Meta-Learning Initializations for Image Segmentation

Anonymous Author(s)
Affiliation
Address
email

Abstract

We evaluate first-order model agnostic meta-learning algorithms (including FOMAML and Reptile) on few-shot image segmentation, present a novel neural network architecture built for fast learning which we call EfficientLab, and leverage a formal definition of the test error of meta-learning algorithms to decrease error on out of distribution tasks. We show state of the art results on the FSS-1000 dataset by meta-training EfficientLab with FOMAML and using Bayesian optimization to infer the optimal test-time adaptation routine hyperparameters. We also construct a benchmark dataset, binary PASCAL, for the empirical study of how image segmentation meta-learning systems improve as a function of the number of labeled examples. On the binary PASCAL dataset, we show that when generalizing out of meta-distribution, meta-learned initializations provide only a small improvement over joint training in accuracy but require significantly fewer gradient updates. Our code and meta-learned model are available at `https://drive.google.com/drive/folders/1VhTJtYQ_byC9woS1fBaRi-hdWksfm5qq?usp=sharing`.

1 Introduction

In recent years, there has been substantial progress in high accuracy image segmentation in the high data regime (see [1] and their references). While meta-learning approaches that utilize neural network representations have made progress in few-shot image classification, reinforcement learning, and, more recently, image semantic segmentation, the training algorithms and model architectures have become increasingly specialized to the low data regime. A desirable property of a learning system is one that effectively applies knowledge gained from a few *or* many examples, while reducing the generalization gap when trained on little data and not being encumbered by its own learning routines when there are many examples. This property is desirable because training and maintaining multiple models is more cumbersome than training and maintaining one model. A natural question that arises is how to develop learning systems that scale from few-shot to many-shot settings while yielding competitive accuracy in both. One scalable potential approach that does not require ensembling many models nor the computational costs of relation networks, is to meta-learn an initialization such as via Model Agnostic Meta-Learning (MAML) [2].

In this work, we specifically address the problem of meta-learning initializations for deep neural networks that must produce dense, structured output for the semantic segmentation of images. We ask the following questions:

1. Do first-order MAML-type algorithms extend to the higher dimensional parameter spaces, dense prediction, and skewed distributions required of semantic segmentation?

Submitted to the 4th Workshop on Meta-Learning at NeurIPS 2020, Vancouver, Canada. Do not distribute.

2. How sensitive is the test-time performance of gradient-based meta-learning to the hyperparameters of the update routine used to adapt the initialization to new tasks?

3. How do first order meta-learning algorithms compare to traditional transfer learning approaches as more labeled data becomes available?

In summary, we address the above research questions as follows: We show that MAML-type algorithms do extend to few shot image segmentation, yielding state of the art results when their update routine is optimized after meta-training and when the model is regularized[1]. Because the test-time performance of MAML is inherently dependent on the neural network architecture, we developed a novel architecture built for fast learning which we call EfficientLab. Addressing question 2, we find that the meta-learned initialization's performance when being adapted to a new task is particularly sensitive to changes in the update routine's hyperparameters (see Figure 3). We show theoretically in section 3.3 and empirically in our results (see Table 3b) that a single update routine used both during meta-training and meta-testing may not have optimal generalization. Finally, we address question 3 by comparing meta-learned initializations to ImageNet [3] and joint-trained initializations in terms of both test-set accuracy and gradient updates on a novel benchmark, which we call binary PASCAL, that contains binary segmentation tasks with up to 50 training examples each. This dataset is derived from the test-set examples of the PASCAL dataset [4]. Our code and meta-learned model are available at `https://drive.google.com/drive/folders/1VhTJtYQ_byC9woS1fBaRi-hdWksfm5qq?usp=sharing`.

2 Related Work

Learning useful models from a small number of labeled examples of a new concept has been studied for decades [5] yet remains a challenging problem with no semblance of a unified solution. The advent of larger labeled datasets containing examples from many distinct concepts [6] has enabled progress in the field in particular by enabling approaches that leverage the representations of nonlinear neural networks. Image segmentation is a well-suited domain for advances in few-shot learning given that the labels are particularly costly to generate [7].

Recent work in few-shot learning for image segmentation has utilized three key components: (1) model ensembling [8], (2) the relation networks of [9] , and (3) late fusion of representations [10, 11, 7]. The inference procedure of ensembling models with a separately trained model for each example has been shown to produce better predictions than single shot approaches but will scale linearly in time and/or space complexity (depending on the implementation) in the number of training examples, as implemented in [8]. The use of multiple passes through subnetworks via iterative optimization modules was shown by [11] to yield improved segmentation results but comes at the expense of additional time complexity during inference. The relation networks proposed in [9] were recently extended to the modality of dense prediction by the authors in [11] and [7], though add complexity since they require processing each support example at test-time.

Model Agnostic Meta-Learning (MAML) is a gradient-based meta-learning approach introduced in [2] that requires no additional architectural complexity at test time. First Order MAML (FO-MAML) reduces the computational cost by not requiring backpropogating the meta-gradient through the inner-loop gradient and has been shown to work similarly well on classification tasks [2, 12]. Though learning an initialization has the potential to unify few-shot and many-shot domains, initializations learned from MAML-type algorithms have been seen to overfit in the low-shot domain when adapting sufficiently expressive models such as deep residual networks that may be more than a small number of convolutional layers [2] [13, 14]. The Meta-SGD learning framework added additional capacity by meta-learning a learning rate for each parameter in the network [15], but lacks a first order approximation. In addition to possessing potential to unify few- and many-shot domains, MAML-type algorithms are intriguing in that they impose no constraints on model architecture, given that the output of the meta-learning process is simply an initialization. Futhermore, the meta-learning dynamics, which learn a temporary memory of a sampled task, are related to the older idea of fast weights [16, 17]. Despite being dataset size and model architecture agnostic, MAML-type al-

[1] We also find that it is critical that the meta-test distribution is similar to the meta-training distribution.

[2] The original MAML and Reptile convolutional neural networks (CNNs) use four convolutional layers with 32 filters each for MiniImagenet [2, 12]

gorithms are unproven for high dimensionality of the hypothesis spaces and the skewed distributions of image segmentation problems data [10].

