Improving Predictive Performance and Calibration by Weight Fusion in Semantic Segmentation

Timo Sämann
Valeo Schalter und Sensoren GmbH
Germany
`timo.saemann@valeo.com`

Ahmed Mostafa Hammam
Stellantis, Opel Automobile GmbH
Germany
ahmedmostafa.hammam@external.stellantis.com

Andrei Bursuc
valeo.ai
France
andrei.bursuc@valeo.com

Christoph Stiller
Institute of Measurement and Control Systems, Karlsruhe Institute of Technology
Germany
stiller@kit.edu

Horst-Michael Groß
Ilmenau University of Technology, Neuroinformatics and Cognitive Robotics Lab
Germany
Horst-Michael.Gross@tu-ilmenau.de

Abstract

Averaging predictions of a deep ensemble of networks is a popular and effective method to improve predictive performance and calibration in various benchmarks and Kaggle competitions. However, the runtime and training cost of deep ensembles grow linearly with the size of the ensemble, making them unsuitable for many applications. Averaging ensemble weights instead of predictions circumvents this disadvantage during inference and is typically applied to intermediate checkpoints of a model to reduce training cost. Albeit effective, only few works have improved the understanding and the performance of weight averaging. Here, we revisit this approach and show that a simple weight fusion (WF) strategy can lead to a significantly improved predictive performance and calibration. We describe what prerequisites the weights must meet in terms of weight space, functional space and loss. Furthermore, we present a new test method (called oracle test) to measure the functional space between weights. We demonstrate the versatility of our WF strategy across state of the art segmentation CNNs and Transformers as well as real world datasets such as BDD100K and Cityscapes. We compare WF with similar approaches and show our superiority for in- and out-of-distribution data in terms of predictive performance and calibration.

1 Introduction

Real-world applications for which AI systems are in the safety-critical path, i.e., represent the basis for the decision-making process, impose the highest demands on the AI modules composing these systems. This refers to generalization capability, robustness, calibration, and for practical reasons, computational efficiency. Deep ensembles can demonstrably contribute to an improvement in the mentioned points (27; 16) except for efficiency. For a variety of applications, e.g., autonomous driving and driving assistance systems, the efficiency consideration is not insignificant. Despite a steadily increasing performance of information processing capabilities by both hardware and software, the required computing power and runtime is a tremendous challenge for a large number of applications. Fusion of weights instead of predictions or logits (20; 12) ultimately leads to a single set of weight only, thus effectively addressing the limited computational budget requirement. The runtime of fused weights is exactly the same as for a single network, with the additional benefit of encapsulating diverse information from multiple networks.

Fort et al. (10) show that although the same Deep Neural Network (DNN) architecture was trained with the same data and hyperparameters (only the initialization was different), the weak spots of the resulting weights were different. The basic idea of averaging weights is to compensate the weak spots of the individual weights. Averaging weights have been shot to lead to wider optima and better generalization (18). So far, few works have proposed a better understanding and improvements of this strategy. In addition, weight averaging has been mostly applied to classification tasks and the breakthrough to real world tasks such as semantic segmentation has been largely missing. One reason for the missing breakthrough may be that too little is known about the prerequisites of the weights to be fused and how these prerequisites could be fulfilled.

Here we focus on the real world task of semantic segmentation and shed light on the prerequisites of the weights to be fused. These preconditions include measurements in the weights space and function space between the weights. We measure the weight space using the cosine similarity which should be as high as possible for the weight fusion. At the same time, the weights in function space should be as far away as possible, since the distance correlates with the diversity of the weak spots. The higher the diversity, the higher the potential for weight fusion. For the measurement of the function space, we introduce the so-called oracle testing, which is novel to the best of the authors' knowledge. We give insights and guidance to find the optimal compromise between weight space and function space and describe two approaches to generating the weights to be fused. We apply weight fusion to three SOTA architectures: DeepLabV3+ (4), HRNet (36), and Segmenter (33). We use the BDD100K (40), Cityscapes (6) and ACDC (32) dataset for our experiments.

Our main contribution is to demonstrate a simple and effective method to fuse two or more sets of DNN weights into a single one. **(i)** Our weight fusion method improves predictive performance and calibration without impacting runtime cost. **(ii)** In extensive studies, we derive important properties of DNN weights towards weight fusion. **(iii)** We introduce a new testing method that can measure the functional space between weights, called oracle testing. **(iv)** We conduct extensive experiments on multiple SoTA segmentation architectures (CNNs and Transformers) on multiple real world datasets. **(v)** We show the superiority of our approach in a comparison with Stochastic Weight Averaging (SWA) (18) and deep ensembles for in-distribution as well as for out-of-distribution data (ACDC) (32).

2 Weight Fusion

Our approach of weight fusion is to generate at least two weights of the same DNN and to fuse them with a weighted averaging into a single network. The goal of the weight fusion is to improve the performance as well as the calibration of the DNN, while not impacting the runtime performance of the DNN.

Formally, we define a training dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ with n samples and labels, where $\mathbf{x}_i \in \mathbb{R}^{h \times w \times c}$ are images with spatial resolution $h \times w$ and c color channels, and $\mathbf{y}_i \in \mathbb{R}^{h \times w \times 1}$ are semantic segmentation labels (one label for each pixel in $\mathbf{x}_i$). We consider a neural network $f_\theta(\cdot)$ with trainable parameters θ, that processes the input image $\mathbf{x}_i$ and outputs a semantic map prediction $\hat{\mathbf{y}}_i$. Different random initializations of the parameters θ lead to different sets of weights after training.

We denote with $\theta^{(\tau)}$ the parameters of a specific training process τ and with $f_{\theta^{(\tau)}}(\cdot)$ the corresponding network.

The weight fusion represents a weighted averaging of the weight files (checkpoints) computed from two different training runs. Formally we define the element-wise fusion of two sets of parameters as follows: $\bar{\theta} = \alpha\theta^{(1)} + \beta\theta^{(2)}$, where $\theta^{(1)}$ and $\theta^{(2)}$ are two sets of parameters for f, α and β are fusion parameters with $\beta = 1 - \alpha$, and $\bar{\theta}$ is the set of fused weights.

2.1 Weight Properties for Improved Fusion

We hypothesize that weights $\theta^{(\tau)}$ must satisfy three properties to achieve optimal performance and calibration improvements: (i) **high cosine similarity** to each other; (ii) a **high value in oracle testing**; (iii) a **low loss on the validation data**. In the following we detail these properties.

(i) Cosine similarity. Cosine similarity is a measure of the similarity of two vectors. The cosine of the angle between two vectors $\theta^{(1)}$ and $\theta^{(2)}$ (we flatten and concat the DNN parameters into a single vector) is determined by: $cos() = \frac{\theta^{(1)} \cdot \theta^{(2)}}{||\theta^{(1)}|| \cdot ||\theta^{(2)}||}$. An angle of 0 degrees corresponds to 1, i.e., the vectors are parallel, while an angle of 90 degrees results in 0, i.e., the vectors are perpendicular to each other. Cosine similarity can be considered as a measurement in the weight space. DNNs with different initializations can reach highly different local optima after training. Two networks can have very similar predictions but low similarity, e.g., convolutional filters from the two networks are permuted. Since the weight fusion is done element-wise it is essential for the two networks to not be far away from each other as the final weights $\bar{\theta}$ risk being useless. We add the pytorch code snippets for weight fusion and cosine similarity in Appendix C.

(ii) Oracle testing. In Oracle testing, an oracle merges all true positives of the DNN predictions into a single output/prediction based on the ground truth. Then, the corresponding metric is calculated for the merged prediction (e.g., mIoU). If the DNN predictions are very similar in their errors, the result of the metric will be similar to the single predictions. In other words, the DNNs are close together in the functional space. If the DNN predictions are diverse in terms of true positives, the oracle test yields a high value. This means that the DNNs are far away in the function space. The quality of an ensemble is a major advantage of DE (20) over other approaches (10). Higher diversity means that the models are not stuck in the same local optimum and do not have the same weak spots. Quantitatively, diversity is usually computed as dissimilarity between predictions from pairs of networks (1; 10; 30). Oracle testing takes the ground truth into account and can be considered as a measurement in the function space. The pytorch code snippet is listed below in 1.

```
1  for num in range(number_weights):
2      mask[num] = [outputs[num] == ground_truth]
3      outputs[0][mask[num]] = outputs[num][mask[num]]
```

Listing 1: Iterate over the number of weights to be used to calculate the oracle testing. *Outputs* is a list containing the output of the models.

