Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML

Aniruddh Raghu *
MIT
araghu@mit.edu

Maithra Raghu *
Cornell University & Google Brain
maithrar@gmail.com

Samy Bengio
Google Brain

Oriol Vinyals
DeepMind

Abstract

Model Agnostic Meta Learning (MAML) is a highly popular algorithm for few shot learning. MAML consists of two optimization loops; the outer loop finds a meta-initialization, from which the inner loop can efficiently learn new tasks. Despite MAML's success, a fundamental open question remains – is the algorithm's effectiveness due to the meta-initialization being primed for *rapid learning* (large, efficient changes in the representations given the task) or due to *feature reuse*, with the meta initialization already containing high quality features? We investigate this question, finding that feature reuse is the dominant factor. This leads to the ANIL (Almost No Inner Loop) algorithm, a simplification of MAML where we remove the inner loop for all but the (task-specific) head of the network. We further study the contributions of the head and body of the network, showing that performance on the test tasks is entirely determined by the quality of the learned features, and we can remove even the head of the network (the NIL algorithm).

1 Introduction

Metalearning approaches [13, 30, 26, 4, 24, 22, 19] have been widely used to tackle the *few-shot learning* problem. These approaches define a family of tasks, some of which are used for training and others solely for evaluation. A metalearning algorithm looks at learning properties that generalize across the training tasks, and result in fast and efficient learning of the evaluation tasks.

One highly successful meta-learning algorithm has been *Model Agnostic Meta-Learning (MAML)* [4] MAML comprises of two optimization loops; the outer loop aims to find an effective *meta-initialization*, from which the inner loop can perform standard optimization to learn new tasks with very few labelled examples. This algorithm, with deep neural networks as the underlying model, has been highly influential, with significant follow on work [19, 6, 23, 11, 5, 9, 28, 31, 12].

Despite the popularity of MAML, there remains a fundamental open question on the basic algorithm. Does the meta-initialization learned by the outer loop result in *rapid learning* (efficient but significant changes in the representations) on the unseen tasks or is the success primarily due to *feature reuse* (with the meta-initialization already having high quality representations)? In this paper, we explore this question and its many surprising consequences. Our main contributions are:

- We perform layer freezing experiments and latent representational analysis of MAML, finding that feature reuse is the predominant reason for efficient learning.
- Building on this insight, we propose the *ANIL (Almost No Inner Loop) algorithm*, a simplification to MAML that removes the inner loop updates for all but the head (final layer) of the neural network. ANIL performs identically to MAML on few-shot classification and RL tasks.

*Equal contribution

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

- We further study the effect of the head of the network, finding that it is important for task-specific alignment at training, but can be removed at test time, yielding the *NIL (No Inner Loop) algorithm*.

Related Work: Although the MAML algorithm has seen significant recent developments [1, 6, 9, 23], there is little work investigating why the double-optimization loop structure of the algorithm is empirically highly effective. We address this shortcoming here and analyze *why* MAML leads to effective few shot learning, and propose a simplification that almost completely removes the inner optimization loop, with no reduction in performance.

2 Rapid Learning or Feature Reuse in MAML

Overview of MAML: The MAML algorithm finds an *initialization* for a neural network so that new tasks can be learnt with very few examples. This is done via two optimization loops:

- **Outer Loop:** Updates the initialization of the neural network parameters (often called the *meta-initialization*) to a setting that enables fast adaptation to new tasks.
- **Inner Loop:** Performs *adaptation*: take the outer loop initialization, and, *separately for each task*, perform a few gradient updates over k labelled examples provided for adaptation (the support set).

More details are given in Appendix A.

Our goal is to understand whether the MAML algorithm efficiently solves new tasks due to *rapid learning* or *feature reuse*. In rapid learning, large representational and parameter changes occur during adaptation to each new task as a result of favorable weight conditioning from the meta-initialization. In feature reuse, the meta-initialization already contains highly useful features that can mostly be reused as is for new tasks, so little task-specific adaptation occurs. Figure 1 shows a schematic of these two hypotheses.

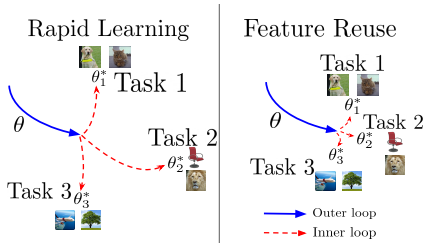

Figure 1: **Rapid learning and feature reuse paradigms.** In Rapid Learning, outer loop training leads to a parameter setting that is well-conditioned for fast learning, and inner loop updates result in significant task specialization. In Feature Reuse, the outer loop leads to parameter values corresponding to transferable features, from which the parameters do not move significantly in the inner loop. Images from [10, 7, 29, 2, 17, 27].

In investigating the rapid learning vs feature reuse question, there is an important distinction between the *head* (final layer) of the network and the earlier layers (the *body* of the network). In each few-shot learning task, the alignment changes between the output neurons and classes. For instance, in task $\mathcal{T}_1$, the (wlog) five output neurons might correspond, in order, to the classes (dog, cat, frog, cupcake, phone), while for a different task, $\mathcal{T}_2$, they might correspond, in order, to (airplane, frog, boat, car, pumpkin). This means that the head must necessarily change for each task to learn the new alignment, and for the rapid learning vs feature reuse question, we are primarily interested in the behavior of body of the network. We return to this in more detail in Section 4, where we present an algorithm (NIL) that does not use a head at test time.

We study the rapid learning vs feature reuse question through two sets of experiments:

- **Freezing:** We freeze a contiguous subset of layers of the network, excluding the head (final layer), during the inner loop. These frozen layers are not updated at inference time in the inner loop, and must reuse the features learned by the meta-initialization from the outer loop. We compare the few shot learning accuracy when freezing to the accuracy when allowing inner loop adaptation. Note that the inner loop adaptation still exists at *training time*; only at inference time do we prevent it to gauge the quality of the learned features *after* meta-initialization.
- **Representational Similarity:** We use representational similarity tools to directly analyze how much the network features and representations change through the inner loop. Following several recent works [20, 25, 18, 16, 21, 8, 3] we measure this by applying Canonical Correlation Analysis (CCA) to the latent representations of the network (see [20, 18] for details about CCA). We

compare the representational similarity between each layer's activations before and after the inner loop update step, averaging the similarity score across different random seeds and tasks.