In recent work, [18] found that standard joint pre-training on all meta-training tasks on mini-imagenet, tiered ImageNet, and other few shot image *classification* benchmarks, with a sufficiently large network is on par with many sophisticated few-shot learning algorithms. Furthermore, implementing meta-training code comes with additional complexity. Thus it is worth testing how a vanilla training loop on the "joint" distribution of all the 760 non-test tasks compares to a meta-learned initialization.

3 Preliminaries

3.1 Generalization Error in Meta-learning

In the context of image segmentation, an example from a task τ is comprised of an image x and its corresponding binary mask y, which assigns each pixel membership to the target (ex. black bear) or background class. Examples (x, y) from the domain $\mathcal{D}_\tau$ are distributed according to $q_\tau(x, y)$, and we measure the loss $\mathcal{L}$ of predictions $\hat{y}$ generated from parameters θ and a learning algorithm U. For a distribution $p(\tau)$ over the domain of tasks $\mathcal{T}$, the parameters that minimize the expected loss are

$$\theta^* = \arg\min_\theta \mathbb{E}_p \left[\mathbb{E}_{q_\tau} \left[\mathcal{L} \left(U(\theta) \right) \right] \right] \tag{1}$$

In practice, we only have access to a finite subset of the tasks, which we divide into the training $\mathcal{T}^{tr}$, validation $\mathcal{T}^{val}$, and test tasks $\mathcal{T}^{test}$, and instead optimize over an empirical distribution $\hat{p}(\tau) := p(\tau | \tau \in \mathcal{T}^{tr})$. For examples within each available task, we can similarly define $\mathcal{D}_\tau^{tr}$, $\mathcal{D}_\tau^{val}$, $\mathcal{D}_\tau^{test}$, and the corresponding empirical distribution $\hat{q}_\tau(x, y) := q_\tau(x, y | (x, y) \in \mathcal{D}_\tau^{tr})$. As a corollary to 1, the empirically optimal initialization

$$\hat{\theta}^* = \arg\min_\theta \mathbb{E}_{\hat{p}} \left[\mathbb{E}_{\hat{q}_\tau} \left[\mathcal{L} \left(U(\theta) \right) \right] \right] \tag{2}$$

has a generalization gap that can then be expressed as

$$\mathbb{E}_p \left[\mathbb{E}_{q_\tau} \left[\mathcal{L} \left(U(\hat{\theta}^*) \right) \right] \right] - \mathbb{E}_{\hat{p}} \left[\mathbb{E}_{\hat{q}_\tau} \left[\mathcal{L} \left(U(\hat{\theta}^*) \right) \right] \right] \tag{3}$$

We include a proof in the supplementary material. The generalization gap between the actual and empirical error in meta-learning is twofold: from the domain of all tasks $\mathcal{T}$ to the sample $\mathcal{T}^{tr}$, and within that, from all examples in $\mathcal{D}_\tau$ to $\mathcal{D}_\tau^{tr}$.

3.2 Model Agnostic Meta-learning

The MAML algorithm introduced in [2] uses a gradient-based update procedure U with hyperparameters ω, which applies a limited number of training steps with a few-shot training dataset $\mathcal{D}_\tau^{tr}$ to adapt a meta-learned initialization θ to each task. To be precise, U maps from an initialization θ and examples in $\mathcal{D}_\tau$ to updated parameters θ_τ which parameterize a task-specific prediction function $f(x;\ \theta_\tau)$:

$$f(x;\ \theta_\tau) = f(x;\ U(\theta;\ \mathcal{D}_\tau)) \tag{4}$$

We adopt the shorthand $\mathcal{L}(U(\theta))$ used in [12] to indicate that the loss $\mathcal{L}$ is computed over $f(x;\ \theta_\tau)$ for $x, y \in \mathcal{D}_\tau$:

$$\mathcal{L}(U(\theta)) := \mathcal{L}(f(x;\ \theta_\tau), y) \tag{5}$$

To minimize the loss incurred in the update routine, we first take the derivative with respect to the initialization

$$\frac{\partial}{\partial \theta} \mathcal{L}(U(\theta)) = U'(\theta) \cdot \mathcal{L}'(U(\theta)) \tag{6}$$

where the resulting term U' is the derivative of a gradient based update procedure, and hence, contains second order derivatives. In first-order renditions explored in [12], FOMAML and Reptile,

finite differences are used to approximate the gradient of the meta-update $\nabla\theta$. The difference between the two approximations can be summarized by how they make use of $\mathcal{D}_\tau^{tr}$ and $\mathcal{D}_\tau^{val}$:

$$\theta^{tr} \leftarrow U(\theta \ ; \mathcal{D}_\tau^{tr}, \omega^{tr}) \tag{7}$$

$$\theta^{val} \leftarrow U(\theta^{tr} ; \mathcal{D}_\tau^{val}, \omega^{val}) \tag{8}$$

$$\theta^{both} \leftarrow U(\theta \ ; \mathcal{D}_\tau^{tr} \cup \mathcal{D}_\tau^{val}, \omega^{tr}) \tag{9}$$

Reptile trains jointly on both, while FOMAML trains on the two sets separately in sequence, favoring initializations that differ less between the splits.

$$\text{Reptile: } \nabla\theta \propto \theta^{both} - \theta \tag{10}$$

$$\text{FOMAML: } \nabla\theta \propto \theta^{val} - \theta^{tr} \tag{11}$$

The gradient approximation $\nabla\theta$ can then be used to optimize the initialization by stochastic gradient descent or any other gradient-based update procedure.