(iii) Validation loss. Empirically, we can show that the validation loss of the weights to be fused is a relevant factor and should be close to its local minimum.

2.2 Generation of DNN Weights

To generate the weights, we distinguish the following two options, which we examine in more detail in sec. 3.

1) The weights are generated by training from "scratch" (see §3.1). Here it is necessary to keep the hyperparameters as well as the training data the same for each training run. To achieve a high oracle test result, we show in our experiments with DeeplabV3+, HRNet, and the Segmenter that changing the initialization seed is sufficient. This change does reduce cosine similarity, but only slightly. With this approach we achieve a good compromise between cosine similarity and oracle testing for the mentioned architectures.

2) The weights are derived from a finetuning where an already trained weight is used as starting weight (see §3.2). The same data is used for the finetuning as for the prior training. The finetuning is

performed with a cosine annealing learning rate schedule and can range from one to several cycles. The cosine annealing learning rate schedule is defined as: $\text{lr} = \frac{1}{2} \cdot \sum_{T=1}^{N} \text{start\_lr} \cdot \left(1 + \frac{T}{N \cdot \pi}\right)$. T is the current iteration and N is the total amount of iterations, i.e., the multiplication of the number of epochs by the number of iterations per epoch. The length of the cycles can range from one to several epochs. The height of the start-learning rate for the cycles is crucial for the optimal property of the weights for the fusion.

Further procedure. In 1) and 2) we use the presented properties (cosine similarity, oracle testing, validation loss) of the weights to determine the suitable weights for the fusion. Then, we determine the fusion parameter α using grid search in 0.05 steps on the validation data. Please note that training the α parameter would not lead to any noteworthy improvement for the following two reasons: (i) Finer resolution of the step size did not achieve any significant improvements $\leq +0.1$ mIoU). (ii) There is little reduction in computational effort when comparing the forward passes required for the grid search with the forward plus backward passes required in a training process over multiple epochs. Once several fused weights have been determined, one selects the fused weight with the best predictive performance on the validation data as the final fused weight. This procedure is transferable to new DNNs, datasets or tasks.

3 Experiments

The generation of the weights to be fused can be done in two ways. (i) By training from "scratch" and (ii) finetuning. Therefore, we divide the experiments into two subsections. §3.1 describes weight fusion based on weights obtained through multiple trainings from scratch. We show that our weight fusion approach generalizes to multiple DNN architectures as well as datasets. We demonstrate statistically significant results for three state-of-the-art architectures for semantic segmentation DeeplabV3+ (4), HRNet (36) and Segmenter (33), based on real world datasets such as BDD100K and Cityscapes. We consider the fusion properties for performing weight fusion, but not focusing on analysing them in this subsection.

In §3.2, we use a final trained weight as a starting point for finetuning. We demonstrate detailed analysis on cosine similarity, oracle testing and validation loss in the context of weight fusion. For this and subsequent experiments, we limit ourselves to using DeeplabV3+ and BDD100K due to limited compute budget.

To avoid negative influence of label errors in the ground truth data of the BDD100K dataset, we adapted it manually. Furthermore, we split the available 7k images into 5k training data, 1k validation data and 1k test data. The process for adaptation and splitting as well as a description of Cityscapes (6) and ACDC (32) is described in more detail in Appendix D. For reproducibility of the experiments, we list all the hyperparameters used for the trainings in Appendix C.

3.1 Fusion of Weights Trained from Scratch

For the generation of the weights we run several trainings from scratch. We change only the initialization by specifying different random seeds, i.e., the architecture, the hyperparameters and the training data remain exactly the same. These constraints lead to weight sets with a good compromise between oracle testing and cosine similarity, which is needed for weight fusion. To ensure a certain statistical significance, we performed four training runs each with DeepLabV3+, HRNet and Segmenter on the BDD100K and Cityscapes datasets - a total of 20 runs.[1] This serves to reduce statistical fluctuations due to different sources of variation (7) and allowing us to reliably attribute the performance improvement to weight fusion only. The resulting 4 weights per DNN architecture and dataset allow for 6 different possible combinations when fusing 2 weights each according to: $\frac{n!}{r!(n-r)!}$ with n = 4 and r = 2.

Fig. 1 illustrates the behavior of the DeepLabV3+ architecture during weight fusion in terms of predictive performance and calibration. Fig. 1(a) shows the mIoU for DeepLabV3+ on the validation data from BDD100K and Cityscapes and on the test data from BDD100K. The mIoU values are obtained by averaging the mIoU values over the 6 fused weights and are represented by a black line.

---

[1]Due to convergence difficulties with the Segmenter on the BDD100K data, we limit the evaluation of weight fusion to Cityscapes.
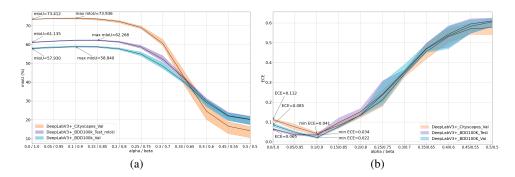
Figure 1: DeepLabV3+ performance and calibration for 6 fused weights averaged for α from 0 to 1 in 0.05 steps. Please note that $\alpha/\beta = 0/1$ corresponds to the single weights. *(a)* mIoU *(b)* ECE.

Table 1: Per class mIoU in % with the fusion parameters $\alpha/\beta = 0.1/0.9$ for BDD100K and 0.15/0.85 for Cityscapes. The number between brackets shows the improvement (green) or deterioration (red) compared to the single weights. Especially the classes with a small pixel density have improved the most. Please note that the class `train` was not learned at all probably due to the limited images containing `train` in the BDD100K training data.

	Road	Sidewalk	Building	Wall	Fence	Pole	Traffic Light	Traffic Sign	Vegetation	Terrain	Sky	Person	Rider	Car	Truck	Bus	Train	Motorcycle	Bicycle	Overall
2*BDD100K	94.71 (+0.14)	65.40 (+0.51)	85.77 (+0.25)	29.29 (+0.02)	51.14 (+0.14)	52.91 (+2.86)	58.33 (+4.67)	56.29 (+3.46)	86.49 (+0.12)	51.17 (+0.56)	95.32 (+0.17)	65.73 (+2.10)	46.36 (-1.33)	90.70 (+0.39)	58.88 (+1.73)	81.41 (+2.30)	0.00 (0.00)	55.87 (+1.42)	57.30 (+3.67)	62.27 (+1.14)
2*Cityscapes	97.73 (+0.03)	82.92 (-0.01)	91.39 (+0.25)	52.72 (+0.44)	56.77 (-0.23)	60.09 (+0.78)	67.03 (+2.93)	75.64 (+1.67)	91.56 (+0.17)	62.95 (+0.14)	94.06 (+0.60)	79.41 (+1.25)	60.62 (+1.00)	93.48 (+0.01)	64.25 (-0.35)	80.67 (+0.25)	61.55 (+0.99)	58.86 (+0.20)	74.59 (+1.36)	73.94 (+0.53)

The colored areas show the minimum and maximum mIoU values. The x-axis describes α and β, the scalar values used as fusion parameter. The mIoU values are mirrored at the point $\alpha = 0.5$, so that the order of the weights can be neglected. For this reason, the x-axis that extends to 0.5 is complete. It is illustrated that with the fusion parameters $\alpha/\beta = 0.1/0.9$ and 0.15/0.85, the maximum mIoU value is obtained over the 6 averaged mIoU values (see arrows in Fig. 1(a)). Table 1 shows the classes mIoU for these fusion parameters. It can be seen that the classes with a low pixel density such as `traffic light`, `traffic sign`, `bicycle` and `person` benefit the most from weight fusion.

How this behavior looks in semantic segmentation masks can be seen in Fig. 2(a). The images with $\alpha = 0$ and $\alpha = 1$ correspond to the single predictions of weights 0 and 1, respectively. As alpha approaches 0.5, fewer predictions are seen for the small classes such as `traffic light`, `person`, etc., which explains the increased precision and reduced recall. Looking at the `sidewalk` in the image with $\alpha = 0.15$, we notice that it is most similar to ground truth. This shows that the fused weight is not simply an alignment of the two weights 0 and 1, but represents a completely new function. In other words, the weight fusion results in a new point in function space. More figures confirming this behavior can be found in Appendix D.

