# Conflicting Bundles: Adapting Architectures Towards the Improved Training of Deep Neural Networks - Supplementary Material

David Peer    Sebastian Stabinger    Antonio Rodríguez-Sánchez
University of Innsbruck
Austria
https://iis.uibk.ac.at/

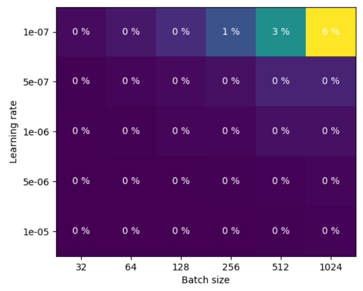
## 2.2 Conflicting bundle metric

The following approximation for definiiton 1 is motivated in the paper to check if two samples $x_i$ and $x_j$ are bundled:

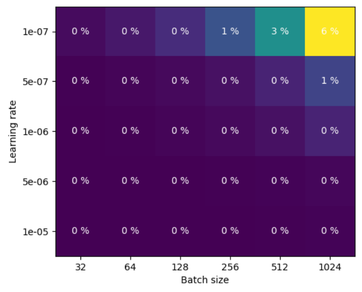
$$\frac{\alpha}{|\mathcal{B}|}\ ||a_i^{(l+1)} - a_j^{(l+1)}||_\infty \leq \gamma \tag{1}$$

To further support this claim, we demonstrate next using a heatmap how many bundles can be correctly detected if the finite representation is considered or if the finite representation is neglected. In a second experiment we compare eq. (1) that considers the resolution during backpropagation with the check $a_i^{(l+1)} = a_j^{(l+1)}$ that not considers this.

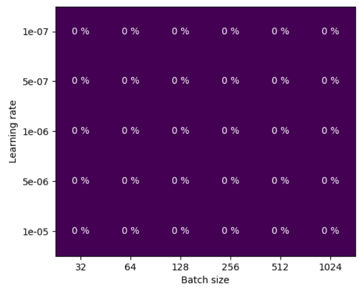
**Forward vs. backpropagation** We sampled 60k random $a^{(L)}$ values from $[0, 1)$ and randomly created weights within $[0.5, 1.0)$. For different learning rates and batch sizes, if there are two $a_i^{(L)}$ and $a_j^{(L)}$ values that update the weights $W$ exactly the same way, we report them as bundled in fig. 1a. Afterwards, we report how many bundles can be detected by simply using the $a_i^{(l+1)} = a_j^{(l+1)}$ check in fig. 1c. We report the same for the proposed method eq. (1) in fig. 1b.



(a) (Generated) real conflicts



(b) Bundles measured with eq. (1) to consider the resolution during backpropagation

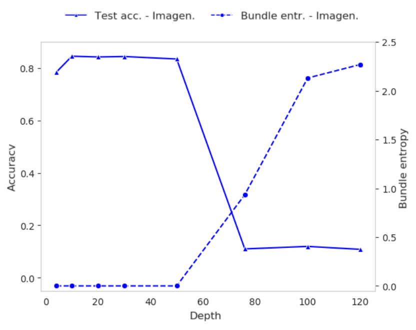


(c) Bundles measured with $a_i^{(L)} = a_j^{(L)}$ which not consideres the resolution during backpropagation

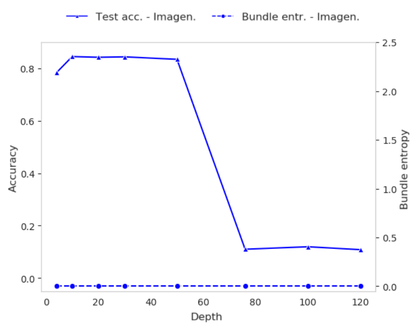
Figure 1: A heatmap that shows how many bundles are detected with different metrics.

It can be seen in fig. 1b that the floating-point representation must be considered to be able to measure the real formation of bundles during backpropagation, otherwise bundles are not detected as shown in fig. 1c.