We take MAML models at 10000, 20000, and 30000 iterations into training on the MiniImageNet-5way5shot benchmark, and perform the above experiments. Results are presented in Figure 2. We see that over training, even when freezing all layers but the head, performance hardly changes. This demonstrates that the meta-initialization has already learned good enough features that can be reused without any adaptation across the different tasks.

The representational similarity results show that from early on in training, the CCA similarity between activations pre and post inner loop update on MiniImageNet-5way-5shot is very high for all layers but the head indicating that lower layers' representations hardly change at all during inner loop adaptation. These results strongly support the feature reuse hypothesis: layers don't change rapidly at adaptation time; they already contain good features from the meta-initialization. Further results supporting the hypothesis are in Appendix D.4.

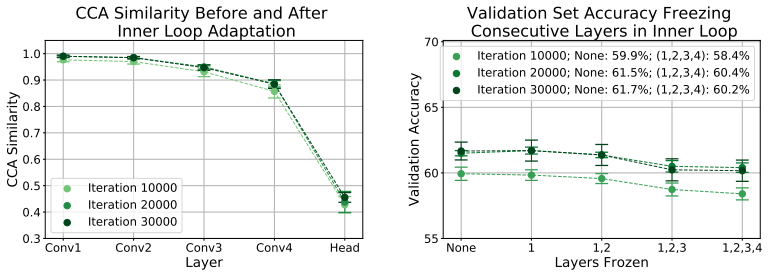

Figure 2: **Inner loop updates have little effect on learned representations from early on in learning.** Right pane: we freeze contiguous blocks of layers (no adaptation at test time), on MiniImageNet-5way-5shot and see almost identical performance. Left pane: representations of all layers except the head are highly similar pre/post adaptation – i.e. features are being reused. This is true from very early (iteration 10000) in training.

3 The ANIL (Almost No Inner Loop) Algorithm

We have seen that for all layers except the head (final layer) of the neural network, the meta-initialization learned by the outer loop of MAML results in very good features that can be reused as is on new tasks; inner loop adaptation does not significantly affect these layers. This suggests a natural simplification of the MAML algorithm: the *ANIL (Almost No Inner Loop) algorithm*.

In ANIL, during training **and** testing, we remove the inner loop updates for the network body, and apply inner loop adaptation only to the head. The head requires the inner loop to allow it to align to the different classes in each task. Figure 3 presents the difference between MAML and ANIL, and Appendix E considers a simple example of the gradient update in ANIL. Note that this is distinct to the freezing experiments, where we only removed the inner loop at inference time. We posit that because inner loop updates have little effect on the network body over the course of training also (Figure 2), we can remove them entirely. [31] proposed a related algorithm where there are inner/outer loop specific parameters; however, inner loop parameters are captured as additional input 'context vector(s)' to hidden layer(s) of the network, contrasting with the framing in ANIL.

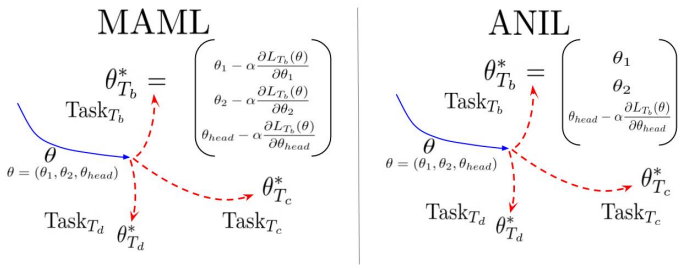

Figure 3: **Schematic of MAML and ANIL algorithms.** The difference between the MAML and ANIL algorithms: in MAML (left), the inner loop (task-specific) gradient updates are applied to all parameters θ, which are initialized with the meta-initialization from the outer loop. In ANIL (right), only the parameters corresponding to the network head θ_{head} are updated by the inner loop.

Method	Omniglot-20way-1shot	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
MAML	93.7 ± 0.7	46.9 ± 0.2	63.1 ± 0.4
ANIL	96.2 ± 0.5	46.7 ± 0.4	61.5 ± 0.5
	HalfCheetah-Direction	HalfCheetah-Velocity	2D-Navigation
MAML	170.4 ± 21.0	-139.0 ± 18.9	-20.3 ± 3.2
ANIL	363.2 ± 14.8	-120.9 ± 6.3	-20.1 ± 2.3
	Omniglot 20way-1shot	MiniImageNet 5way-1shot	MiniImageNet 5way-5shot
NIL	96.7 ± 0.3	48.0 ± 0.7	62.2 ± 0.5

Table 1: **Top: ANIL matches the performance of MAML on few shot image classification and RL.** On three standard benchmarks for few-shot image classification and RL benchmarks seen in [4], considering accuracy for classification and average return for RL, ANIL performs comparably to MAML (higher is better) **Bottom: NIL algorithm performs as well as/better than MAML and ANIL on test-time few-shot image classification.** Accuracy of NIL on test-time few shot image classification benchmarks. With no test-time inner loop, and just learned features, NIL performs comparably to than MAML and ANIL, indicating the strength of the learned features, and the relative lack of importance of the head at test time.

Results of ANIL on Standard Benchmarks: We evaluate ANIL on few-shot image classification and RL benchmarks, using the same model architectures as the original MAML authors, for both supervised learning and RL. Further implementation details are in Appendix E.5. The results in Table 1, top section (mean and standard deviation of performance over three random initializations) show that ANIL matches the performance of MAML on both few shot classification and RL, demonstrating that the inner loop adaptation is almost unnecessary for learning good features. In Appendix E.4, we further compare MAML and ANIL, and find that both algorithms have similar representations and learning curves, illustrating that the inner loop (for the body) doesn't impact feature learning.