3.3 Optimizing Test-Time Update Hyperparameters

As shown clearly in equations 4 and 5, the error of any function f learned or predicted from a dataset $\mathcal{D}_\tau$ depends on the learning algorithm U. This analysis motivates research question 2 in section 1, which asks how significant is the effect of the hyperparameters of U. To address this question, we leverage the flexibility to choose hyperparameters ω^{test} when adapting to new tasks, separately from the hyperparameters ω^{tr} used in meta-training. The optimal choice of ω^{test} can be determined by minimizing the expected loss in eq. 1 with respect to the hyperparameters, treating $\hat{\theta}^*$ and $\mathcal{D}_\tau^{tr}$ as parameters of the update routine:

$$\hat{\omega}^* = \arg\min_{\omega} \mathbb{E}_{\hat{p}} \left[\mathbb{E}_{\hat{q}_\tau} \left[\mathcal{L} \left(U(\omega \ ; \hat{\theta}^*, \mathcal{D}_\tau^{tr}) \right) \right] \right] \tag{12}$$

Empirical estimations of the optimal initialization $\hat{\theta}^*$ have an implicit dependence on $\mathcal{T}^{tr}$ and ω^{tr} (eq. 2), and the optimal hyperparameters $\hat{\omega}^*$ depend on the $\hat{\theta}^*$ in turn (eq. 12). We call the general procedure of optimizing the update routine's hyperparameters to decrease meta-test-time error update hyperparameter optimization (UHO) and describe it in further detail in the appendix.

4 EfficientLab Architecture for Image Segmentation

To extend first-order MAML-type algorithms to more expressive models, with larger hypothesis spaces, we developed a novel neural network architecture, which we term EfficientLab. The top level hierarchy of the network's organization of computational layers is similar to [19] with convolutional blocks that successively halve the features in spatial resolution while increasing the number of feature maps. This is followed by bilinear upsampling of features which are concatenated with features from long skip connections from the downsampling blocks in the encoding part of the network. The concatenated low and high resolution features are then fed through a novel atrous spatial pyramid pooling (ASPP) module, which we call a residual skip decoder (RSD), and finally bilinearly upsampled to the original image size.

For the encoding subnetwork, we utilize the recently proposed EfficientNet [20]. After encoding the images, the feature maps are upsampled through a parameterized number of RSD modules. The RSD computational graph of operations is shown in Figure 1. EfficientLab-3 has one RSD module at the third stage while EfficientLab-6-3 has RSD modules at the 6th and 3rd stages as shown in figure 1. The RSD module utilizes three parallel branches of a 1×1 convolution, 3×3 convolution with dilation rate = 2, and a simple average-pooling across spatial dimensions of the feature maps. The output of the three branches is concatenated and fed into a final 3×3 convolutional layer with 112 filters. A residual connection wraps around the convolutional layers to ease gradient flow [3]. Before the final 1×1 convolution that produces the unnormalized heatmap of class scores, we use a single layer of dropout with a drop rate probability = 0.2 [4]. We use the standard softmax to produce the normalized predicted probabilities.

[3] Residual connections have been suggested to make the loss landscape of deep neural networks more convex [21]. If this is the case, it could be especially helpful in finding low-error minima via gradient-based update routines such as those used by MAML, FOMAML, and Reptile.

[4] As described in [22] and used in [20] the dropout layer is applied after all batch norm layers.

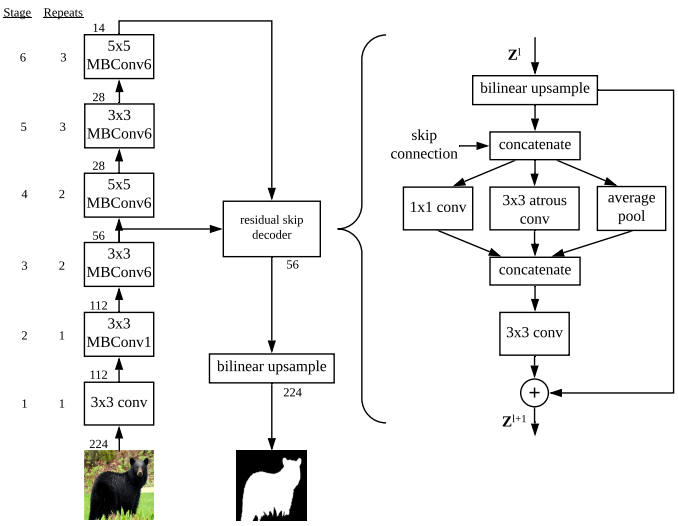


Figure 1: Diagram of the computations performed by the EfficientLab-3 neural network. Nodes represent functions and edges represent output tensors. Output spatial resolutions are written next to the output edge. The high level architecture shows the EfficientNet feature extractor on the left with mobile inverted bottleneck convolutional blocks (see [20, 23] for more details). On the right is the residual skip decoder (RSD) module that we utilize in the upsampling branch of EfficientLab. The numbers suffixing EfficientLab denote the stage at which RSDs are located, with EfficientLab-3, having an RSD with skip connections from stage 3 in the downsampling layers.

We use batch normalization layers following convolutional layers [24]. We meta-learn the β and γ parameters, adapt them at test time to test tasks, and use running averages as estimates for the population mean and variance, $E[x]$ and $Var[x]$, at inference time as suggested in [25]. *All* parameters at the end of an evaluation call are reset to their pre-adaptation values to stop information leakage between the training and validation sets. The network is trained with the binary cross entropy minus the log of the dice score [26] , which we adapt from the loss function of [27], plus an L_2 regularization on the weights:

$$\mathcal{L} = H - log(J) + \lambda \left\|\theta\right\|_2^2 \tag{13}$$

where H is binary cross entropy loss:

$$H = -\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log \hat{y}_i + (1-y_i)\log\left(1-\hat{y}_i\right)\right) \tag{14}$$

J is the modified Dice score:

$$J = \frac{2IoU}{IoU+1} \tag{15}$$

and IoU is the intersection over union metric:

$$IoU = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i\hat{y}_i + \epsilon}{y_i + \hat{y}_i - y_i\hat{y}_i + \epsilon}\right) \tag{16}$$

5 Experiments

We evaluate the FOMAML and Reptile meta-learning algorithms on the FSS-1000 and binary PASCAL datasets. Model topology development and meta-training hyperparameter search was done on the held out set of validation tasks, $\mathcal{T}^{val}$, and not the final test tasks. For the final evaluations reported in Table 3b, we meta-train for 50,000 meta-iterations, which is $\sim$330 epochs through the training and validation tasks $\mathcal{T}^{tr} \cup \mathcal{T}^{val}$ of the FSS-1000 dataset, using a meta-batch size of 5, an inner loop batch size of 8, and 5 inner loop iterations. For reptile, we experiment with setting the

train shots to 5 and 10. During training, we use stochastic gradient descent (SGD) in the inner loop with a fixed learning rate of 0.005. During training and evaluation, we apply simple augmentations to the few-shot examples including random translation, rotation, horizontal flips, additive Gaussian noise, brightness, and random eraser [28]. We use L_2 regularization on all weights with a coefficient $\lambda = 5\text{e}{-4}$.