**Measure the bundle entropy** In this experiment, we compare the bundle entropy that is measured using $a_i^{(L)} = a_j^{(L)}$ with the bundle entropy measured using eq. (1). The result is shown in fig. 2.



(a) Bundle entropy measured with eq. (1).



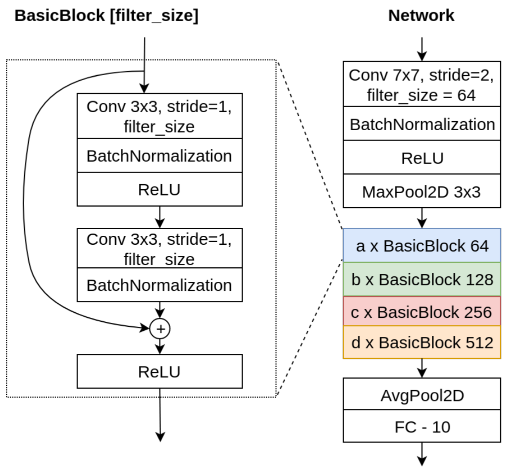
(b) Bundle entropy measured with $a_i^{(L)} = a_j^{(L)}$

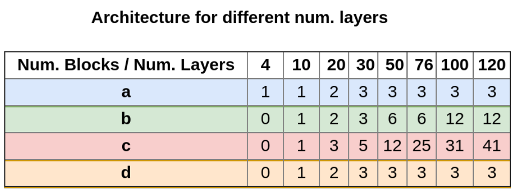
Figure 2: Comparison of the bundle entropy if the finite representation of the used floating points value is considered or not.

It can be seen that bundles are only detected if the real representation that is used in CPUs and GPUs is considered also for this real-world example. Otherwise, bundling happens during backpropagation, but it cannot be quantified correctly.

## 3.1 Setup

In image fig. 3 the architecture that is used for VGG net (without residual connection) and ResNet is shown. We followed He et al. [1] to design the basic blocks and scale the number of layers.





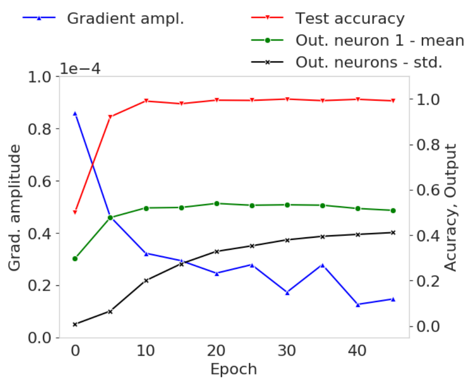
**Architecture for different num. layers**

| Num. Blocks / Num. Layers | 4 | 10 | 20 | 30 | 50 | 76 | 100 | 120 |
|---|---|---|---|---|---|---|---|---|
| a | 1 | 1 | 2 | 3 | 3 | 3 | 3 | 3 |
| b | 0 | 1 | 2 | 3 | 6 | 6 | 12 | 12 |
| c | 0 | 1 | 3 | 5 | 12 | 25 | 31 | 41 |
| d | 0 | 1 | 2 | 3 | 3 | 3 | 3 | 3 |

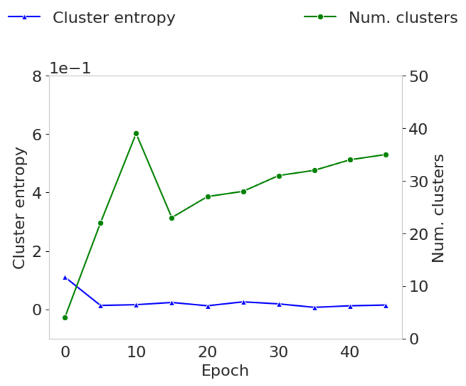
Figure 3: Architecture motivated by He et al. [1] that is used for experimental evaluation.