4 Removing the Head: The NIL (No Inner Loop) Algorithm

So far, we have seen that MAML mainly relies on feature reuse and that the inner loop is not necessary at training time for learning good features in the network body. The ANIL algorithm keeps inner loop adaptation only for the network head to enable task specific alignment which helps with learning. A next question is whether the head (and task specific alignment) is necessary when good features have *already* been learned. To study this, we propose the *No Inner Loop (NIL) algorithm*:

- Train a few-shot learning model with ANIL/MAML algorithm as standard. We use ANIL training.
- At test time, take the trained model, and remove the head. For each task, first pass the k labelled examples (support set) through the body of the network, to get their penultimate layer representations. Then, for a test example, compute cosine similarities between its penultimate layer representation and those of the support set, using these to weight the support set labels [30].

The results for the NIL algorithm, following ANIL training, on few shot classification benchmarks are given in Table 1, bottom. Despite not using the network head at all, NIL performs comparably to MAML and ANIL. This demonstrates that the features learned by the network body when training with MAML/ANIL (and reused at test time) is *the critical component* in effectively tackling these benchmarks. The head's contribution is mostly to help facilitate this feature learning through task specific alignment. Further analysis of this phenomenon is presented in Appendix F.

5 Conclusion

In this paper, we studied whether the MAML algorithm relies on rapid learning or feature reuse. We found that *feature reuse* is paramount to MAML's efficacy. This insight led to the ANIL (Almost No Inner Loop) algorithm, a simplification of MAML that has identical performance on standard image classification and RL benchmarks. We further study the importance of the head (final layer) of a neural network trained with MAML, discovering that the body (lower layers) of a network is sufficient for few-shot classification at test time, allowing us to completely remove the network head for testing (NIL) and still match performance. Our study mainly considered the Omniglot and MiniImageNet datasets; it is an interesting future direction to consider rapid learning and feature reuse in MAML on other few-shot learning tasks, such as those from [28].

References

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

[2] PMP Art. Overview figure: Lion image. `https://www.pinterest.com/pin/350436414739113168/`. Accessed: 2019-09-09.

[3] Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. Identifying and controlling important neurons in neural machine translation. *arXiv preprint arXiv:1811.01157*, 2018.

[4] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[5] Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *arXiv preprint arXiv:1710.11622*, 2017.

[6] Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pages 9516–9527, 2018.

[7] Nathan Glover. Overview figure: Cat image. `https://thesecatsdonotexist.com/`. Accessed: 2019-09-09.

[8] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *arXiv preprint arXiv:1810.13243*, 2018.

[9] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

[10] GuideDogVerity. Overview figure: Dog image. `https://twitter.com/guidedogverity`. Accessed: 2019-09-09.

[11] Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. *arXiv preprint arXiv:1810.02334*, 2018.

[12] Khurram Javed and Martha White. Meta-learning representations for continual learning. *arXiv preprint arXiv:1905.12588*, 2019.

[13] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, 2015.

[14] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

[15] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? *arXiv preprint arXiv:1805.08974*, 2018.

[16] Niru Maheswaranathan, Alex H. Willams, Matthew D. Golub, Surya Ganguli, and David Sussillo. Universality and individuality in neural dynamics across large populations of recurrent networks. *arXiv preprint arXiv:1907.08549*, 2019.

[17] Ahmed Malik. Overview figure: Plane image. `https://appadvice.com/app/real-airplane-pilot-flight-simulator-game-for-free/1186146488`. Accessed: 2019-09-09.

[18] Ari S Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. *arXiv preprint arXiv:1806.05759*, 2018.

[19] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2, 2018.

[20] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, pages 6076–6085, 2017.

[21] Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning with applications to medical imaging. *arXiv preprint arXiv:1902.07208*, 2019.

[22] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[23] 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.

[24] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016.

[25] Naomi Saphra and Adam Lopez. Understanding learning dynamics of language models with svcca. *arXiv preprint arXiv:1811.00225*, 2018.

[26] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

[27] Save Sheffield Trees. Overview figure: Tree image. `https://twitter.com/saveshefftrees`. Accessed: 2019-09-09.

[28] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[29] Vexels. Overview figure: Chair image. `https://www.vexels.com/png-svg/preview/148959/small-office-chair-clipart`. Accessed: 2019-09-09.

[30] 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.

[31] Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Caml: Fast context adaptation via meta-learning. *arXiv preprint arXiv:1810.03642*, 2018.

A MAML details

The MAML algorithm finds an *initialization* for a neural network so that new tasks can be learnt with very few examples (k examples from each class for k-shot learning) via two optimization loops:

- **Outer Loop:** Updates the initialization of the neural network parameters (often called the *meta-initialization*) to a setting that enables fast adaptation to new tasks.
- **Inner Loop:** Performs *adaptation*: takes the outer loop initialization, and, *separately for each task*, performs a few gradient updates over the k labelled examples (the *support set*) provided for adaptation.

More formally, we first define our base model to be neural network with meta-initialization parameters θ; let this be represented by f_θ. We have have a distribution $\mathcal{D}$ over tasks, and draw a batch $\{T_1, ..., T_B\}$ of B tasks from $\mathcal{D}$. For each task T_b, we have a *support set* of examples $\mathcal{S}_{T_b}$, which are used for inner loop updates, and a *target set* of examples $\mathcal{Z}_{T_b}$, which are used for outer loop updates. Let $\theta_i^{(b)}$ signify θ after i gradient updates for task T_b, and let $\theta_0^{(b)} = \theta$. In the inner loop, during each update, we compute

$$\theta_m^{(b)} = \theta_{m-1}^{(b)} - \alpha \nabla_{\theta_{m-1}^{(b)}} \mathcal{L}_{S_{T_b}}(f_{\theta_{m-1}^{(b)}(\theta)}) \tag{1}$$

for m fixed across all tasks, where $\mathcal{L}_{S_{T_b}}(f_{\theta_{m-1}^{(b)}(\theta)})$ is the loss on the support set of T_b after $m-1$ inner loop updates.