5.1 Joint-Trained Initialization

We trained EfficientLab on the "joint" distribution of $\mathcal{T}^{tr} \cup \mathcal{T}^{val}$ in a standard training loop, without an inner loop. Each batch contained a random sample of examples from any of the classes as is standard in SGD. The only change to the network architecture was that instead of predicting 2 channel output (foreground and background), the network was trained to predict the number of task classes plus a background class. Other than these changes, we matched meta-training hyperparameters as faithfully as possible: training for 200 epochs, batch size of 8 image-mask pairs, using a learning rate of 0.005 with a linear decay, and regularization.

5.2 Results

We show the results of experimenting with different decoder architectures for EfficientLab in Table 1. The bulk of our experiments were done with EfficientLab-3, including all binary PASCAL experiments, though our best results were found using EfficientLab-6-3.

Network Architecture	$\overline{\textbf{IoU}}$
Auto-DeepLab decoder	$71.16 \pm 1.03\%$
RSD at Stage 3 w/o residual	$77.55 \pm 1.08\%$
RSD at Stage 3	$79.89 \pm 0.98\%$
RSD at Stages 6 & 3	$80.43 \pm 0.91\%$

Table 1: **EfficientLab architecture ablations**. Each network is meta-trained in the same way following Section 5 and tested on the set of test tasks from FSS-1000 [7]. The row "RSD at Stage 3 w/o residual" contains results of removing the short-range residual connection from our proposed RSD module. The final row is the best network we find for 5-shot performance via model agnostic meta-learning.

The results of our model with an initialization meta-learned using Reptile and FOMAML are shown in Table 3b. We find that EfficientLab trained with FOMAML and importantly with an adaptation routine optimized for low out of distribution test error and regularization yields state of the art results on the FSS-1000 dataset. Given that previous works have used regularization minimally or not at all during meta-training, we also conducted an ablation of removing regularization on the model. We find, unsurprisingly, that the combination of an $L2$ loss on the weights, with simple augmentations, and a final layer of dropout significantly increases generalization performance. We have included a visualization of example predictions for a small set of randomly sampled test tasks in 2. See the supplementary material for additional examples and failure cases.

Importantly, we also find that the original definition of FOMAML in which the mini-datasets D^{tr} and D^{val} are disjoint yields worse results than sampling with some amount of overlap. We find that by sampling D^{tr} and D^{val} with replacement from $D^{tr} \cup D^{val}$ yields better results. This sampling procedure is denoted by FOMAML* below and can be interpreted as a stochastic interpolation between the original Reptile and FOMAML definitions put forth in [12]. We suspect that this sampling strategy serves as a form of meta-regularization though further work would be required on this detail to be conclusive. Similarly, we find that meta-training with Reptile using all 10 examples in each task of FSS-1000 produces worse results than meta-training with 5 examples per task in each meta-example.

To address research question 2 in section 1, we also searched through a range of update routine learning rates, α, that were $10\times$ less to $10\times$ greater than the learning rate used during meta-training. As clearly shown in Figure 3, the learned representations are **not** robust to such large variations in the hyperparameter. We find that: (1) the estimated optimal hyperparameters for the update routine on the validation tasks are not the same as those specified a priori during meta-training, as

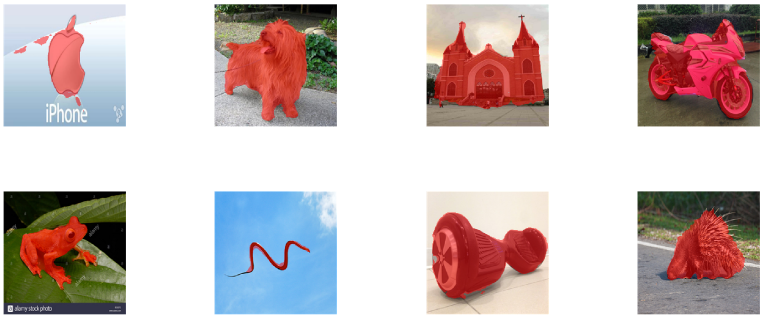
Method	$\overline{\text{IoU}}$
FSS-1000 Baseline	73.47%
ImageNet-trained encoder	$42.46 \pm 1.40\%$
Joint-trained	28.07 ± 0.99
Joint-trained + UHO	$32.07 \pm 1.17\%$
Reptile	73.99 ± 1.38
FOMAML*	$75.87 \pm 1.10\%$
FOMAML* + UHO	$\mathbf{76.45 \pm 1.16\%}$

(a) FSS-1000 1-shot

Method	$\overline{\text{IoU}}$
FSS-1000 Baseline	80.12%
ImageNet-trained encoder	$50.26 \pm 1.41\%$
Joint-trained on FSS-1000	$25.03 \pm 0.94\%$
Joint-trained on FSS-1000 + UHO	$45.05 \pm 1.37\%$
Reptile	$79.78 \pm 0.95\%$
FOMAML $D^{tr} \cap D^{val} = \varnothing$	$75.02 \pm 1.07\%$
FOMAML* - regularization	$77.89 \pm 1.03\%$
FOMAML*	$79.89 \pm 0.98\%$
FOMAML* + UHO	$81.36 \pm 0.80\%$
EffLab-6-3 FOMAML* + UHO	$\mathbf{82.78 \pm 0.74\%}$

(b) FSS-1000 5-shot

Table 2: **Training paradigms**. Mean IoU scores of the EfficientLab-3 and EfficientLab-6-3 network evaluated on FSS-1000 test set of tasks for 1-shot and 5-shot learning. We report the FSS-1000 baseline from [7]. Our best found model combined FOMAML*, EfficientLab-6-3, regularization, and UHO. FOMAML $D^{tr} \cap D^{val} = \varnothing$ denotes the original definition of FOMAML put forth in [12] in which D^{tr} and D^{val} are completely disjoint while FOMAML* denotes that the two mini-datasets have been sampled with replacement from $D^{tr} \cup D^{val}$.