## 3.2 Training with a fully conflicting bundle

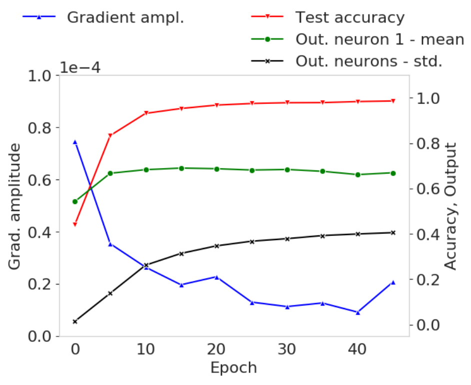
**Detailed results.** In the following graphs we show detailed training results of each experiment that we executed for the fully conflicting bundle case under controlled settings:



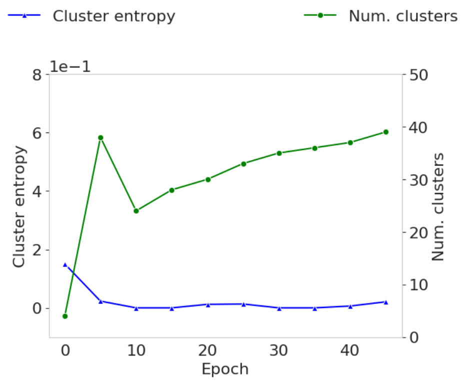
(a) No conflicting bundle, balanced dataset.



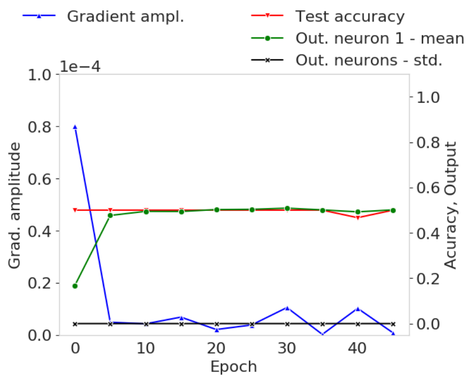
(b) No conflicting bundle, balanced dataset.



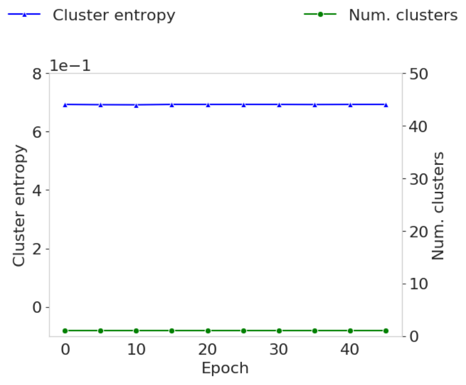
(c) No conflicting bundle, inbalanced dataset.



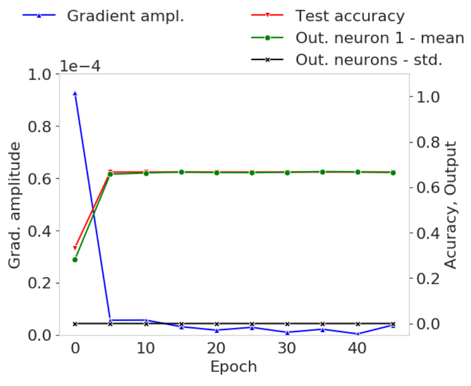
(d) No conflicting bundle, inbalanced dataset.



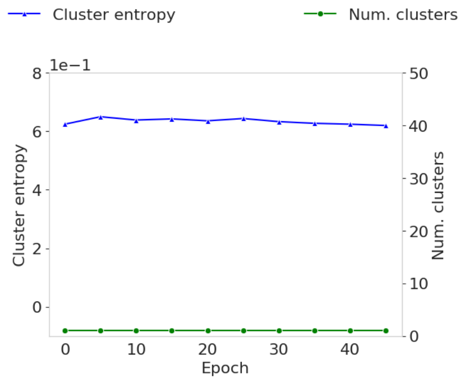
(e) Fully conflicting bundle, balanced dataset.



(f) Fully conflicting bundle, balanced dataset.



(g) Fully conflicting bundle, balanced dataset.



(h) Fully conflicting bundle, balanced dataset.