We then define the meta loss as

$$\mathcal{L}_{meta}(\theta) = \sum_{b=1}^{B} \mathcal{L}_{\mathcal{Z}_{T_b}}(f_{\theta_m^{(b)}(\theta)})$$

where $\mathcal{L}_{Z_{T_b}}(f_{\theta_m^{(b)}(\theta)})$ is the loss on the target set of T_b after m inner loop updates, making clear the dependence of $f_{\theta_m^{(b)}}$ on θ. The outer optimization loop then updates θ as

$$\theta = \theta - \eta \nabla_\theta \mathcal{L}_{meta}(\theta)$$

At test time, we draw unseen tasks $\{T_1^{(test)}, ..., T_n^{(test)}\}$ from the task distribution, and evaluate the loss and accuracy on $\mathcal{Z}_{T_i^{(test)}}$ after inner loop adaptation using $\mathcal{S}_{T_i^{(test)}}$ (e.g. loss is $\mathcal{L}_{\mathcal{Z}_{T_i^{(test)}}}\left(f_{\theta_m^{(i)}(\theta)}\right)$).

B Experimental Setup for Freezing and CCA studies

For freezing and CCA studies, we concentrate on the MiniImageNet dataset, a popular benchmark for few shot learning that is more complex than MNIST or Omniglot. The dataset is described in Section C.

The model architecture we use for our experiments is identical to that used in the original MAML paper: 4 modules with a 3 x 3 convolutions and 32 filters, followed by batch normalization, ReLU nonlinearity, and 2 x 2 max pooling. Our models are trained using a batch size of 4, 5 inner loop update steps, and an inner learning rate of 0.01. 10 inner gradient steps are used for evaluation at test time. We train models 3 times with different random seeds. Models were trained for 30000 iterations.

C Few-Shot Image Classification Datasets and Experimental Setups

We consider the few-shot learning paradigm for image classification to evaluate MAML and ANIL. We evaluate using two datasets often used for few-shot multiclass classification – the Omniglot dataset and the MiniImageNet dataset.

Omniglot: The Omniglot dataset consists of over 1600 different handwritten character classes from 23 alphabets. The dataset is split on a character-level, so that certain characters are in the training set, and others in the validation set. We consider the 20-way 1-shot task on this dataset, where at test

Freeze layers	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
None	44.5 ± 0.8	61.7 ± 0.6
1	44.8 ± 0.7	61.7 ± 0.7
1,2	44.8 ± 0.8	61.4 ± 0.7
1,2,3	44.7 ± 0.8	60.2 ± 0.7
1,2,3,4	44.7 ± 0.8	60.2 ± 0.7

Table 2: **Freezing successive layers (preventing inner loop adaptation) does not significantly impact accuracy.** To test the amount of feature reuse happening in the inner loop adaptation, we test the accuracy of the model when we freeze (prevent inner loop adaptation) a contiguous block of layers. We find that freezing even all four convolutional layers of the network (all layers except the network head) hardly affects test-time accuracy. This strongly supports the feature reuse hypothesis: layers don't have to change rapidly at adaptation time; they already contain good features from the meta-initialization.

time, we wish our classifier to discriminate between 20 randomly chosen character classes from the validation set, given only one labelled example from each class from this set of 20 testing classes. The model architecture used is identical to that in the original MAML paper, namely: 4 modules with a 3 x 3 convolutions and 64 filters with a stride of 2, followed by batch normalization, and a ReLU nonlinearity. The Omniglot images are downsampled to 28 x 28, so the dimensionality of the last hidden layer is 64. The last layer is fed into a 20-way softmax. Our models are trained using a batch size of 16, 5 inner loop updates, and an inner learning rate of 0.1.

MiniImageNet: The MiniImagenet dataset was proposed by [22], and consists of 64 training classes, 12 validation classes, and 24 test classes. We consider the 5-way 1-shot and 5-way 5-shot tasks on this dataset, where the test-time task is to classify among 5 different randomly chosen validation classes, given only 1 and 5 labelled examples respectively. The model architecture is again identical to that in the original paper: 4 modules with a 3 x 3 convolutions and 32 filters, followed by batch normalization, ReLU nonlinearity, and 2 x 2 max pooling. Our models are trained using a batch size of 4. 5 inner loop update steps, and an inner learning rate of 0.01 are used. 10 inner gradient steps are used for evaluation at test time.

D Additional Freezing and Representational Similarity Results

In this section, we provide more details and further results from freezing and representational similarity experiments.

D.1 Freezing Layer Representations

To study the impact of the inner loop adaptation, we freeze a contiguous subset of layers of the network, excluding the head (final layer), during the inner loop. These frozen layers are not updated at test time in the inner loop, and must reuse the features learned by the meta-initialization from the outer loop. We compare the few shot learning accuracy when freezing to the accuracy when allowing inner loop adaptation.

Results are shown in Table 2. We observe that even when freezing all layers but the head, performance hardly changes. This demonstrates that the meta-initialization has already learned good enough features that can be reused without any adaptation across the different tasks.

D.2 Representational Similarity Experiments

We next study how much the latent representations (the latent functions) learned by the neural network change during the inner loop adaptation phase. Following several recent works [20, 25, 18, 16, 21, 8, 3] we measure this by applying Canonical Correlation Analysis (CCA) to the latent representations of the network. CCA takes in as inputs $L_1 = \{z_1^{(1)}, z_2^{(1)}, ...z_m^{(1)}\}$ and $L_2 = \{z_1^{(2)}, z_2^{(1)}, ...z_n^{(2)}\}$, where L_1, L_2 are layers, and $z_i^{(j)}$ is a neuron activation vector: the vector of outputs of neuron i (of layer L_j) over a set of inputs X. It then finds linear combinations of the neurons in L_1 and neurons in L_2 so that the resulting activation vectors are maximally correlated, which is summarized in the canonical correlation coefficient. Iteratively repeating this process gives a similarity score (in $[0, 1]$

with 1 identical and 0 completely different) between the representations of L_1 and L_2. For full details, see [20, 18].

