min}}{TotalSteps}$
    if $Tcurr \leq t_m$ then
        $lr \leftarrow mx + lr_{min}$
    else if $t_m \leq Tcurr \leq 2t_m$ then
        $lr \leftarrow -m \cdot (x - 2t_m) + lr_{min}$
    else
        $lr \leftarrow -m_{ann} \cdot (x - 2t_m) + lr_{min}$
    end if
end procedure
```

[2] Training times for a single epoch are different between LeNet5 and CapsNet: the former takes an average of 17 seconds using the fixed learning rate, while the latter takes 49 seconds.

Algorithm 2 Warm Restarts for CapsNet: the learning rate is decayed with cosine annealing.

```
▷ WR stands for WarmRestarts
procedure WR($lr_{min}, lr_{max}, T_{curr}, T_i$)
    ▷ Learning rate update
    $lr \leftarrow lr_{min} + \frac{1}{2}(lr_{max} - lr_{min})\left(1 + \cos\pi\frac{T_{curr}}{T_i}\right)$
    if $T_{curr} = T_i$ then
        ▷ Warm Restart after $T_i$ training steps
        $T_{curr} \leftarrow 0$
    else
        ▷ Current step update
        $T_{curr} \leftarrow T_{curr} + 1$
    end if
    return $T_{curr}$
end procedure
```

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 and the batch size will assume the values 2^P, 2^{P+1} and 2^{P+2} (see Algorithm 3).

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

```
procedure AdaBatch($P, CurrentEpoch$)
    if $CurrentEpoch \leq 3$ then
        $BatchSize \leftarrow 1$
    else if $4 \leq CurrentEpoch \leq 8$ then
        $BatchSize \leftarrow 2^P$
    else if $9 \leq CurrentEpoch \leq 13$ then
        $BatchSize \leftarrow 2^{P+1}$
    else
        $BatchSize \leftarrow 2^{P+2}$
    end if
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 10×16 values, where 9×16 are null. Therefore, we optimize the model by using a reduced-sized decoder (Fig. 7 right) with only the 1×16 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

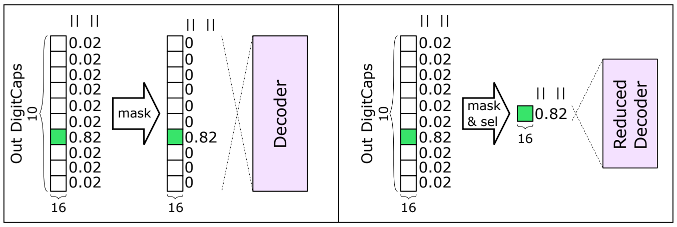


Fig. 7: (left) All the DigitCaps outputs except the one with the highest magnitude are set to zero. Then the decoder receives 10x16 inputs. **(right)** Only the DigitCaps output with the highest magnitude is fed to a reduced decoder, with 1x16 inputs.

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 capsule. 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.

```
▷ BatchSize is the dimension containing single elements
▷ Input size is [BatchSize, 32, 36, 8]
procedure DigitCaps($input, BatchSize$)
    ▷ Weight size is [BatchSize, 32, 1, 10, 16, 8]
    $initialize\ weight$
    ▷ Bias size is [BatchSize, 1, 10, 16, 1]
    $initialize\ bias$
    ▷ We move along the dimension with 36 elements
    ▷ S here stands for Step
    for $S = 1, 36$ do
        ▷ Result size is [bs, 32, 36, 10, 16, 1]
        ▷ We use the same weight, instead of cycling
        $u[S] \leftarrow matrix\_multiply(weight[1], input[S])$
    end for
    ▷ Output size is [BatchSize, 1, 10, 16, 1]
    $v \leftarrow routing(u, bias)$
    return $v$
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 5), 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.