Figure 4: Experiment with a toy dataset and manually initialized weights to compare training with a fully conflicting bundle for balanced and imbalanced datasets.

**Weight initialization.** For this experiment, it is important to ensure that a single fully conflicting bundle is created in order to be able to evaluate if the predictions of our hypothesis are correct. The weights are initialized as follows:

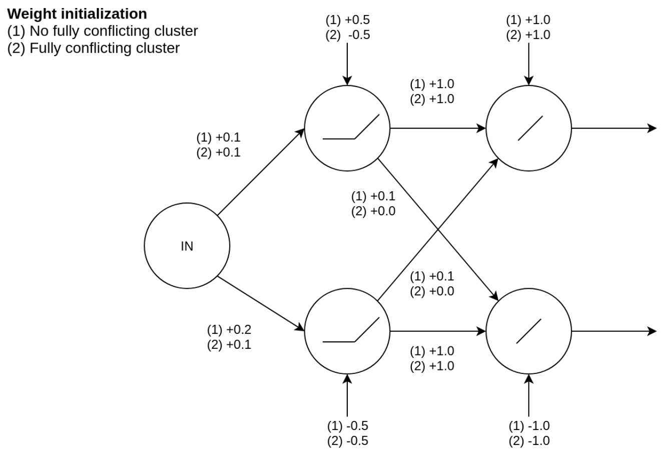


Figure 5: Initialization of the weights to not produce conflicting bundles (1) or to produce a fully conflicting bundle (2)

## 3.5 Residual connections

**Detailed proof that residuals bypass conflicts.** We prove that residual connections bypass or solve conflicting layers. A graphical idea of the proof is given in fig. 6.

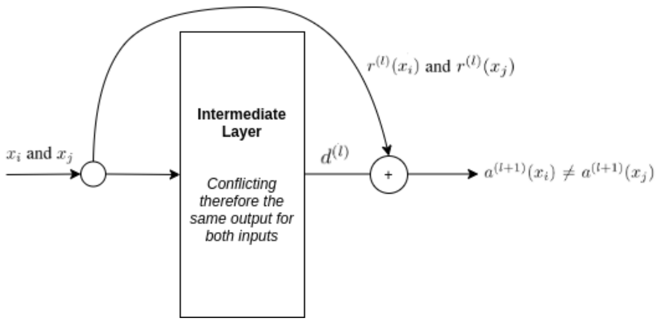

$x_i$ and $x_j$
Intermediate Layer
Conflicting therefore the same output for both inputs
$d^{(l)}$
$r^{(l)}(x_i)$ and $r^{(l)}(x_j)$
+
$a^{(l+1)}(x_i) \neq a^{(l+1)}(x_j)$

Figure 6: Graphical illustration of the proof that bijective residual functions bypass conflicting layers.

Additionally, we relax the assumption that the bypass is an identity mapping as this strong assumption is not needed for this proof, we simply assume that the bypass or residual mapping $r(x)^l$ for layer $l$ is some bijective function:

*Proof.* Assume that layer $l$ produces conflicts for $x_i$ and $x_j$ without a residual connection. We call this intermediate conflicting output $d^{(l)}$. The output for $x_i$ and $x_j$ of the layer which adds a residual connection $r^{(l)}(x)$ is therefore $a^{(l+1)}(x) = r^{(l)}(x) + d^{(l)}$. The residual $r^{(l)}(x)$ is a bijective function [1] and therefore it can be shown that the output $a^{(l+1)}(x)$ is bijective for inputs $x_i$ and $x_j$:

*Injective:* $a^{(l+1)}(x)$ is injective for inputs $x_i$ and $x_j$ iff $a^{(l+1)}(x_i) = a^{(l+1)}(x_j) \rightarrow x_i = x_j$. We know that $r^{(l)}(x_i) + d^{(l)} = r^{(l)}(x_j) + d^{(l)}$ iff $r^{(l)}(x_i) = r^{(l)}(x_j)$ iff $x_i = x_j$, because $r^{(l)}(x)$ is assumed to be bijective. Therefore $a^{(l+1)}(x)$ is injective on inputs $x_i$ and $x_j$.