In our analysis, we take L_1 to be a layer before the inner loop adaptation steps, and L_2 after the inner loop adaptation steps. We compute CCA similarity between L_1, L_2, averaging the similarity score across different random seeds of the model and different test time tasks.

The result is shown in Figure 4, left pane. We see that the representations in the conv layers of the network are highly similar to each other pre and post adaptation. Conv1 and conv2 have a CCA similarity of almost 1 which indicates that these layers barely change. Conv3 and conv4 show slightly less similarity (due to small representational differences in conv1, conv2 having a compounding effect in the deeper layers) but similarities are still greater than 0.9 (very high). By contrast, the head (final layer), which does change significantly during the inner loop, has a CCA similarity of less than 0.5.

In Figure 4 right pane, we also compute CKA (Centered Kernel Alignment) [14] another similarity metric for neural network representations, which highlights the same high level conclusion (layers in the network body highly similar before and after adaptation.) Overall, the representational analysis results also strongly support the feature reuse hypothesis.

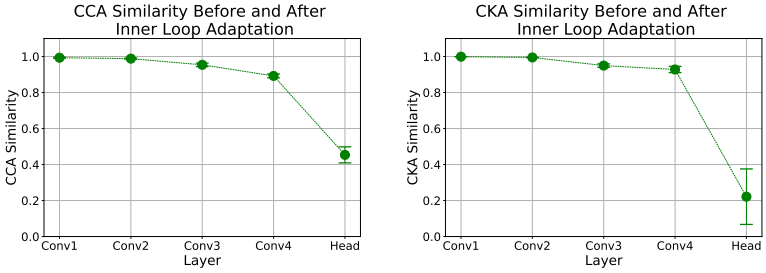

Figure 4: **High CCA/CKA similarity between representations before and after adaptation for all layers except the head.** We compute CCA/CKA similarity between the representation of a layer before the inner loop adaptation and after adaptation. We observe that for all layers except the head, the CCA/CKA similarity is almost 1, indicating perfect similarity. This suggests that these layers do not change much during adaptation, but mostly perform feature reuse. Note that there is a slight dip in similarity in the higher conv layers (e.g. conv3, conv4); this is likely because the slight representational differences in conv1, conv2 have a compounding effect on the representations of conv3, conv4. The head of the network must change significantly during adaptation, and this is reflected in the much lower CCA/CKA similarity.

D.3 Similarity Before and After Inner Loop with Euclidean Distance

In addition to assessing representational similarity with CCA, we also consider the simpler measure of Euclidean distance, capturing how much weights of the network change during the inner loop update (task-specific finetuning). We note that this experiment does not assess functional changes on inner loop updates as well as the CCA experiments do; however, they serve to provide useful intuition.

We see that most layer parameters do not change significantly during finetuning. We plot the per-layer average Euclidean distance between the initialization θ and the finetuned weights $\theta_m^{(b)}$ across different tasks T_b, i.e.

$$\frac{1}{N}\sum_{b=1}^{N} ||(\theta_l) - (\theta_l)_m^{(b)}||$$

across different layers l, for MiniImageNet in Figure 5. We observe that very quickly after the start of training, all layers except for the last layer have small Euclidean distance difference before and after finetuning, suggesting significant feature reuse. (Note that this is despite the fact that these layers have more parameters than the final layer.)

D.4 MiniImageNet-5way-1shot Freezing and CCA Over Training

Figure 6 shows that from early on in training, on MiniImageNet-5way-1shot, that the CCA similarity between activations pre and post inner loop update is very high for all layers but the head. We further

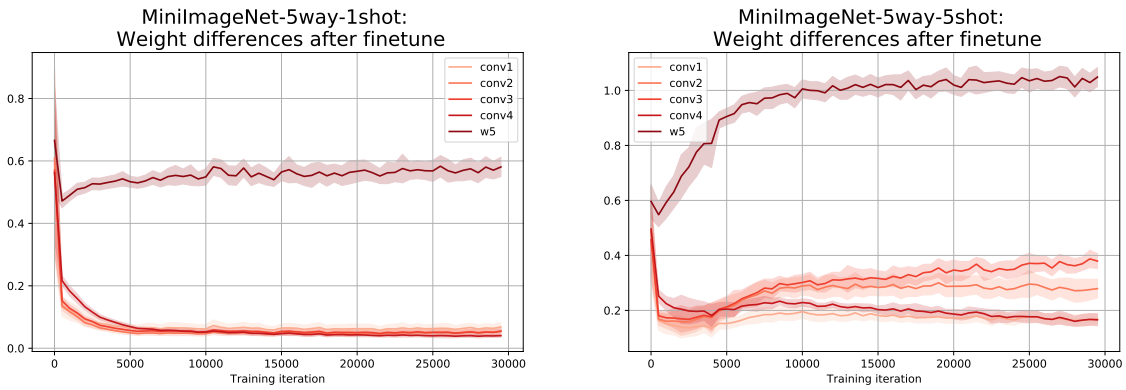

Figure 5: **Euclidean distance before and after finetuning for MiniImageNet**. We compute the average (across tasks) Euclidean distance between the weights before and after inner loop adaptation, separately for different layers. We observe that all layers except for the final layer show very little difference before and after inner loop adaptation (despite having more parameters), suggesting significant feature reuse.