```
▷ WAB stands for WarmAdaBatch
procedure WAB($lr_{min}$, $lr_{max}$, $MaxEpoch$, $MaxStep$)
    $T_{curr} \leftarrow 0$
    for $Epoch = 1, MaxEpoch$ do
        ▷ Batch size update
        $Adabatch(4, Epoch)$
        if $Epoch \leq 3$ then
            ▷ Steps in 3 epochs with batch size 1
            $T_i \leftarrow 3 * 60,000$
        else
            ▷ Steps in 27 epochs with batch size 16
            $T_i \leftarrow 27 * 3,750$
        end if
        for $Step = 1, MaxStep$ do
            ▷ Learning Rate update
            $T_{curr} \leftarrow WR(lr_{min}, lr_{max}, T_{curr}, T_i)$
        end for
    end for
end procedure
```

to improve. For instance, using *WarmAdaBatch*, the accuracy and the training time are automatically co-optimized. The number of parameters can be reduced, at the cost of some accuracy loss and training time increase, by enabling the *weight sharing*, along with the *WarmAdaBatch*.

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 shown in Table I and Fig. 8a,c.

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

Accuracy		**Epochs to reach max accuracy**		**Parameters**	**Weight Sharing**
FashionMNIST	*MNIST*	*FashionMNIST*	*MNIST*		
90.99%	99.37%	17	29	Fixed (Baseline)	No
91.47%	99.45%	27	8	WAB	No
90.47%	99.26%	17	26	Fixed (Baseline)	Yes
90.67%	99.38%	20	11	WAB	Yes

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 w.r.t. 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 WarmAdaBatch: As for the batch, the first cycle of learning rate lasts for 3 epochs and the second one for 27 epochs. Variation of batch size and learning rate cycles are synchronized. This solution allows to have a 0.088% gain in accuracy w.r.t. the baseline CapsNet implementation, and the baseline maximum accuracy is reached by CapsNet with *WarmAdaBatch* in 3 epochs against 29 epochs. After the first three epochs, the batch size changes and the learning rate is restarted. Hence, there is a drop of accuracy, which re-converges in a few steps to the highest and stable value.

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 reductions 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. 8c, 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+WS.

WarmAdaBatch: This solution provides the optimal point in terms of accuracy and training time, because it achieves

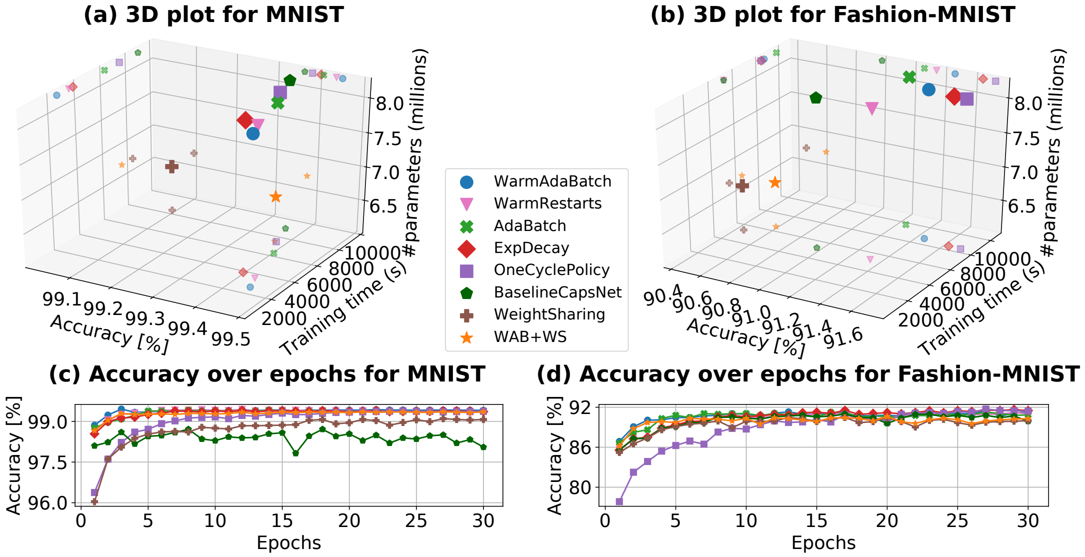


Fig. 8: *The legend is common for all the figures.* **(a,b)** Comparison of different optimization types integrated in our *FasTrCaps* framework, on the basis of accuracy, training time and number of parameters. The training time is computed as the number of epochs to reach the maximum accuracy, multiplied by time (in seconds) per epoch. The abbreviated terms WAB and WS stand for *WarmAdaBatch* and *WeightSharing*, respectively, with *WeightSharing* including also the small-decoder optimization. **(c,d)** Accuracy improvements / changes over the training epochs for different optimization solutions. (**a,c**) Results for MNIST. (**b,d**) Results for Fashion-MNIST.

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 I and Fig. 8b,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

[1] R. Babanezhad Harikandeh, M. O. Ahmed, A. Virani, M. Schmidt, J. Konečný, and S. Sallinen. Stopwasting my gradients: Practical svrg. In *Advances in Neural Information Processing Systems 28*. 2015.

[2] K. Bache, D. DeCoste, and P. Smyth. Hot swapping for online adaptation of optimization hyperparameters. 2014.

[3] Y. Bengio. *Practical Recommendations for Gradient-Based Training of Deep Architectures*, pages 437–478. Springer Berlin Heidelberg, 2012.

[4] T. Breuel. The effects of hyperparameters on sgd training of neural networks. 2015.

[5] H. Daneshmand, A. Lucchi, and T. Hofmann. Starting small - learning with adaptive sample sizes. In M.-F. Balcan and K. Q. Weinberger, editors, *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1463–1471. JMLR.org, 2016.

[6] S. De, A. K. Yadav, D. W. Jacobs, and T. Goldstein. Big batch sgd: Automated inference using adaptive batch sizes. *ArXiv*, abs/1610.05792, 2016.