Further analysis was done on the weight fusion by evaluating the calibration of the fused DNNs based on the expected calibration error (ECE) (14; 26), see Fig. 1(b). Explanation of the ECE are in Appendix H available. It can be seen that the lowest ECE is found at $\alpha = 0.1$ where the ECE reaches the lowest value for all three data splits, which means that a significantly better calibration is achieved matching its counterpart maximum mIoU in Fig. 1(a). This indicates that the fusion does not only improve the mIoU but also improves the calibration of the network. Further evaluations regarding precision and recall as well as Kullback-Leibler (KL) divergence can be found in Appendix I.

Average results for DeepLabV3+, HRNet, and Segmenter are shown Table 2. The numbers in parentheses indicate the improvement (green) and deterioration (red) from the single weight. Please note that the numbers given are averages over 4 single weights and the resulting 6 fused weights. Furthermore, for the sake of clarity, the same α parameter was used for all 6 fused weights, i.e., there was no individual selection of α, which would have led to even better results. For the results on BDD100K, the α value that performed best on the validation data was used. Improvements in performance (mIoU, precision, recall) and calibration (ECE, KL) can be seen across the board. This demonstrates that the weight fusion is also transferable to different CNNs and also Transformers.
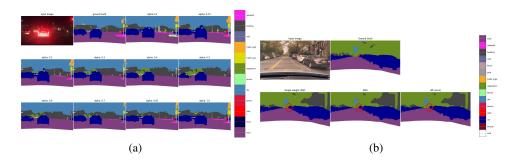
Figure 2: (a) Illustration of semantic segmentation for different alpha values using DeepLabV3+ and an example image from BDD100K. $\alpha = 0.0$ and 1.0 show the predictions of the single weights. $\alpha = 0.15$ shows the prediction of the fused weight where the pedestrian and the sidewalk are better segmented. Please note that the color white represents `void`. (b) Visual comparison of the SWA, weight fusion (WF) and single weight (SW) methods. WF is able to segment the rider correctly (see ellipse).

Table 2: Results for three architectures.

| | DeepLabV3+ | | HRNet | | Segmenter |
| | BDD100K | Cityscapes | BDD100K | Cityscapes | Cityscapes |
Metrics	alpha=0.1	alpha=0.1	alpha=0.45	alpha=0.1	alpha=0.05
Mean IoU (%) ($\uparrow$)	62.24 (+1.10)	73.94 (+0.53)	55.57(+0.79)	69.65(+0.23)	70.88 (+0.34)
Precision (%) ($\uparrow$)	75.28 (+2.92)	86.47 (+2.19)	72.05(+0.46)	83.68(+1.20)	84.14(+0.30)
Recall (%) ($\uparrow$)	72.69 (-1.34)	82.66 (-1.40)	65.31(+1.56)	79.86(+0.63)	80.24(+0.22)
ECE ($\downarrow$)	0.034 (-0.031)	0.041 (-0.071)	0.111(-0.002)	0.132(-0.007)	0.030(-0.001)
KL ($\uparrow$)	1.919 (+0.312)	1.826 (+0.297)	0.858(-0.077)	0.952(+0.07)	2.445(+0.016)

3.2 Fusion of Finetuned Weights

Besides training from scratch (see §3.1), finetuning is a way to generate the weights for the WF. Finetuning uses the same data as training from scratch. A fully trained weight is used for initialization which we consider as starting weight. In our experiments the finetuning is performed with learning rates between 0.002 and 0.02 in 0.002 steps and covers 10 epochs. The cosine annealing learning rate schedule was used so that the 10 epochs correspond to one cycle (see Fig. 3(a)). For the fusion we use the starting weight (at epoch 50) and one of the individual checkpoints of the respective epochs 1 to 10. For this experiment we use the DeepLabV3+ and the BDD100K validation data set. Fig. 4(a) and (b) show the cosine similarity and the mIoU of the oracle testing, respectively, for the starting weight and the weight of the 10th epoch. Fig. 4(c) shows the mIoU after fusion of the starting weight with the checkpoint of the 10th epoch along the fusion parameter α for the corresponding learning rates. The improvements of the mIoU compared to the starting weight, which is 58.61 %, are marked with a diamond. From Fig. 4 we derive the following findings:

Finding 1: Cosine similarity and oracle testing are correlated.

The cosine similarity measures the distance of the weights in the weight space and the oracle testing in the functional space. The closer the weights are in weight space, the closer they are in functional space and vice versa. Using Fig. 4(a) and (b) it can be seen that the lower the learning rate the closer they are together in weight space (cosine similarity approaches 1.0) and at the same time they are close together in functional space (oracle testing approaches the single weight mIoU of 58.61 %).

Finding 2: Cosine similarity must not be too low.

A low cosine similarity is equivalent to a long distance in weight space. For a successful fusion the weights must not be too far apart in weight space. Based on the mIoU of the fused weights (see Fig. 4(a) and (c)) it can be seen that a cosine similarity smaller than about 0.925 at no α value results in a higher mIoU than the starting value. This is the case for learning rates 0.02, 0.018 and 0.016.
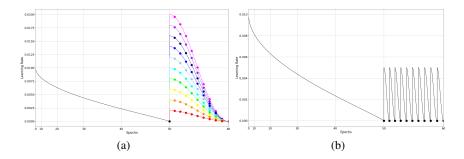
Figure 3: Representation of the learning rate schedule. The blobs indicate the positions from which a checkpoint was stored for our experiments. (a) Used in §3.2. (b) Used in §4
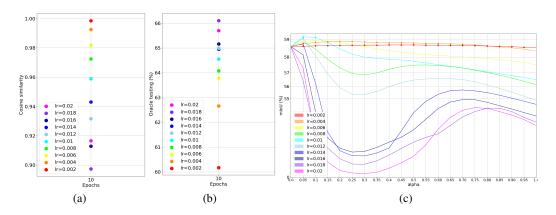

Figure 4: *(a)* Cosine similarity for the 10th epoch weight compared to the starting weight for the corresponding learning rates. *(b)* Oracle testing for the 10th epoch weight and the starting weight for the corresponding learning rates. *(c)* mIoU after fusion of the starting weight with the checkpoint after the 10th epoch along the fusion parameter alpha for the corresponding learning rates.

Finding 3: Oracle test and cosine similarity should both be as high as possible.

Based on Finding 1, we know that these are negatively correlated with each other and thus a compromise must be found. The learning rate 0.01 seems to be a good compromise, since it leads to the highest fused mIoU in Fig. 4(c).

Two further findings regarding the weighting parameter and the validation loss have been derived and are described in Appendix B. By and large the combination of a low validation loss with a high oracle testing and a high cosine similarity are criteria for a successful weight fusion. This tradeoff is not achievable for SWA because the weights are equal-weighted averaged, which in turn requires a very high cosine similarity between the weights (cf. Finding 4 in B). WF can find a better compromise by exploiting the α parameter. To apply this knowledge to new problems, we have summarized some practical advice in Appendix E.

4 Results

For the results section, we restrict ourselves to the DeepLabV3+ architecture for our WF approach as well as for the comparison methods. We train all methods on the BDD100K training dataset and evaluate them using the test dataset from BDD100K and ACDC. The ACDC dataset is used to evaluate the generalization ability of our WF in the presence of a distribution shift. The ACDC dataset contains 19 classes as BDD100K and 4 test datasets of the domains fog, night, rain and snow. First, in §4.1 we describe the methods used for the results and then in §4.2 we present mainly the qualitative results in Table 3.

7

4.1 WF and Comparison Methods

For the comparison we consider the SWA method as well as deep ensembles. For SWA we train the DeepLabV3+ for 50 epochs and then finetune with cosine annealing for 10 cycles, where one cycle corresponds to one epoch. After 50 epochs and at the end of each cycle, we save the weights for the SWA baseline. This results in a total of 11 weights. We show the scheduling of the learning rate in Fig. 3(b)[2]. The blobs in this figure indicate the weights used for the SWA baseline. We refer to the single weight after 50 epochs as SW. To construct the SWA baseline, we perform an equally weighted averaging over the 11 weights. We subsequently adapt that statistics of the BatchNorm layers by updating the running means and variances over the training data (13; 18). However, this adaptation had very little impact on performance and calibration in our evaluation.

For the comparison of our weight fusion (WF) to SWA, we choose 2 out of the 11 weights. We take the weight after 50 epochs (SW) and one of the remaining 10 weights. We test all 10 combinations and determine α by using grid search in 0.05 steps on the validation data. We list an analysis for cosine similarity, oracle testing and validation loss for this in Appendix F. For the test data, the same α parameter is used that was determined on the validation data.