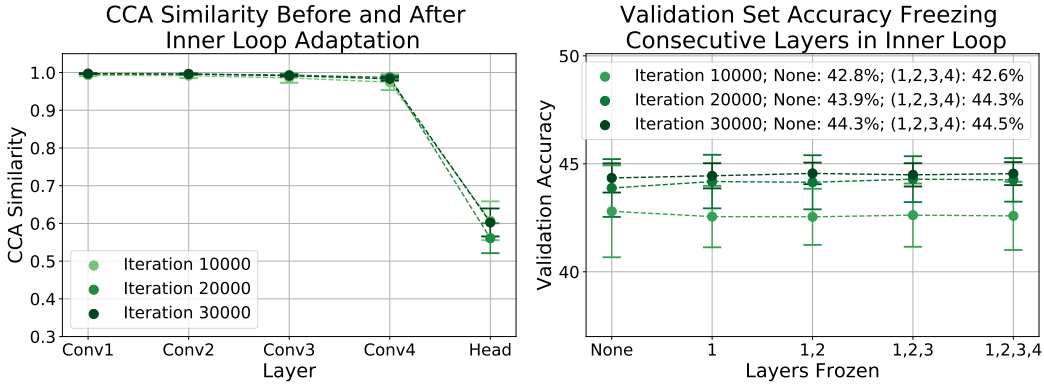

Figure 6: **Inner loop updates have little effect on learned representations from early on in learning.** We consider freezing and representational similarity experiments for MiniImageNet-5way-1shot. We see that early on in training (from as few as 10k iterations in), the inner loop updates have little effect on the learned representations and features, and that removing the inner loop updates for all layers but the head have little-to-no impact on the validation set accuracy.

see that the validation set accuracy suffers almost no decrease if we remove the inner loop updates and freeze all layers but the head. This shows that even early on in training, the inner loop appears to have minimal effect on learned representations and features. This supplements the results seen in Figure 2 on MiniImageNet-5way-5shot.

E ANIL Algorithm: More Details

In this section, we provide more details about the ANIL algorithm, including an example of the ANIL update, implementation details, and further experimental results.

E.1 ANIL Algorithm Mathematical Formulation

For the ANIL algorithm, mathematically, let $\theta = (\theta_1, ..., \theta_l)$ be the (meta) initialization parameters for the l layers of the network. Following the notation of Section A, let $\theta_m^{(b)}$ be the parameters after m inner gradient updates for task $\mathcal{T}_b$. In ANIL, we have that:

$$\theta_m^{(b)} = \left(\theta_1, \ldots, (\theta_l)_{m-1}^{(b)} - \alpha \nabla_{(\theta_l)_{m-1}^{(b)}} \mathcal{L}_{T_b}(\theta_{m-1}^{(b)}, \mathcal{S}_{T_b})\right)$$

i.e. only the final layer gets the inner loop updates. As before, we then define the meta-loss, and compute the outer loop gradient update.

E.2 An Example of the ANIL Update

Consider a simple, two layer linear network with a single hidden unit in each layer: $\hat{y}(x; \boldsymbol{\theta}) = \theta_2(\theta_1 x)$. In this example, θ_2 is the *head*. Consider the 1-shot regression problem, where we have access to examples $\left\{(x_1^{(t)}, y_1^{(t)}), (x_2^{(t)}, y_2^{(t)})\right\}$ for tasks $t = 1, \ldots, T$. Note that $(x_1^{(t)}, y_1^{(t)})$ is the (example, label) pair in the meta-training set (used for inner loop adaptation), and $(x_2^{(t)}, y_2^{(t)})$ is the pair in the meta-validation set (used for the outer loop update).

In the few-shot learning setting, we firstly draw a set of N tasks and labelled examples from our meta-training set: $\left\{(x_1^{(1)}, y_1^{(1)}), \ldots, (x_1^{(N)}, y_1^{(N)})\right\}$. The inner loop updates for each task are defined as follows:

$$\theta_1^{(t)} \leftarrow \theta_1 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_1} \tag{1}$$

$$\theta_2^{(t)} \leftarrow \theta_2 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_2} \tag{2}$$

where $L(\cdot, \cdot)$ is the loss function, (e.g. mean squared error) and $\theta_i^{(t)}$ refers to a parameter after inner loop update for task t.

The task-adapted parameters for MAML and ANIL are as follows. Note how only the head parameters change per-task in ANIL:

$$\boldsymbol{\theta}_{\text{MAML}}^{(t)} = \left[\theta_1^{(t)}, \theta_2^{(t)}\right] \tag{3}$$

$$\boldsymbol{\theta}_{\text{ANIL}}^{(t)} = \left[\theta_1, \theta_2^{(t)}\right] \tag{4}$$

In the outer loop update, we then perform the following operations using the data from the meta-validation set:

$$\theta_1 \leftarrow \theta_1 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x_2^{(t)}; \boldsymbol{\theta}^{(t)}), y_2^{(t)})}{\partial \theta_1} \tag{5}$$

$$\theta_2 \leftarrow \theta_2 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x_2^{(t)}; \boldsymbol{\theta}^{(t)}), y_2^{(t)})}{\partial \theta_2} \tag{6}$$

Considering the update for θ_1 in more detail for our simple, two layer, linear network (the case for θ_2 is analogous), we have the following update for MAML:

$$\theta_1 \leftarrow \theta_1 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x_2^{(t)}; \boldsymbol{\theta}_{\text{MAML}}^{(t)}), y_2^{(t)})}{\partial \theta_1} \tag{7}$$

$$\hat{y}(x_2^{(t)}; \boldsymbol{\theta}_{\text{MAML}}^{(t)}) = \left(\left[\theta_2 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_2}\right] \cdot \left[\theta_1 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_1}\right] \cdot x_2\right) \tag{8}$$

For ANIL, on the other hand, the update will be:

$$\theta_1 \leftarrow \theta_1 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x_2^{(t)}; \boldsymbol{\theta}_{\text{ANIL}}^{(t)}), y_2^{(t)})}{\partial \theta_1} \tag{9}$$

$$\hat{y}(x_2^{(t)}; \boldsymbol{\theta}_{\text{ANIL}}^{(t)}) = \left(\left[\theta_2 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_2}\right] \cdot \theta_1 \cdot x_2\right) \tag{10}$$

Note the lack of inner loop update for θ_1, and how we do not remove second order terms in ANIL (unlike in first-order MAML); second order terms still persist through the derivative of the inner loop update for the head parameters.

E.3 ANIL Learns Almost Identically to MAML

MAML and ANIL perform equally well on few-shot learning benchmarks, illustrating that removing the inner loop during training does not hinder performance. To study the similarity between MAML and ANIL models further, we consider learning curves for both algorithms in Figure 7. We find that learning proceeds almost identically for ANIL and MAML. This helps validate the hypothesis that feature learning happens primarily in the outer loop of MAML, with little adaptation happening in the inner loop.