.

Figure 2: Randomly sampled example 5-shot predictions on the test images from test tasks. Positive class prediction is overlaid in red. From left to right, top to bottom, the classes are `apple_icon`, `australian_terrier`, `church`, `motorbike`, `flying_frog`, `flying_snakes`, `hover_board`, `porcupine`

illustrated in Figure 3. (2) Optimizing the hyperparameters after meta-training improves test-time results on unseen tasks. Furthermore, we find that *meta-training* from scratch (and evaluating) with the UHO-selected hyperparameters yields nearly identical results to meta-training with the initial hyper parameters learning rate = 0.005 and inner-iterations = 5. This further suggests that it may be useful to tune the hyperparameters ω *after* meta-training to improve the generalization performance of the gradient-based adaptation routine U.

When evaluating meta-learned, joint-trained, and ImageNet-pretrained EfficientLab initializations on the binary PASCAL dataset, we found that meta-learned initializations provided only a small improvement over joint training in terms of accuracy but required significantly fewer gradient updates. While the number of gradient updates required by the FOMAML initialization was substantially lower than those required by the ImageNet-pretrained initialization, we found that the joint-trained initialization required equal or fewer gradient updates on average for $k > 30$. These results suggest that there may be a Goldilocks zone in terms of number of labeled examples expected to be available at meta-test time for meta-learned models.

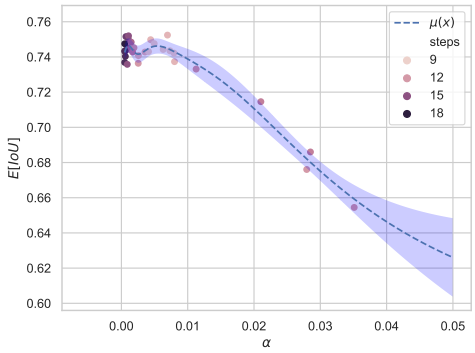


Figure 3: Mean IoU is shown as a function of the learning rate α and the number of gradient steps over the set of 200 validation tasks $\mathcal{T}^{val}$. During optimization of the learning rate and the number of steps, the relationship between the learning rate and the IoU is modeled as a Gaussian process (shown in blue dashed line with 95% confidence interval). Points are colored by median of the iterations each task was trained for before stopped by early stopping.

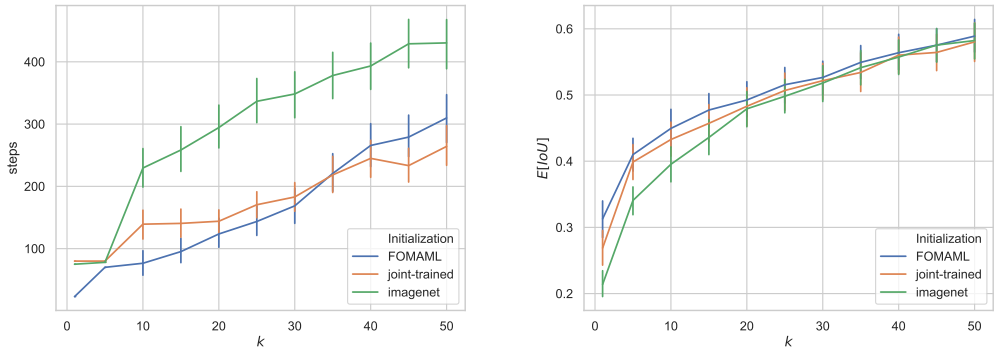


Figure 4: Gradient steps and mean IoU results as a function of the training set size (k) of our EfficientLab-3 model adapted to tasks of the binary-PASCAL dataset. On the left we show the estimated optimal number of gradient steps for value of k across the three pre-training paradigms. For $k \leq 5$, we estimate the optimal number of gradient updates when adapting to a new task by using UHO as described in A.1, while for $k > 5$ we use early stopping on 20% of k. On the right we show mean IoU evaluated over $\mathcal{D}_{\tau}^{test}$.

6 Conclusions

In this work, we showed that gradient-based first order model agnostic meta-learning algorithms extend to the high dimensionality of the hypothesis spaces and the skewed distributions of few-shot image segmentation problems, but are sensitive to the hyperparameters, ω, of the update routine, U. When generalizing out of meta-distribution on the binary PASCAL dataset, we found that the model produced by FOMAML required significantly fewer gradient updates and reached slightly higher accuracy than both FSS-1000 joint-trained and ImageNet-pretrained initializations. These results provide important context on the value in terms of both labeled data and computational efficiency of applying FOMAML to new image segmentation tasks. Future work should investigate more critically, both empirically and theoretically, the efficacy of few-shot learning systems at generalizing out of meta-distribution and, in particular, as more labeled data becomes available. Lastly, we hope that this work draws attention to the open problem of building learning systems that can unify small and large data regimes by gaining expertise and integrating new information as more data becomes available, much as people do.

References

[1] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–92, 2019.

[2] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning (ICML)*, 2017.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[4] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[5] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems*, pages 640–646, 1996.

[6] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[7] Tianhan Wei, Xiang Li, Yau Pun Chen, Yu-Wing Tai, and Chi-Keung Tang. Fss-1000: A 1000-class dataset for few-shot segmentation. *arXiv preprint arXiv:1907.12347*, 2019.

[8] Amirreza Shaban, Shray Bansal, Zhen Liu, Irfan Essa, and Byron Boots. One-shot learning for semantic segmentation. *arXiv preprint arXiv:1709.03410*, 2017.

[9] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[10] Kate Rakelly, Evan Shelhamer, Trevor Darrell, Alyosha Efros, and Sergey Levine. Conditional networks for few-shot semantic segmentation, 2018.

[11] Chi Zhang, Guosheng Lin, Fayao Liu, Rui Yao, and Chunhua Shen. Canet: Class-agnostic segmentation networks with iterative refinement and attentive few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5217–5226, 2019.

[12] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2018.

[13] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

[14] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

[15] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[16] Geoffrey E Hinton and David C Plaut. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pages 177–186, 1987.