We also compare against DE of varying sizes: two, three and four member ensembles denoted as DE_2, DE_3 and DE_4, respectively. We use the 4 DeepLabV3+ models from § 3.1 trained from scratch with different random initializations. The fusion is performed at softmax level, i.e., we average the softmax predictions of the members of the ensemble.

For our comparison approaches (WF_2, WF_3, WF_4) to the deep ensembles, we use the same weights. To determine the fusion parameter α for WF_2, we perform a grid search in 0.05 steps using the training dataset and fuse the two weights accordingly. For WF_3 and WF_4, we need one and two additional fusion parameters, respectively. Since the computational cost of grid search increases exponentially with each additional fusion parameter, we selected the fusion parameters by a random process. Fusion parameters were chosen randomly and the mIoU were determined on the training data. This process was performed 50 times for WF_3 and WF_4, which kept 2 NVIDIA TITAN RTX GPUs busy for 1 day. The fusion parameter that achieved the highest mIoU on the training data was used to create the final weight for WF_3 and WF_4. The tables with the corresponding numerical data for this process to better understand the selection of members for DE_2 and DE_3 can be found in Appendix G. We also use the 4 single weights used by the DE_4 and WF_4 as baselines. We report the mean of the mIoU, ECE and KL metrics for the predictions of the 4 single weights. We denote this by "Mean SW".

4.2 Qualitative and Quantitative Results

After explaining the methods used, we present the qualitative results on BDD100K and ACDC in Table 3. For SWA and WF we report the difference in brackets (green or red) with respect to SW. For DE_2 to WF_4 we show the difference in brackets (green or red) with respect to Mean SW. The change of the reference method from SW to Mean SW seems to be a logical consequence, since the 4 weight files of Mean SW are partially represented in DE_2, DE_3, WF_2 and WF_3 and completely in DE_4 and WF_4. Our WF method outperforms the single weight as well as our main competitor SWA in mIoU, ECE and KL. Looking at the test data of ACDC, the difference is significant. For example, in the domain `snow` WF boosts mIou with +4.28%, while SWA is worse by -0.34% mIoU than the single weight. The comparison of the calibration metrics ECE and KL show that WF is slightly better calibrated than SWA.

We conduct a visual comparison between the SW, SWA and WF predictions in Fig. 2(b). Our approach predicts the `rider` class (see red ellipse) more accurately. Please note that in the ground truth image the `bike` was incorrectly labeled as `motorcyle`, which is however correctly predicted by all approaches.

The evaluation of the weight fusion methods (WF_2 to WF_4) reveals a clear improvement over the baseline Mean SW except for the mIoU of the `rain` domain. Using more than two weights slightly improves predictive performance and calibration almost consistently. The comparison with DE shows a significantly better KL divergence and a partially better ECE. Only when comparing the mIoU of

---

[2]For the 10 cycles, we tested the learning rates 0.0025, 0.005, and 0.0075. With the learning rate 0.005 we have achieved the best result for SWA and therefore we will focus on this learning rate in the following.

Table 3: Results of weight fusion and comparable methods on BDD100K (in-distribution) and ACDC (out-of-distribution). Please refer to §4.1 for a full explanation of the methods.

| | BDD100K | | | ACDC | | | | | | | | | | | |
| | test data | | | fog | | | night | | | rain | | | snow | | |
Methods	mIoU (%) (↑)	ECE (↓)	KL (↑)	mIoU (%) (↑)	ECE (↓)	KL (↑)	mIoU (%)	ECE (↓)	KL (↑)	mIoU (%) (↑)	ECE (↓)	KL (↑)	mIoU (%)	ECE (↓)	KL (↑)
Single weight (SW)	61.81	0.246	0.4222	44.34	0.191	0.694	22.08	0.214	0.869	42.08	0.189	0.792	37.28	0.206	0.622
SWA	61.67 (+0.86)	0.111 (-0.135)	0.944 (+0.522)	44.57 (+0.23)	0.139 (-0.052)	0.823 (+0.129)	22.80 (+0.72)	0.169 (-0.045)	0.830 (-0.039)	42.25 (+0.17)	0.138 (-0.051)	0.864 (+0.072)	36.94 (-0.34)	0.17 (-0.036)	0.888 (+0.266)
WF (ours)	62.05 (+1.24)	0.080 (-0.166)	1.103 (+0.681)	47.38 (+3.04)	0.149 (-0.042)	1.135 (+0.441)	24.95 (+2.87)	0.159 (-0.055)	0.863 (-0.006)	44.60 (+2.52)	0.141 (-0.048)	1.051 (+0.259)	41.56 (+4.28)	0.162 (-0.044)	1.018 (+0.396)
Mean SW	61.14	0.194	0.720	45.48	0.198	0.729	23.32	0.231	0.646	41.99	0.198	0.837	38.45	0.214	0.656
DE_2	62.04 (+0.90)	0.149 (-0.045)	0.890 (+0.17)	42.29 (-3.19)	0.239 (+0.041)	0.688 (-0.041)	21.66 (-1.66)	0.269 (+0.038)	0.507 (-0.139)	38.80 (-3.19)	0.232 (+0.034)	0.868 (+0.031)	35.53 (-2.92)	0.249 (+0.035)	0.566 (-0.09)
DE_3	62.39 (+1.25)	0.144 (-0.05)	0.912 (+0.192)	46.16 (+0.68)	0.178 (-0.02)	0.689 (-0.04)	24.12 (+0.8)	0.210 (-0.021)	0.550 (-0.139)	42.68 (+0.69)	0.183 (-0.015)	0.798 (-0.039)	38.51 (+0.06)	0.202 (-0.012)	0.546 (-0.11)
DE_4	62.88 (+1.74)	0.159 (-0.035)	0.883 (+0.163)	47.35 (+1.87)	0.170 (-0.028)	0.807 (+0.078)	24.78 (+1.46)	0.192 (-0.039)	0.470 (-0.176)	43.40 (+1.41)	0.159 (-0.039)	0.736 (-0.101)	39.61 (+1.16)	0.187 (-0.027)	0.568 (-0.088)
WF_2 (ours)	62.17 (+1.03)	0.130 (-0.064)	1.106 (+0.386)	46.58 (+1.10)	0.185 (-0.013)	1.461 (+0.732)	23.78 (+0.46)	0.189 (-0.042)	1.534 (+0.888)	41.63 (-0.36)	0.189 (-0.009)	1.310 (+0.473)	38.84 (+0.39)	0.207 (-0.007)	1.299 (+0.643)
WF_3 (ours)	62.27 (+1.13)	0.131 (-0.063)	1.113 (+0.393)	46.73 (+1.25)	0.193 (-0.005)	1.647 (+0.918)	23.91 (+0.59)	0.192 (-0.022)	1.610 (+0.964)	41.47 (-0.52)	0.189 (-0.009)	1.340 (+0.503)	38.81 (+0.36)	0.210 (-0.004)	1.290 (+0.668)
WF_4 (ours)	62.32 (+1.18)	0.114 (-0.08)	1.130 (+0.41)	46.80 (+1.32)	0.164 (-0.027)	1.578 (+0.849)	24.14 (+0.82)	0.158 (-0.073)	1.514 (+0.868)	41.86 (-0.13)	0.177 (-0.021)	1.378 (+0.541)	38.76 (+0.31)	0.182 (-0.032)	1.292 (+0.67)

DE variant with 4 members (DE_4), WF_2 to WF_4 fare worse. We emphasize that the runtime of DE increases linearly with the number of members. Thus, DE_4 is 4× slower at inference than SWA and WF.

All in all, weight fusion has advantages over its main competitor SWA in terms of performance and calibration. Especially when it comes to generalization capability, WF excels. The comparison to DE shows that weight fusion leads to a better calibration and can keep up with the performance of the an ensemble of three networks, while taking only 1/3 of the runtime in this case. Additional visual results and per class IoU analysis for the BDD test data are included in Appendix J.

5 Conclusion

We have presented a strategy to fuse two or more sets of weights into a single one. The weights can be computed by training from scratch or by finetuning. We have derived from extensive experiments that the similarity in weight space between the weights must be as large as possible and at the same time the similarity in functional space must be as small as possible. To measure the functional space, we introduced a new testing method called oralce testing. When computing the weights by finetuning, the validation loss plays a decisive role, which should be as close as possible to the local minimum. Based on extensive studies with SoTA architectures (CNNs and Transformers) we presented the improvements in predictive performance and calibration. We have shown that equal-weighted fusion as performed in SWA does not usually lead to the best results and can even be detrimental if the cosine similarity is too low. We outperformed stochastic weight averaging (SWA) on the BDD100K test data as well as under presence of a distribution shift on the ACDC data in performance and calibration. Furthermore, comparison with deep ensembles shows that our weight fusion has slightly better calibration and comparable performance to the deep ensemble with 3 members, albeit the latter requires 3× the runtime.