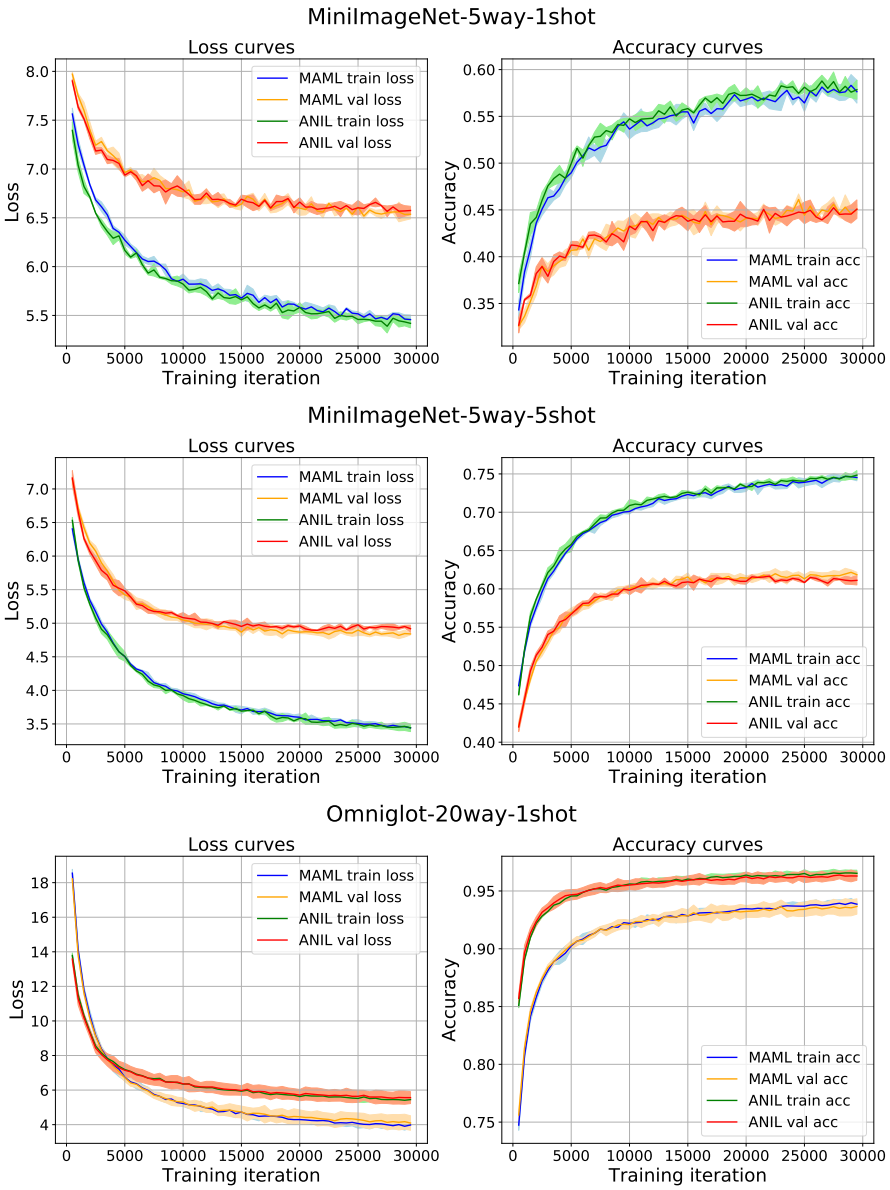

Figure 7: **ANIL and MAML on MiniImageNet and Omniglot**. Loss and accuracy curves for ANIL and MAML on (i) MiniImageNet-5way-1shot (ii) MiniImageNet-5way-5shot (iii) Omniglot-20way-1shot.

Model Pair	CCA Similarity	CKA Similarity
MAML-MAML	0.51	0.83
ANIL-ANIL	0.51	0.86
ANIL-MAML	0.50	0.83

Table 3: **MAML and ANIL models have clear representational correspondence.** Representational similarities of MAML and ANIL after training show that MAML models are as similar to each other as they are to ANIL models.

E.4 ANIL and MAML Learn Similar Representations

Table 3 presents CCA and CKA (another representational similarity measure, from [15]) similarities for MAML and ANIL models. We see that MAML-ANIL similarity is almost equal to that of two MAML models or two ANIL models. ANIL and MAML models are clearly highly similar, even through training, and both algorithms can evidently be used to discover good features for few-shot learning.

E.5 ANIL Implementation Details

Supervised Learning Implementation: We used the TensorFlow MAML implementation open-sourced by the original authors [4]. We used the same model architectures as in the original MAML paper for our experiments, and train models 3 times with different random seeds. All models were trained for 30000 iterations, with a batch size of 4, 5 inner loop update steps, and an inner learning rate of 0.01. 10 inner gradient steps were used for evaluation at test time.

Reinforcement Learning Implementation: We used the open source PyTorch implementation of MAML for RL [2], due to challenges encountered when running the open-sourced TensorFlow implementation from the original authors. We note that the results for MAML in these RL domains do not exactly match those in the original paper; this may be due to large variance in results, depending on the random initialization. We used the same model architecture as the original paper (two layer MLP with 100 hidden units in each layer), a batch size of 40, 1 inner loop update step with an inner learning rate of 0.1 and 20 trajectories for inner loop adaptation. We trained three MAML and ANIL models with different random initialization, and quote the mean and standard deviation of the results. As in the original MAML paper, for RL experiments, we select the best performing model over 500 iterations of training and evaluate this model at test time on a new set of tasks.

E.6 ANIL is Computationally Simpler Than MAML

Table 4 shows results from a comparison of the computation time for MAML, First Order MAML, and ANIL, during training and inference, with the TensorFlow implementation described previously, on both MiniImageNet domains. These results are average time for executing forward and backward passes during training (above) and a forward pass during inference (bottom), for a task batch size of 1, and a target set size of 1. Results are averaged over 2000 such batches. Speedup is calculated relative to MAML's execution time. Each batches' images were loaded into memory before running the TensorFlow computation graph, to ensure that data loading time was not captured in the timing. Experiments were run on a single NVIDIA Titan-Xp GPU.