*Surjective:* $a^{(l+1)}(x)$ is surjective for inputs $x_i$ and $x_j$ iff $\forall y\ \exists x.\ a(x) = y$. But for all $y$ there exists some $x$ such that $y - d^{(l)} = r^{(l)}(x)$ since $r^{(l)}(x)$ is assumed to be bijective. Therefore $a^{(l+1)}(x)$ is injective on inputs $x_i$ and $x_j$.

*Bijective:* $a^{(l+1)}(x)$ is bijective for inputs $x_i$ and $x_j$ because it is surjective and injective for those inputs. Therefore $a^{(l+1)}(x_i) \neq a^{(l+1)}(x_j)$ which conflicts the definition of bundles and it can be concluded that conflicts are solved (or bypassed) if the residual $r^{(l)}(x)$ is a bijective mapping. $\square$

We proved that *conflicts introduced by a layer are solved (or bypassed) if a bijective residual connection is added.* Note that we **not** proved that *conflicts are impossible if bijective residual connections are added*, although the experiments indicate that this happens in practice. More precisely, it is not possible to prove this stronger statement because there exists a counterexample: If the learned intermediate result is exactly the negative function of the residual, the sum of the intermediate result and the residual is always zero and therefore fully conflicting, but this case is very unlikely and therefore, we have never seen this case in the experiments (see also He et al. [1]) .

## 3.6 Auto-tune

**Evolution of the architecture.** In this table, we give one detailed execution, how the auto-tune algorithm pruned the 120 layer network architecture. The table below shows the adaption for the network trained on Cifar (Execution 1).

| Epoch | Block a | Block b | Block c | Block d |
|---|---|---|---|---|
| Start | 3 | 12 | 41 | 3 |
| 1 | 3 | 4 | 41 | 3 |
| 2 | 3 | 3 | 41 | 3 |
| **3** | **3** | **3** | **0** | **3** |

It can be seen that only three epochs are needed in order to find this architecture, although the new architecture is faster, needs less memory and maintains the accuracy as shown in the paper.

**Multiple executions.** In the tables below we report each architecture that the auto-tune algorithm created for each dataset and three different executions:

Imagenette:

| | | Block | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Execution | Layers | a | b | c | d | Accuracy | Mem. [MB] | Time / Step [ms] |
| 1 | 26 | 3 | 3 | 3 | 3 | 84.5 | 69 | 200 |
| 2 | 22 | 3 | 4 | 0 | 3 | 84.7 | 55 | 200 |
| 3 | 24 | 3 | 3 | 2 | 3 | 84.8 | 65 | 200 |

Cifar:

| | | Block | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Execution | Layers | a | b | c | d | Accuracy [%] | Mem. [MB] | Time / Step [ms] |
| 1 | 20 | 3 | 3 | 0 | 3 | 85.3 | 54 | 50 |
| 2 | 18 | 3 | 2 | 0 | 3 | 84.9 | 53 | 45 |
| 3 | 20 | 3 | 3 | 0 | 3 | 85.8 | 53 | 50 |

Svhn:

| | | Block | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Execution | Layers | a | b | c | d | Accuracy | Mem. [MB] | Time / Step [ms] |
| 1 | 20 | 3 | 3 | 0 | 3 | 95.3 | 54 | 47 |
| 2 | 18 | 3 | 2 | 0 | 3 | 95.0 | 53 | 43 |
| 3 | 20 | 3 | 3 | 0 | 3 | 95.1 | 54 | 47 |

Mnist:

| | | Block | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Execution | Layers | a | b | c | d | Accuracy | Mem. [MB] | Time / Step [ms] |
| 1 | 18 | 3 | 1 | 1 | 3 | 99.2 | 57 | 40 |
| 2 | 16 | 3 | 1 | 0 | 3 | 99.3 | 52 | 40 |
| 3 | 16 | 3 | 1 | 0 | 3 | 99.3 | 52 | 40 |

## References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.