6 Outlook

(i) Considering the beneficial effect of our weight fusion, the question arises about the potential of more complex variants, such as layer-level fusion or nonlinear weight fusion. (ii) The question arises whether with a new loss function in training the weights can be driven further apart in their functional space while keeping the distance in weight space sufficiently small, which is expected to result in further improvement in performance and calibration. (iii) Based on the results in Fig. 5 subplot 1 where the fused mIoU did not occur at the end of the cycle but during the cycle, the question arises whether SWA and its variants should better average the weights in the middle of the cycle than at the end. (iv) Furthermore, we leave the applicability of weight fusion to other autonomous driving tasks like object detection or point cloud segmentation to future work.

Acknowledgments and Disclosure of Funding

References

[1] Aksela, M.: Comparison of classifier selection methods for improving committee performance. In: International Workshop on Multiple Classifier Systems (2003)

[2] Ashukha, A., Lyzhov, A., Molchanov, D., Vetrov, D.: Pitfalls of in-domain uncertainty estimation and ensembling in deep learning (2020)

[3] Cha, J., Chun, S., Lee, K., Cho, H.C., Park, S., Lee, Y., Park, S.: Swad: Domain generalization by seeking flat minima. Advances in Neural Information Processing Systems **34** (2021)

[4] Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-decoder with atrous separable convolution for semantic image segmentation. In: Proceedings of the European conference on computer vision (ECCV). pp. 801–818 (2018)

[5] Cheng, B., Collins, M.D., Zhu, Y., Liu, T., Huang, T.S., Adam, H., Chen, L.C.: Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation (2020)

[6] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3213–3223 (2016)

[7] Dehghani, M., Tay, Y., Gritsenko, A.A., Zhao, Z., Houlsby, N., Diaz, F., Metzler, D., Vinyals, O.: The benchmark lottery. arXiv preprint arXiv:2107.07002 (2021)

[8] Dinh, L., Pascanu, R., Bengio, S., Bengio, Y.: Sharp minima can generalize for deep nets. In: International Conference on Machine Learning. pp. 1019–1028. PMLR (2017)

[9] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale (2021)

[10] Fort, S., Hu, H., Lakshminarayanan, B.: Deep ensembles: A loss landscape perspective. arXiv preprint arXiv:1912.02757 (2019)

[11] Franchi, G., Belkhir, N., Ha, M.L., Hu, Y., Bursuc, A., Blanz, V., Yao, A.: Robust semantic segmentation with superpixel-mix. arXiv preprint arXiv:2108.00968 (2021)

[12] Gal, Y., Ghahramani, Z.: Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In: ICML (2016)

[13] Garipov, T., Izmailov, P., Podoprikhin, D., Vetrov, D.P., Wilson, A.G.: Loss surfaces, mode connectivity, and fast ensembling of dnns. Advances in neural information processing systems **31** (2018)

[14] Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q.: On calibration of modern neural networks. In: International Conference on Machine Learning. pp. 1321–1330. PMLR (2017)

[15] Guo, H., Jin, J., Liu, B.: Stochastic weight averaging revisited. arXiv preprint arXiv:2201.00519 (2022)

[16] Gustafsson, F.K., Danelljan, M., Schön, T.B.: Evaluating scalable bayesian deep learning methods for robust computer vision. In: CVPRW (2020)

[17] Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J.E., Weinberger, K.Q.: Snapshot ensembles: Train 1, get m for free (2017)

[18] Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., Wilson, A.G.: Averaging weights leads to wider optima and better generalization. arXiv preprint arXiv:1803.05407 (2018)

[19] Kullback, S., Leibler, R.A.: On information and sufficiency. The annals of mathematical statistics **22**(1), 79–86 (1951)

[20] Lakshminarayanan, B., Pritzel, A., Blundell, C.: Simple and scalable predictive uncertainty estimation using deep ensembles. In: NeurIPS (2017)

[21] Lee, S., Purushwalkam, S., Cogswell, M., Crandall, D., Batra, D.: Why m heads are better than one: Training a diverse ensemble of deep networks. arXiv preprint arXiv:1511.06314 (2015)

[22] Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection (2017)

[23] Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR (2015)

[24] Mukhoti, J., Kulharia, V., Sanyal, A., Golodetz, S., Torr, P., Dokania, P.: Calibrating deep neural networks using focal loss. Advances in Neural Information Processing Systems **33**, 15288–15299 (2020)

[25] Neuhold, G., Ollmann, T., Rota Bulo, S., Kontschieder, P.: The mapillary vistas dataset for semantic understanding of street scenes. In: ICCV (2017)

[26] Neumann, L., Zisserman, A., Vedaldi, A.: Relaxed softmax: Efficient confidence auto-calibration for safe pedestrian detection (2018)

[27] Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J., Lakshminarayanan, B., Snoek, J.: Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In: NeurIPS (2019)

[28] Paszke, A., Chaurasia, A., Kim, S., Culurciello, E.: Enet: A deep neural network architecture for real-time semantic segmentation. arXiv preprint arXiv:1606.02147 (2016)

[29] Platt, J., et al.: Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. Advances in large margin classifiers **10**(3), 61–74 (1999)

[30] Rame, A., Cord, M.: Dice: Diversity in deep ensembles via conditional redundancy adversarial estimation. In: ICLR (2021)

[31] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: MICCAI (2015)

[32] Sakaridis, C., Dai, D., Van Gool, L.: Acdc: The adverse conditions dataset with correspondences for semantic driving scene understanding. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 10765–10775 (2021)

[33] Strudel, R., Garcia, R., Laptev, I., Schmid, C.: Segmenter: Transformer for semantic segmentation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 7262–7272 (2021)

[34] Tarvainen, A., Valpola, H.: Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. Advances in neural information processing systems **30** (2017)

[35] Thulasidasan, S., Chennupati, G., Bilmes, J.A., Bhattacharya, T., Michalak, S.: On mixup training: Improved calibration and predictive uncertainty for deep neural networks. Advances in Neural Information Processing Systems **32** (2019)

[36] Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., et al.: Deep high-resolution representation learning for visual recognition. IEEE transactions on pattern analysis and machine intelligence **43**(10), 3349–3364 (2020)

[37] Wen, Y., Tran, D., Ba, J.: Batchensemble: an alternative approach to efficient ensemble and lifelong learning (2020)

[38] Xie, E., Wang, W., Yu, Z., Anandkumar, A., Alvarez, J.M., Luo, P.: Segformer: Simple and efficient design for semantic segmentation with transformers. arXiv preprint arXiv:2105.15203 (2021)

[39] Yu, F., Koltun, V., Funkhouser, T.: Dilated residual networks. In: CVPR (2017)

[40] Yu, F., Xian, W., Chen, Y., Liu, F., Liao, M., Madhavan, V., Darrell, T.: Bdd100k: A diverse driving video database with scalable annotation tooling. arXiv preprint arXiv:1805.04687 **2**(5), 6 (2018)

[41] Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., Yoo, Y.: Cutmix: Regularization strategy to train strong classifiers with localizable features. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 6023–6032 (2019)

[42] Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412 (2017)

[43] Zhao, H., Qi, X., Shen, X., Shi, J., Jia, J.: Icnet for real-time semantic segmentation on high-resolution images. In: ECCV (2018)

[44] Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J.: Pyramid scene parsing network (2017)

[45] Zheng, S., Lu, J., Zhao, H., Zhu, X., Luo, Z., Wang, Y., Fu, Y., Feng, J., Xiang, T., Torr, P.H., Zhang, L.: Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In: CVPR (2021)

[46] Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., Torralba, A.: Scene parsing through ade20k dataset. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2017)

A   Related Work

Ensemble strategies.   Ensembles are a simple approach for generating robust predictions by averaging decisions from multiple networks trained with different random initialization seeds. Deep Ensemble (DE) (20) is a highly effective method for achieving both high accuracy and calibration scores (27; 16; 10). However, its computational costs for training and inferences make them prohibitive for real-time predictions. These limitations have been addressed by several approaches that generate an ensemble from a single training run by collecting intermediate checkpoints (17; 13), imbuing a network with multiple prediction heads (21) or multiple low-rank weights (37), using multiple Dropout masks during inference (12), multiple test-time augmentations of an input image (2). These approaches, while faster to train, still have slow runtime due to the multiple forward passes that are performed. In practical settings, fast predictions are key and our approach does not bring any extra compute cost at test-time.