During training, we see that ANIL is as fast as First Order MAML (which does not compute second order terms during training), and about 1.7x as fast as MAML. This leads to a significant overall training speedup, especially when coupled with the fact that the rate of learning for these ANIL and MAML is very similar; see learning curves in Appendix E.3. Note that unlike First Order MAML, ANIL also performs very comparably to MAML on benchmark tasks (on some tasks, First Order MAML performs worse [4]). During inference, ANIL achieves over a 4x speedup over MAML (and thus also 4x over First Order MAML, which is identical to MAML at inference time). Both training and inference speedups illustrate the significant computational benefit of ANIL over MAML.

[2] `https://github.com/tristandeleu/pytorch-maml-rl`

	Training: 5way-1shot			Training: 5way-5shot		
	Mean (s)	Median (s)	Speedup	Mean (s)	Median (s)	Speedup
MAML	0.15	0.13	1	0.68	0.67	1
First Order MAML	0.089	0.083	1.69	0.40	0.39	1.7
ANIL	0.084	0.072	1.79	0.37	0.36	1.84

	Inference: 5way-1shot			Inference: 5way-5shot		
	Mean (s)	Median (s)	Speedup	Mean (s)	Median (s)	Speedup
MAML	0.083	0.078	1	0.37	0.36	1
ANIL	0.020	0.017	4.15	0.076	0.071	4.87

Table 4: **ANIL offers significant computational speedup over MAML, during both training and inference.** Table comparing execution times and speedups of MAML, First Order MAML, and ANIL during training (above) and inference (below) on MiniImageNet domains. Speedup is calculated relative to MAML's execution time. We see that ANIL offers noticeable speedup over MAML, as a result of removing the inner loop almost completely. This permits faster training and inference.

F Analyzing the Network Head and Body

The results for the NIL algorithm, following ANIL training, on few-shot classification benchmarks are given in Table 1. Despite not using the network head at all, NIL performs comparably to MAML and ANIL. This demonstrates that the features learned by the network body when training with MAML/ANIL (and reused at test time) is the critical component in effectively tackling these benchmarks. The head's contribution is mostly to help facilitate this feature learning through task specific alignment.

The results of the NIL algorithm suggest another natural question – is the head, and the resulting task-specific alignment, really necessary during training time to learn good features? More broadly speaking, we ask: how do different training regimes for the body affect the quality of features learned? So far we have studied the features learned through MAML and ANIL training regimes, but other alternatives, e.g. multiclass classification, or multitask learning are also possibilities. Performance of the latter, which has no task-specific head, directly answers the initial question – the effect of the head during training time.

We investigate the effectiveness of these different training regimes as follows:

- We train MAML/ANIL networks as standard, and do standard test time adaptation.
- For multiclass training, we first (pre)train with multiclass classification, then throw away the head and freeze the body. We initialize a new e.g. 5-class head, and train that (on top of the frozen multiclass pretrained features) with MAML. At test time we perform standard adaptation.
- The same process is applied to multitask training.
- A similar process is applied to random features, except the network is initialized and then frozen.

The results are shown in Table 5. We find that MAML and ANIL result in the network body learning the best features (evidenced by their performance on the benchmarks) followed by multiclass classification. Multitask training performs significantly worse, due to the alignment problem – the lack of a head that can correctly align the network to each task. Interestingly, multitask training performs *worse* than using random features.

We also observe very little performance difference between using a MAML/ANIL head and a NIL head for each training regime. Specifically, task performance is purely determined by the quality of the features and representations learned during training, with task-specific alignment at test time being (i) unnecessary (ii) unable to influence the final performance of the model (e.g. multitask training performance is equally with a MAML head as it is with a NIL-head.)

F.0.1 Analyzing the Representations

To gain further insight on the features learned by the different training regimes, we apply CCA [20] and CKA [14] to representations learned by the body of the network. The results, shown in

Method	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
MAML training-MAML head	46.9 ± 0.2	63.1 ± 0.4
MAML training-NIL head	48.4 ± 0.3	61.5 ± 0.8
ANIL training-ANIL head	46.7 ± 0.4	61.5 ± 0.5
ANIL training-NIL head	48.0 ± 0.7	62.2 ± 0.5
Multiclass pretrain-MAML head	38.4 ± 0.8	54.6 ± 0.4
Multiclass pretrain-NIL head	39.7 ± 0.3	54.4 ± 0.5
Multitask pretrain-MAML head	26.5 ± 0.8	32.8 ± 0.6
Multitask pretrain-NIL head	26.5 ± 1.1	34.2 ± 3.5
Random features-MAML head	32.1 ± 0.5	43.1 ± 0.3
Random features-NIL head	32.9 ± 0.6	43.2 ± 0.5

Table 5: **Test time performance is dominated by features learned, with no difference between NIL/MAML heads.** We see identical performances of MAML/NIL heads at test time, indicating that MAML/ANIL training leads to better learned features.

Table 6, show a clear correspondence to the performance results – training regimes with features and representations more similar to MAML also perform better. Specifically, across both similarity measures, multiclass has representations most similar to MAML representations, followed by random features and then multitask, mapping exactly to the performance ordering of the different methods.

Feature pair	CCA Similarity	CKA Similarity
(MAML, MAML)	0.51	0.83
(Multiclass pretrain, MAML)	0.48	0.79
(Random features, MAML)	0.40	0.72
(Multitask pretrain, MAML)	0.28	0.65

Table 6: **MAML training most closely resembles multiclass pretraining, as illustrated by CCA and CKA similarities**. On analyzing the CCA and CKA similarities between different baseline models and MAML (comparing across different tasks and seeds), we see that multiclass pretraining results in features most similar to MAML training. Multitask pretraining differs quite significantly from MAML-learned features, potentially due to the alignment problem.