[7] A. Devarakonda, M. Naumov, and M. Garland. Adabatch: Adaptive batch sizes for training deep neural networks. 2017.

[8] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour. 2017.

[9] G. Hinton, S. Sabour, and N. Frosst. Matrix capsules with em routing. 2018.

[10] G. E. Hinton, A. Krizhevsky, and S. D. Wang. Transforming auto-encoders. In *ICANN*, 2011.

[11] S. Jastrzebski, Z. Kenton, D. Arpit, N. Ballas, A. Fischer, Y. Bengio, and A. Storkey. Three factors influencing minima in sgd, 2017.

[12] Y. LeCun, C. Cortes, and C. Burges. The mnist database of handwritten digits. 1998.

[13] I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. 2016.

[14] A. Marchisio and M. Shafique. Capstore: Energy-efficient design and management of the on-chip memory for capsulenet inference accelerators. *ArXiv*, abs/1902.01151, 2019.

[15] A. Marchisio, B. Bussolino, A. Colucci, M. A. Hanif, M. Martina, G. Masera, and M. Shafique. X-traincaps: Accelerated training of capsule nets through lightweight software optimizations. *ArXiv*, abs/1905.10142, 2019.

[16] A. Marchisio, M. A. Hanif, F. Khalid, G. Plastiras, C. Kyrkou, T. Theocharides, and M. Shafique. Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 553–559, 2019.

[17] A. Marchisio, M. A. Hanif, and M. Shafique. Capsacc: An efficient hardware accelerator for capsulenets with data reuse. *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 964–967, 2019.

[18] A. Marchisio, G. Nanfa, F. Khalid, M. A. Hanif, M. Martina, and M. Shafique. Capsattacks: Robust and imperceptible adversarial attacks on capsule networks. *ArXiv*, abs/1901.09878, 2019.

[19] A. Marchisio, B. Bussolino, A. Colucci, M. Martina, G. Masera, and M. Shafique. Q-capsnets: A specialized framework for quantizing capsule networks. *Proceedings of the 57th Annual Design Automation Conference*, 2020.

[20] A. Marchisio, V. Mrazek, M. A. Hanif, and M. Shafique. Red-cane: A systematic methodology for resilience analysis and design of capsule networks under approximations. *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020.

[21] D. Masters and C. Luschi. Revisiting small batch training for deep neural networks. 2018.

[22] F. Michels, T. Uelwer, E. Upschulte, and S. Harmeling. On the vulnerability of capsule networks to adversarial attacks. *ArXiv*, abs/1906.03612, 2019.

[23] P. Mishra and K. Sarawadekar. Polynomial learning rate policy with warm restart for deep neural network. In *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, pages 2087–2092, 2019.

[24] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[25] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3856–3866. 2017.

[26] M. Shafique, T. Theocharides, C. Bouganis, M. A. Hanif, F. Khalid, R. Hafiz, and S. Rehman. An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the iot era. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 827–832, 2018.

[27] M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutlu, L. Orosa, and J. Choi. Robust machine learning systems: Challenges,current trends, perspectives, and the road ahead. *IEEE Design Test*, 37(2):30–57, 2020.

[28] L. Smith and N. Topin. Super-convergence: Very fast training of residual networks using large learning rates. 2017.

[29] L. N. Smith. No more pesky learning rate guessing games. *ArXiv*, abs/1506.01186, 2015.

[30] L. N. Smith. Cyclical learning rates for training neural networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

[31] L. N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018.

[32] Y. Wu, L. Liu, J. Bae, K.-H. Chow, A. Iyengar, C. Pu, W. Wei, L. Yu, and Q. Zhang. Demystifying learning rate polices for high accuracy training of deep neural networks. 2019.

[33] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[34] J. J. Zhang, K. Liu, F. Khalid, M. A. Hanif, S. Rehman, T. Theocharides, A. Artussi, M. Shafique, and S. Garg. Building robust machine learning systems: Current progress, research challenges, and opportunities. *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019.