Fusing strategies.   Deep Ensemble (20) can naturally fuse predictions from the individual networks by averaging predictions. In the quest of mimicking the robustness of ensembles and of flat minima (8) that generalize better, some approaches average network weights instead (34; 18; 3; 15). Most approaches propose simple averaging of the network weights. Here we show that simple averaging (18) is not an optimal strategy and that the way in which model weights are fused can greatly impact the final performance of a model.

Calibration.   Calibration is another key requirement for models deployed in real-world settings as it facilitates thresholding of confidence scores from network predictions. However, DNNs tend to be overconfident even when they are incorrect (14). This limitation can be addressed by a post-training operation that learns to map confidence scores to probabilities, e.g., using Platt scaling (29), temperature scaling (14). However these strategies do not generalize well under distribution shift (27) as the scaling coefficient can overfit to the in-distribution data. Alternative approaches propose strategies for improving the calibration of the DNNs during training. Data augmentation strategies based on mixing (42; 41) have been shown to improve calibration (35; 11) along with crafted loss functions (24). WF also aims for improved calibration but does not require crafted loss functions or augmentation strategies and generalizes across network architectures.

Semantic segmentation.   Semantic segmentation is a key visual perception task, in particular for autonomous driving (6; 40; 25) where it enables an understanding of the environment surrounding the vehicle. The most effective semantic segmentation architectures are currently based on fully convolutional DNNs (23) with encoder-decoder structures. In the last few years tremendous amount of architectures for semantic segmentation have been proposed aiming for fine high-resolution predictions (31; 4; 39; 36; 5), multi-scale processing (44; 22) or computational efficiency (28; 43). Furthermore, the recent progress in Vision Transformers (9) has facilitated the rise of a new generation of transformer-based decoders for semantic segmentation (33; 38; 45). We consider here prominent convolutional and transformer architectures (DeepLabV3+(4), HRNet(36), Segmenter(33)) to illustrate the generalization of our WF strategy.

B   Two Further Findings

Finding 4: If the weights to be fused have a very high similarity in weight space, the weighting parameter (alpha) plays a minor role in the fusion.

$\bar{\theta} = \alpha\theta^{(1)} + (1 - \alpha)\theta^{(2)}$, so if $\theta^{(1)} \sim \theta^{(2)}$, then $\bar{\theta} \sim \theta^{(1)} \sim \theta^{(2)}$, thus α plays a minor role. A high cosine similarity leads to a flat course of the mIoU along the α parameters (see red and orange curve in Fig. 4(c)). Equally weighted fusion is only beneficial when cosine similarity is very high.

The next finding refers to Fig. 5, which is used to analyze the requirements for the weights along the 10-epoch finetuning. For the sake of clarity, we restrict ourselves to learning rates 0.004 to 0.01. The evaluations include the mIoU of the fused weight, the mIoU of the single weight of each epoch, the validation loss, the oracle testing, and the cosine similarity. Please note that for the three subplots fused weight, oracle testing and cosine similarity, the starting weight and the weight of the respective epoch was used. The parameter α used for the weight fusion was determined for each epoch using grid search in 0.05 steps. Based on the first subplot in Fig. 5 it can be seen that the fusion of the starting weight with the weight after epoch 3, 5, 6 and 7 respectively reaches the highest mIoU value.
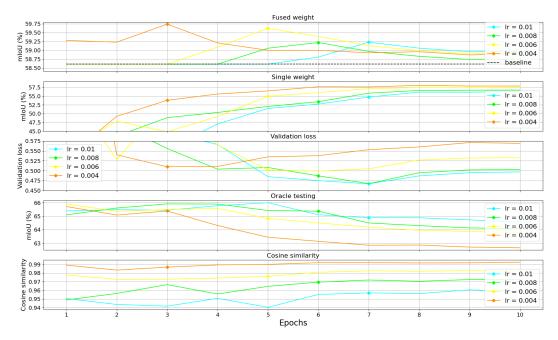
Figure 5: Results for the 10 epochs finetuning with the cyclic learning rate schedule. The first and second subplots show the mIoU for the fused and finetuned weights, respectively. The third subplot shows the validation loss for the finetuned weights. The fourth and fifth subplots show the oracle testing and cosine similarity results, respectively, using the starting weight and the finetuned weights.

For easier comparison of the subplots for these epochs, they are marked with a diamond marker. From Fig. 5 we derive the following finding:

Finding 5: A successful weight fusion is strongly correlated with the validation loss of the second weight in this finetuning setup.

In the subplot of the validation loss, we observe that the diamond markers are mostly in the minimum. This is not completely true for the learning rate 0.008, because here the minimum is one epoch further at epoch 7. However, when checking the oracle test, there is a reduction of the mIoU from epoch 6 to 7. We see this as confirmation of another claim that a high mIoU in oracle testing is important as well.

C WF and Cosine Similarity Pytorch Code Snippet

The code for WF for two weights is listetd in 2 and the code for calculating cosine similarity for two weights is listed in 3. For more details, please refer to the captions and comments of the code snippets.

```
1  for k in checkpoint[0]['state_dict'].keys():
2      checkpoint_fused['state_dict'][k] = \
3          alpha * checkpoint[0]['state_dict'][k] + \
4          beta * checkpoint[1]['state_dict'][k]
```

Listing 2: We iterate over the layers and calculate the fused weight (checkpoint_fused).

```
1   cos = nn.CosineSimilarity(dim=1, eps=1e−6)
2   # iterate over the layers and concatenate the parameters
3   for k in checkpoint[0]['state_dict'].keys():
4       # remove num_batches_tracked in bn layers
5       if len(list(checkpoint[0]['state_dict'][k].shape)) == 0:
6           continue
7       # flatten the parameters before concatenation
8       vector_key1 = checkpoint[0]['state_dict'][k].view(1, −1)
9       vector_key2 = checkpoint[1]['state_dict'][k].view(1, −1)
10      vector1 = torch.cat((vector_key1, vector1), 1)
11      vector2 = torch.cat((vector_key2, vector2), 1)
12  cos_sim = cos(vector1, vector2)
```

Listing 3: We iterate over the layers and flatten the parameters of each layer. We then concatenate the parameters before calculating the cosine similarity.

D   Datasets

In the following, we describe the datasets used and specifically address the adaptations made on the BDD100K dataset. All three datasets contain the same 19 classes and follow the Cityscapes label protocol, allowing for easy transferability of models.

BDD100K. BDD100K (40) is a large-scale and diverse dataset for autonomous driving. It contains diverse data related to weather conditions, environments and geographic. The size of the semantic segmentation part is 10k images, which is divided into 6k training, 1k validation and 3k test data with an resolution of 720x1280 pixels. Since the availability of the test dataset was not given at the time of the conducted experiments, we limited ourselves to the remaining 7k images. We split the 7k images again into 5k training, 1k validation and 1k test data. The 1k test data set corresponds to the originally designed 1k validation data set. The splitting into the 1k validation set was done randomly. Due to numerous errors in the ground truth annotations and the resulting negative impact on result interpretations, we adapted the ground truth data. Obviously there was a systematic error in the BDD100K labeling tool, so that correctly segmented objects were assigned the wrong label. In these cases we changed the objects to void, which has the effect that these objects are ignored during training and evaluation. We made such a change in about every 5th image.

Cityscapes. As BDD100K, the Cityscapes dataset (6) contains scenes from street traffic. The images were recorded in 50 cities of the countries Germany, France, Austria and Switzerland and have a resolution of 1024x2048 pixels. The 5000 image dataset is divided into 2975 training, 500 validation and 1525 test images. The label accuracy is very precise, which has made the dataset one of the most popular datasets for semantic segmentation, among others. Due to the limited accessibility of the test data set, we focus our experiments and evaluations on the validation data.

ACDC. ACDC (32) is a dataset for training and testing semantic segmentation models on road scenes with adverse visual conditions. The images were divided into the domains fog, night, rain, and snow and have a resolution of 1920x1080 pixe;s. Due to the easier accessibility, we use only the training dataset of these domains, each containing 400 images, to test the generalization ability of WF.