[17] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, pages 4331–4339, 2016.

[18] Guneet S Dhillon, Pratik Chaudhari, Avinash Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. *arXiv preprint arXiv:1909.02729*, 2019.

[19] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[20] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[21] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

[22] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2682–2690, 2019.

[23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[25] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. *arXiv preprint arXiv:1810.09502*, 2018.

[26] Lee R Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[27] Vladimir Iglovikov, Sergey Mushinskiy, and Vladimir Osin. Satellite imagery feature detection using deep convolutional neural network: A kaggle competition. *arXiv preprint arXiv:1706.06169*, 2017.

[28] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017.

[29] OpenAI. supervised-reptile. `https://github.com/openai/supervised-reptile`, 2018.

[30] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[31] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[32] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36, 2012.

[33] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[34] Samet Oymak, Zalan Fabian, Mingchen Li, and Mahdi Soltanolkotabi. Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian. *arXiv preprint arXiv:1906.05392*, 2019.

[35] Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*, 2017.

[36] Yoshua Bengio, Tristan Deleu, Nasim Rahaman, Rosemary Ke, Sébastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, and Christopher Pal. A meta-transfer objective for learning to disentangle causal mechanisms. *arXiv preprint arXiv:1901.10912*, 2019.

Appendix A Meta-Training Details

For meta-training, we adapted the code in [29]. We referenced the hyperparameters used in the Reptile meta-training runs for Mini-ImageNet in [12]. Due to the computational cost of meta-training and the combinatorial expansion in the meta-training hyperparameter search space due to having effectively the same number of hyperparameters for both the outer and inner meta-training loops, we did not exhaustively search over all meta-training hyperparameters. We did initial experimentation on tuning the inner batch size and the number of inner-loop gradient steps by evaluating on the validation tasks $\mathcal{T}^{val}$, though found that fine-tuning these values mattered less than optimizing the test-time hyperparameters.

Table 3: Meta-training hyperparameters for Reptile and FOMAML algorithms.

Hyperparameter	value
Meta-batch size	5
Meta-steps	50000
Initial meta-learning rate	0.1
Final meta-learning rate	$1.e-5$
Inner batch size	8
Inner steps	5
Inner learning rate	0.005
Final layer dropout rate	0.2
Augmentation rate	0.5

Both Reptile and FOMAML were meta-trained with the hyperparameters shown in Table 3. The only hyperparameter that we found significantly changed the results between the two approaches was the use of the "train-shots", which is the number of examples that each inner batch can sample from. We found that setting the Reptile train-shots equal to 10, which is the total number of examples per task in the FSS-1000 dataset, significantly reduced test-time performance. By decreasing the train-shots to 5, mIoU increased by $\approx 5\%$ absolute percentage points of mean intersection over union (IoU). Similarly, we found that if we sampled D^{tr} and D^{val} with replacement, as opposed to using all 10 train-shots for FOMAML (using 5 for D^{tr} and 5 for $Dval$), meta-test results increased by $\approx 5\%$ absolute percentage points of mean IoU. We suspect that limiting the number of train shots in this way serves as a form of meta-regularization.

A.1 Test-Time Update Hyperparameter Optimization Methodology

Generalization in meta-learning requires both the ability to learn representations for new tasks efficiently ($\mathcal{T}^{tr}$ to $\mathcal{T}^{test}$), and to select representations that are able to capture unseen test examples effectively ($\mathcal{D}_\tau^{tr}$ to $\mathcal{D}_\tau^{test}$). The approximation scheme of FOMAML addresses the latter by taking the finite difference between updates using the train and validation sets (as shown in eq. 11), favoring initializations that differ less between splits of $\mathcal{D}_\tau^{tr} \cup \mathcal{D}_\tau^{val}$. In investigation of research question 2 in section 1 and to further improve generalization within task to $\mathcal{D}_\tau^{test}$, we tune ω after meta-learning $\hat{\theta}^*$ to find ω^{test} (as shown in eq. 12). We use ω^{test} at meta-test time when adapting the initialization to new tasks. We call this procedure update hyperparameter optimization (UHO). Specifically, we use Bayesian optimization with Gaussian processes to optimize the hyperparameters ω [30]. We apply this UHO procedure to estimate the optimal adaption routine's hyperparameters using 200 randomly validation tasks $\mathcal{T}^{val}$ that are held our from meta-training. We specifically search over the learning rate and the number of gradient updates that are applied when adapting to a new task τ. We report results with and without optimized update hyperparameters in table 3b. We find that optimizing ω significantly improves adaptation performance on the meta-test tasks $\mathcal{T}^{test}$.

For the joint-trained and meta-learned initializations evaluated in Table 2, we experimented with tuning their learning rate by evaluating 30 parameters on each of the 200 validation tasks. The first half of the parameters were randomly from a log-uniform distribution and the second half were sampled from the posterior of the GP to maximize expected improvement in mean IoU over the validation tasks. For all evaluations, we optimize the learning rate over the interval $[0.0005, 0.05]$.

Because the effects of the learning rate are intertwined with the number of gradient updates, we also leveraged early stopping to decrease runtime to more efficently estimate the optimal number of gradient steps when adapting to a new task. The use of early stopping in this way is purely a runtime optimization that reduces the search space that is explored when tuning ω. We train each validation set $\mathcal{T}^{val}$ task independently and record the optimal number of gradient updates for each task. We then evaluate all tasks in $\mathcal{T}^{val}$ at the median number of steps returned by early stopping across tasks in $\mathcal{T}^{val}$. We could have also used the Bayesian optimization with GP prior, but early stopping has the advantage of computational efficiency. Early stopping has been deeply studied with strong empirical and theoretical evidence to support its efficacy as an efficient hyperparameter tuning algorithm [31, 32, 33, 34].

Table 4: Inference hyperparameters returned from BO with GP for initializations of the EfficientLab-3 network. All other hyperparameters were fixed to the values shown in Table 3.