C   Implementation details

In this section we share information about the details on the used DNN architectures and the used hyperparameters for the trainings.

For DeepLabV3+ (4), we used the Resnet101 backbone pretrained on ImageNet and an out-stride of 16. As for the Segmenter (33), we used a Vision Transformer (ViT) (9) backbone pretrained on ADE20K (46) with patch size of 16 and token size of 768. For HRNet (36), we did not use the OCR module to upscale HRNet predictions for the sake of simplicity. Instead, we used simple bilinear interpolation. This is the reason why HRNet performs slightly worse than DeepLabV3+, even though HRNet usually performs better than DeepLabV3+, cf. Cityscapes benchmark.

We provide the common hyper-parameters used for all three DNN architectures in Table 4.

Hyper-parameter	Value
learning rate	0.01
learning rate schedule	polynomial
loss	cross-entropy
optimizer	SGD
weight decay	0.0005
momentum	0.9

Table 4: **Common hyper-parameter configuration used across experiments.**

Regarding the batch-size, this was chosen so that it could be done with one Quadro RTX 8000 that has 48 GB of GPU memory. Furthermore, we performed class weighting for cross-entropy to assign a larger weight to classes with a low pixel count. To reduce the complexity of our experiments, we did not use data augmentation in the training.

The following mean and standard deviation were used for BDD100K and Cityscapes:

Cityscapes: mean=(0.28689554, 0.32513303, 0.28389177), std=(0.18696375, 0.19017339, 0.18720214)

BDD: mean=(0.3702, 0.4145, 0.4245), std=(0.057, 0.0658, 0.0755)

For the sake of reproducablity we share the github repositories where we took the code from:

- DeepLabV3+: https://github.com/jfzhang95/pytorch-deeplab-xception

- HRNet: https://github.com/HRNet/HRNet-Semantic-Segmentation

- Segmenter: https://github.com/rstrudel/segmenter

D   Additional Qualitative Results for WF

In Figure 6 and 7 we show more visual results for WF with different fusion parameters alpha. All images are from the BDD100K test dataset and the segmentation masks were generated using DeepLabV3+. Alpha=0.0 and alpha=1.0 refers to the single weights and serves as baselines to alpha=0.15, which according to Figure 1 in the main paper led to the best results. We want to highlight in Figure 6 the class fence on the right side of the image. The segmentation of fence is different from both baselines, making it clear that the fused weight is a new function and not just a visual combination of the segmentation masks of the baselines. In Figure 7, we want to focus on the class person centered in the image. Only the fused weight with alpha=0.15 is able to segment person completely.

E   Practical Advice when Applying WF to new Problems.

The goal is to find a good compromise between cosine similarity and oracle testing. This trade-off is to achieve the highest possible value for the oracle tests while keeping the cosine similarity not too low (in most of our experiments, a value higher than 0.93 has proven to be effective). How can we exploit this knowledge to generate appropriate weights for fusion?

Broadly speaking: The higher the variance in training/finetuning the higher the value in oracle testing. At the same time, this reduces the cosine similarity. From this we conclude that the variance is the parameter for the right compromise. There are many ways to vary the variance in training/finetuning, e.g. by using a different learning rate, optimizer or initialization seed. By using two different learning rates for training from the scratch, the variance and thus the oracle testing can be increased. If one wants to reduce the variance, it is advisable to set all hyperparameters to the same value and only change the initialization seed. In our experiments it has been shown that the learning rate is a good parameter to adjust the variance during training and finetuning. Specifically, finetuning provides an easier and less computationally expensive way to generate the desired weights for fusion. Our
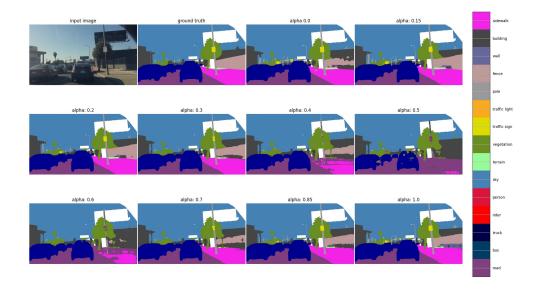
Figure 6: Visual Semantic Segmentation results for DeepLabV3+ on BDD100K data. Please note that the color white refers to void.
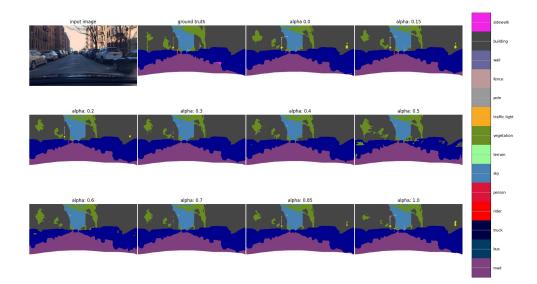

Figure 7: Visual Semantic Segmentation results for DeepLabV3+ on BDD100K data. Please note that the color white refers to void.

results in terms of predictive performance and calibration show no disadvantages to training from the scratch.

How can grid search for alpha be made more computationally efficient? After the appropriate weights have been generated, a grid search is applied to find the appropriate alpha parameter. The step size of 0.05 has proven to be useful, but this does not have to be run stupendously for all possibilities, since the changes of the mIoU are usually linear. I.e. if the alpha value of 0.05 as well as 0.1 provides an improvement, but 0.15 does not, the grid search can continue with alpha = 0.95. If no improvement can be measured at 0.95, the grid search can already be terminated. In this example, the best fused weight can be achieved with alpha 0.1.

F   Analysis of WF on SWA-Setting

Figure 8 shows the analysis of the WF on the BDD100K validation data which we use as a comparison to SWA, cf. 4.1.

The structure of the plot is the same as in Figure 5 from the main paper. The first subplot shows the mIoU after weight fusion of the starting weight with the weight of the respective epoch. The mIoU of the starting weight is called the baseline and the mIoU of SWA is also plotted for reference. The second and third subplots show the mIoU and validation loss of the weight of the corresponding epoch, respectively. Subplot 4 and 5 show the result of oracle testing and cosine similarity between the start weight and the weights of the corresponding epoch. The diamond marker indicates the highest mIoU of the fused weight, which is at epoch 6. At this point, the optimal compromise between low validation loss, high oracle testing and high cosine similarity is reached.
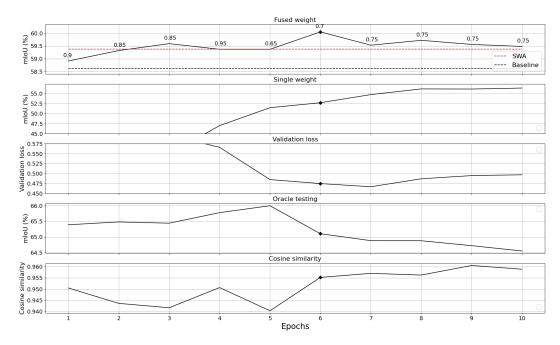

Figure 8: Analysis of the WF on the BDD100K validation data. The first and second subplots show the mIoU for the fused and finetuned weights, respectively. The third subplot shows the validation loss for the finetuned weights. The fourth and fifth subplots show the oracle testing and cosine similarity results, respectively, using the starting point and the weight of the corresponding epoch.

G   Selection of the Weights for DE_2, DE_3, DE_4, and WF_2, WF_3, WF_4

The four weight files used in DE_4 and WF_4 were taken from subsection 4.1, in which DeepLabV3+ was trained four times from the scratch on BDD100K. The selection of weights for DE_2, DE_3 and WF_2, WF_3 was determined by evaluating the mIoU on the training data. The combination of weights (checkpoint IDs 1 to 4) that achieved the highest mIoU on the training data was selected for evaluation on the test data. The bold values in tables 5 and 6 show the checkpoint IDs that reached the highest mIoU. The weights used for the deep ensemble and the weight fusion differ accordingly.

Table 5: Specification of the mIoU (%) for the selection process of the weights for DE_2 and WF_2. The checkpoint IDs (1 to 4) that resulted in the highest mIoU on the training data (marked in bold) were used for the evaluation of the test data.

Checkpoint IDs	DE_2		WF_2	
	Train data	Test data	Train data	Test data
1, 2	84,12	62.11	86.48	61.93
1, 3	84.06	62.25	86.62	62.38
1, 4	83.61	62.45	86.27	62.31
2, 3	**84.18**	**62.04**	86.64	62.18
2, 4	83.70	62.13	86.50	61.89
3, 4	83.65	62.44	**86.68**	**62.17**

Table 6: Specification of the mIoU (%) for the selection process of the weights for DE_3 and WF_3. The checkpoint IDs (1 to 4) that resulted in the highest mIoU on the training data (marked in bold) were used for the evaluation of the test data.