Initialization	learning rate	steps
Joint-trained 1-shot	$8.156e-4$	10
Joint-trained 5-shot	$1.364e-3$	17
FOMAML* 1-shot	$1.734e-3$	8
FOMAML* 5-shot	$6.951e-3$	12

For our largest model, EfficientLab-6-3, we also experimented with using BO on a larger set of hyperparameters and larger number of maximum iterations for early stopping. We search over the learning rate $[0.0005, 0.05]$, the final layer dropout rate $[0.2, 0.5]$, the augmentation rate $[0.5, 1.0]$, and batch size $[1, 10]$ with a maximum early stopping iterations of 80.

Table 5: Inference hyperparameters returned from BO with GP for the EfficientLab-6-3 network.

Initialization	learning rate	steps	dropout rate	augmentation rate	batch size
FOMAML* 5-shot	$5e-4$	59	0.5	0.5	8

Appendix B Example predictions

We have included in Figure 5 a visualization of additional, randomly sampled predictions on test examples $\mathcal{D}^{test}$ from test tasks $\mathcal{T}^{test}$ that were never seen during meta-training. The failure cases are particularly interesting in that they suggest that a foreground object-ness prior has been learned in the meta-learned initialization.

Figure 5: Randomly sampled example 5-shot predictions on the test images from test tasks. Predictions were generated by EfficientLab-6-3 model meta-trained with FOMAML* and evaluated with UHO-returned hyperparameters. Positive class prediction is overlaid in red. From left to right, top to bottom, the classes are `abes_flying_fish`, `australian_terrier`, `flying_frog`, `grey_whale`, `hoverboard`, `manatee`, `marimba`, `porcupine`, `sealion`, `spiderman`, `stingray`, `tunnel`. The final row contains hand-picked failure cases from tasks `american_chameleon`, `marimba`, `motorbike`, and `porcupine`.

Appendix C Datasets

C.1 FSS-1000 Dataset

The first few-shot image segmentation dataset was the PASCAL-5^i presented in [8] which reimagines the PASCAL dataset [4] as a few-shot binary segmentation problem for each of the classes in the original dataset. Unfortunately, the dataset contains relatively few distinct tasks (20 excluding background and unlabeled). The idea of a meta-learning dataset for image segmentation was further developed with the recently introduced FSS-1000 dataset, which contains 1000 classes, 240 of which are dedicated to the meta-test set $\mathcal{T}^{test}$, with 10 image-mask pairs for each class [7]. For each of the rows in the results table 3b, we evaluate the network on the 240 test tasks, sampling two random splits into training and testing sets for each task, yielding 480 data points per meta-learning approach for which the mean intersection over union (IoU) (eq. 16) and 95% confidence interval are reported. The FSS-1000 dataset is the focus of the empirical comparisons of network ablations and meta-learning approaches that we experiment with in this paper.

C.2 Binary PASCAL Dataset

For investigating how the meta-learned representations integrate new information as more data becomes a available, we constructed a novel benchmark dataset that we call binary PASCAL. In binary PASCAL, a binary segmentation model is evaluated across the test-set examples of all classes in PASCAL [4]. During evaluation, we simply randomly sample 20 test examples and sample a training set of k examples over the range $[1, 5, ..., 45, 50]$.

Using this dataset, we train over a range of "k"-training shots from ImageNet-trained [5], joint-trained, and our meta-learned initializations. We report the performance of our EfficientLab network meta-trained with FOMAML over a range of k examples as a benchmark which we hope will inspire future empirical research into studying how meta-learning approaches scale in accuracy and computational complexity as more labeled data become available. For all three initializations, we use UHO for estimating the hyperparameters of U for $k < 10$. For $k \geq 10$, we use a fixed learning rate and early stopping evaluated on 20% of the examples to estimate the optimal number of iterations. These results are shown in Figure 4 and discussed in 5.2.

Appendix D Binary PASCAL Experimental Details

In this section, we describe our testing protocol for evaluating the initializations when adapting to the tasks from the Binary PASCAL dataset. For each tuple of (initialization, k-training shots) we randomly sample 20 examples for a test set $\mathcal{D}^{test}$ for the task and train on k labeled examples $\mathcal{D}^{tr}$. We repeat this random sampling and training process 4 times for each of the 5 tasks, yielding 20 evaluation samples per (initialization, k-training shots) tuple. For all three initializations, we use UHO for estimating the hyperparameters of U for $k < 10$. For $k \geq 10$, we use a fixed learning rate equal to the value used during meta-training and early stopping to estimate the optimal number of iterations. For early stopping, we use 20% of the examples in $\mathcal{D}^{tr}$ to form $\mathcal{D}^{val}$.

Appendix E Proof of Generalization Gap

The generalization gap is the difference between an estimate of the error of a function on an empirical dataset and the (typically non-computable) error over the true distribution [35]. We define the generalization gap as the difference between the expected loss a model f incurs over the true distribution p and the loss measured on a dataset $\hat{p}$

$$\mathbb{E}_p\left[\mathcal{L}\left(\hat{f}\right)\right] - \mathbb{E}_{\hat{p}}\left[\mathcal{L}\left(\hat{f}\right)\right] \tag{17}$$

In meta-learning, $\hat{f}$ is learned on a distribution of examples $\hat{q}_\tau$ sampled from a distribution over tasks $\hat{p}$. Thus there is a function $\hat{f}_\tau$ that is learned on each $\hat{q}_\tau$

$$\mathbb{E}_p\left[\mathbb{E}_{q_\tau}\left[\mathcal{L}\left(\hat{f}_\tau\right)\right]\right] - \mathbb{E}_{\hat{p}}\left[\mathbb{E}_{\hat{q}_\tau}\left[\mathcal{L}\left(\hat{f}_\tau\right)\right]\right] \tag{18}$$

Without loss of generality, we can define an update operator U which maps from a training distribution $q_\tau(x, y)$ and a parameter vector θ to a function $\hat{f}_\tau$:

$$\hat{f}_\tau = U(q_\tau; \theta) \tag{19}$$

To preserve generality, U can be any arbitrary operator that returns a function $\hat{f}_\tau$. Replacing $\hat{f}_\tau$ with U and dropping $q_\tau(x, y)$ for brevity:

$$\mathbb{E}_p\left[\mathbb{E}_{q_\tau}\left[\mathcal{L}\left(U(\theta)\right)\right]\right] - \mathbb{E}_{\hat{p}}\left[\mathbb{E}_{\hat{q}_\tau}\left[\mathcal{L}\left(U(\theta)\right)\right]\right] \tag{20}$$

We can, further, use a meta-learning algorithm to learn an initialization $\hat{\theta}^*$ that we estimate to be optimal on some dataset of tasks $\mathcal{T}$

$$\mathbb{E}_p\left[\mathbb{E}_{q_\tau}\left[\mathcal{L}\left(U(\hat{\theta}^*)\right)\right]\right] - \mathbb{E}_{\hat{p}}\left[\mathbb{E}_{\hat{q}_\tau}\left[\mathcal{L}\left(U(\hat{\theta}^*)\right)\right]\right]. \tag{21}$$

[5] The encoder is trained on ImageNet, while the residual skip decoder and final layer weights are initialized in the same way as EfficientNet [20]

Appendix F Analysis of Weight Updates

In this section, we quantitatively compare the solutions learned by FOMAML to those learned by standard joint training with SGD. We find empirical evidence that gradient-based mete-learning algorithms converge to a point in parameter space that is significantly closer in expectation to each task τ's manifold of optimal solutions for $\tau \in \mathcal{T}$. This builds on the theoretical analysis in Section 5.2 of [12]. First we compute the Euclidean distance between the entire EfficientLab-3 parameter vectors from an initialization θ to an updated weight vector θ_τ after 5 gradient steps on 5 training examples $\mathcal{D}^{tr}$ from meta-test tasks $\tau \in \mathcal{T}^{test}$:

$$d_1 = \|\theta - \theta_\tau\|_2 \tag{22}$$

We compute this distance twice on a random train-test split of the 10 examples for all 240 FSS-1000 test tasks, yielding 480 updated weight vector samples for each of the two meta-learned and joint-trained initializations.

Initialization method	$E_{\mathcal{T}^{test}}[d_1]$
Joint-trained	0.995 ± 0.022
Meta-learned	0.169 ± 0.008

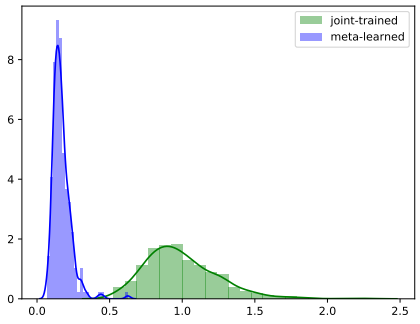


Figure 6: **Left:** Average Euclidean distance between initialization and updated weights with 95% confidence interval. **Right:** Distributions of Euclidean distances between initialization and updated weights.

Let $\mathbf{v} \in \mathbb{R}^n$ be a weight vector that is a subvector from an initialization θ and $\mathbf{u}_\tau \in \mathbb{R}^n$ be the updated weight vector after gradient steps on examples sampled from q_τ. The subvectors $\mathbf{v}$ and $\mathbf{u}_\tau$ represent the weight tensors from an EfficientLab block unrolled into a vector. In Figure 7, we show the unit-normalized Euclidean distance between EfficientLab blocks, where a block is either the stem convolutional block, a mobile inverted bottleneck convolutional block [20, 23], or our residual skip decoder:

$$d_2 = \left\| \frac{\mathbf{v}}{\|\mathbf{v}\|_2} - \frac{\mathbf{u}_\tau}{\|\mathbf{u}_\tau\|_2} \right\|_2 \tag{23}$$

We also plot the mean absolute difference to get a sense of the absolute distance traveled by individual parameters:

$$d_3 = \frac{1}{n} \sum_{i=1}^{n} |\mathbf{v} - \mathbf{u}_\tau|_i \tag{24}$$

As shown in Figure 7, we find that the joint-trained initialization travels significantly further when adapted to tasks from $\mathcal{T}^{test}$, even though the same learning rate and number of gradient steps are used at test time for both initializations. This implies that stable minima that produce low error lie closer in expectation over tasks from $\mathcal{T}^{test}$ to the meta-learned initialization. Evidence for this interpretation is further found in the test time metrics when evaluating with 5 gradient steps which show that the meta-learned initialization has a pixel-error rate that is 3.6 times smaller. This difference in error rate is found by comparing the joint-trained and FOMAML* methods in Table 2.

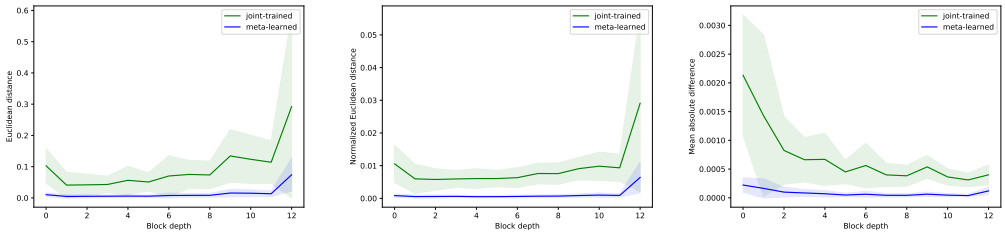


Figure 7: Differences between weights at initialization and after 5 gradient steps with a step size of 0.005 on 5 training examples $\mathcal{D}^{tr}$ from test tasks $\mathcal{T}^{test}$. The left column shows the Euclidean distance (d_1) between EfficientLab-3 block weight vectors before and after training on $\mathcal{D}^{tr}$. Middle column shows Euclidean distance between unit norm weight vectors (d_2). Right column shows the mean absolute difference (d_3) between individual parameters in an EfficientLab block.

The results in Figure 7, show that the adaptation to new tasks for both pre-training methods is non-uniform. The largest relative changes are shown in the final layers due to changes in the directionality of the EfficientLab weight subvectors. In contrast, the largest absolute changes in individual parameter values are found in the early layers of the EfficientLab model. Both initializations demonstrate similar patterns in the distribution of weight updates as a function of block depth but the changes in weights are up to an order of magnitude higher for the joint-trained initialization for all three difference metrics we investigated. The large difference between joint-trained and meta-learned distances is also in line with recent results of [36] that show that when the knowledge of a model is factorized properly, the expected gradient over parameters when adapting to new tasks will be closer to zero.