Checkpoint IDs	DE_3		WF_3	
	Train data	Test data	Train data	Test data
1, 2, 3	**84.29**	**62.39**	86.71	62.36
1, 2, 4	83.99	62.52	86.57	61.96
1, 3, 4	83.94	62.65	86.74	62.25
2, 3, 4	83.99	62.51	**86.74**	**62.27**

For the determination of the fusion parameter alpha for WF, a grid search in 0.05 steps was performed for each combination. Please note that there is no particular reason to select the train instead of the validation data for this selection process.

H   Expected Calibration Error (ECE)

Evaluation of DNN calibration is done using both expected calibration error (ECE) and maximum calibration error (MCE) (14; 26).

To evaluate the calibration metrics the segmentation predictions should be first sorted into fixed-sized bins β_i, $i = 1, \ldots, B$ according to their confidence. Afterwards, for each bin β_i, accuracy (acc) and confidence (conf) are computed as:

$$\text{acc}_i = \frac{\text{TP}_i}{|\beta_i|}, \qquad \text{conf}_i = \frac{1}{|\beta_i|} \sum_{j=1}^{|\beta_i|} \hat{c}_i \tag{1}$$

where $|\beta_i|$ indicates the number of examples in β_i and $\hat{c}_i$ is the respective confidence, i.e., softmax score. TP_i denotes the number of correctly classified in β_i. The ECE and MCE are then calculated accordingly:

$$ECE = \frac{1}{B} \sum_{i=1}^{B} |\text{acc}_i - \text{conf}_i|, \qquad MCE = max|\text{acc}_i - \text{conf}_i|. \tag{2}$$

I   Precision, Recall, KL

Fig. 9(a) shows the precision and recall for DeepLabV3+ on the BDD100K data. Interestingly, precision increases significantly along the x-axis, while recall decreases sharply.

Furthermore, the ability to separate between correct and incorrect classifications is an important factor in our tests. For that, we look into calculating the network's ability in being able to differentiate
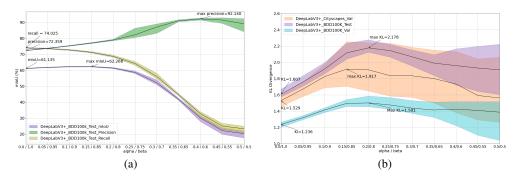
Figure 9: DeepLabV3+ performance (precision, recall) and calibration (KL) for 6 fused weights averaged for alpha from 0 to 1 in 0.05 steps. Please note that alpha/beta = 0/1 corresponds to the single weights. *(a)* precision and recall *(b)* KL.

between correct and incorrect labels (distinctive distributions) in 9(b). Accordingly, the distributions of both correct and incorrect predictions are plotted with respect to confidence score, then both distributions are compared using the Kullback-Leibler (KL) divergence metric (19) as follows:

$$D_{KL}(P \parallel Q) = - \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right),$$ (3)

where P and Q represent the correct and incorrect distributions respectively, and $\mathcal{X}$ represents the confidence score space.

High KL value indicates dissimilar distinctive distributions reflecting the efficiency of the DNN and vice versa for low KL divergence value, whilst the value 0 indicates the inability of using the confidence score in distinguishing between correct and incorrect predictions. Figure 9(b) shows that the fusion also improves the KL divergence achieving higher value than the initial weights.

J Additional Quantitative and Qualitative Results for WF and SWA Comparison

Table 7 shows the per class IoU for the comparison between the single weight, SWA and WF. On the BDD100K validation data, SWA is 0.77 % mIoU better than the single weight. WF exceeds this with an improvement of 1.44 % mIoU. On the test data, the picture is similar. Again, WF outperforms the single weight and SWA. It should be noted again that for WF the fusion parameter alpha has been determined on the validation data using grid search in 0.05 steps. It shows that the increased mIoU by WF generalizes to the test data as well. Looking at the per class IoU it is noticeable that mainly classes with a smaller pixel density such as traffic light, traffic sign, pole etc. benefit the most from the fusion. This applies to WF and SWA.

Figure 10 and 11 show further visual comparisons of SW, SWA, and WF. The areas to be pointed out are marked with a red ellipse. In figure 10 we see false positive segmentation from SW and SWA for the person class. In figure 11, on the other hand, the person class is segmented more completely by WF.
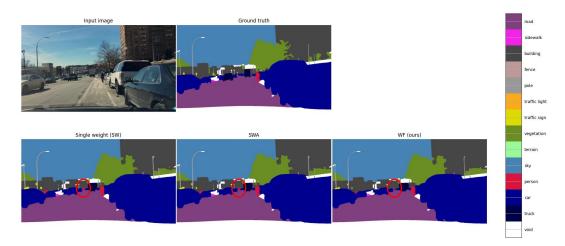
19

Figure 10: Comparison of semantic segmentation masks between SW, SWA and WF. The areas to be pointed out are marked with a red ellipse.
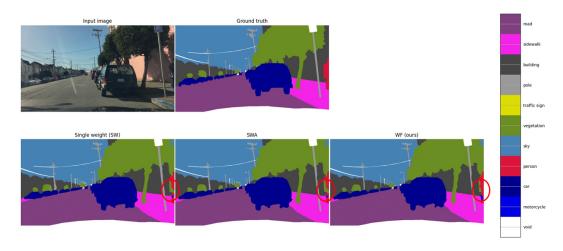

Figure 11: Comparison of semantic segmentation masks between SW, SWA and WF. The areas to be pointed out are marked with a red ellipse.

Table 7: Per class mIoU in (%) for DeepLabV3+ and the BDD100K data.

	BDD100K validation data			BDD100K test data		
Classes	Single weight	SWA	WF (ours)	Single weight	SWA	WF (ours)
Road	94.48	94.53(+0.05)	94.79(+0.26)	94.53	94.53(0.00)	94.67(+0.14)
Sidewalk	63.48	63.25(-0.23)	64.66(+1.18)	64.57	64.66(+0.09)	65.59(+1.02)
Building	83.08	83.29(+0.21)	83.40(+0.32)	85.42	85.72(+0.3)	85.69(+0.27)
Wall	37.50	42.66(+5.16)	43.33(+5.83)	26.99	31.00(+4.01)	29.13(+2.14)
Fence	46.85	46.92(+0.07)	46.58(-0.27)	50.24	50.95(+0.71)	51.21(+0.97)
Pole	47.00	50.00(+3.00)	50.25(+3.25)	49.71	52.68(+2.97)	52.91(+3.20)
Traffic Light	49.02	53.42(+4.40)	53.55(+4.53)	53.04	57.16(+4.12)	58.14(+5.10)
Traffic Sign	58.78	61.54(+2.76)	62.66(+3.88)	53.16	55.65(+2.49)	56.39(+3.23)
Vegetation	85.11	85.48(+0.37)	85.44(+0.33)	86.32	86.29(-0.03)	86.43(+0.11)
Terrain	45.76	46.94 (+1.18)	45.87(+0.11)	50.42	50.98(+0.56)	51.00(+0.58)
Sky	95.64	95.72(+0.08)	95.62(-0.02)	95.20	95.09(-0.11)	95.20(0.00)
Person	62.70	63.42(+0.72)	65.25(+2.55)	63.80	64.67(+0.87)	65.14(+1.34)
Rider	45.93	42.22(-3.71)	48.09(+2.16)	50.07	51.06(+0.99)	52.48(+2.41)
Car	88.10	88.63(+0.53)	88.73(+0.63)	90.37	90.44(+0.07)	90.79(+0.42)
Truck	59.18	62.35(+3.17)	61.56(+2.38)	57.49	59.57(+2.08)	60.55(+3.06)
Bus	73.36	74.66(+1.30)	70.47(-2.89)	78.93	78.90(-0.03)	80.66(+1.73)
Train	0	0	0	0	0	0
Motorcycle	35.09	32.14(-2.95)	36.25(+1.16)	55.46	52.70(-2.76)	54.24(-1.22)
Bicycle	42.60	41.78(-0.82)	44.52(+1.92)	49.75	48.68(-1.07)	48.78(-0.97)
Overall	58.61	59.38(+0.77)	60.05(+1.44)	60.81	61.67(+0.86)	62.05(+1.24)