A Example of a Self-Attention Mechanism for Supervised Learning

Self-attention in its rudimentary form can implement a cosine-similarity-based sample weighting, which can also be viewed as a simple 1-step MAML-like learning algorithm. This can be seen by considering a simple classification model

$$f(x;\theta) = s(\boldsymbol{W} \cdot \boldsymbol{e}(x;\phi) + \boldsymbol{b})$$

with $\theta = (\boldsymbol{W}, \boldsymbol{b}, \phi)$, where $\boldsymbol{e}(x;\phi)$ is the embedding and $s(\cdot)$ is a softmax function. MAML algorithm identifies such initial weights θ_0 that given any task T just a few gradient descent steps with respect to the loss $\mathcal{L}_T$ starting at θ_0 bring the model towards a task-specific local optimum of $\mathcal{L}_T$.

Notice that if any label assignment in the training tasks is equally likely, it is natural for $f(x;\theta_0)$ to not prefer any particular label over the others. Guided by this, let us choose $\boldsymbol{W}_0$ and $\boldsymbol{b}_0$ that are *label-independent*. Substituting $\theta = \theta_0 + \delta\theta$ into $f(x;\theta)$, we obtain

$$f_\ell(x;\theta) = f_\ell(x;\theta_0) + s'_\ell(\cdot)(\delta\boldsymbol{W}_\ell \cdot \boldsymbol{e}(x;\phi_0) + \delta\boldsymbol{b}_\ell + \boldsymbol{W}_{0,\ell} \cdot \boldsymbol{e}'(x;\phi_0)\delta\phi) + O(\delta\theta^2),$$

where ℓ is the label index and $\delta\theta = (\delta\boldsymbol{W}, \delta\boldsymbol{b}, \delta\phi)$. We see that the lowest-order label-dependent correction to $f_\ell(x;\theta_0)$ is given simply by $s'_\ell(\cdot)(\delta\boldsymbol{W}_\ell \cdot \boldsymbol{e}(x;\phi_0) + \delta\boldsymbol{b}_\ell)$. In other words, in the lowest-order, the model only adjusts the final logits layer to adapt the pretrained embedding $\boldsymbol{e}(x;\phi_0)$ to a new task. It is easy to calculate then that for a simple softmax cross-entropy loss, a single step of the gradient descent results in the following logits weight and bias updates:

$$\delta W_{ij} = \frac{\gamma}{n}\sum_{m=1}^{n}\left(y_i^{(m)} - \frac{1}{|C|}\right) e_j(x^{(m)};\phi_0), \qquad \delta b_i = \frac{\gamma}{n}\sum_{m=1}^{n}\left(y_i^{(m)} - \frac{1}{|C|}\right). \tag{2}$$

Here γ is the learning rate, n is the total number of support-set samples, $|C|$ is the number of classes and $\boldsymbol{y}^{(m)}$ is the one-hot label corresponding to $x^{(m)}$.

Now consider a self-attention module generating the last logits layer and acting on a sequence of processed input samples[5] $\mathcal{I}^L = \{(\,\xi(c_i)\,,\, h_{\boldsymbol{\phi}_l}(e_i)\,)\}_{i=1,\dots,n}$ and weight placeholders $\mathcal{W}^L := \{(\,\mu(k)\,,\,0\,)\}_{k=1,\dots,|C|}$, where $|C|$ is the number of classes and also the number of weight slices of $\boldsymbol{W}$ if each slice corresponds to an output layer channel. The output of a simple self-attention module for the weight slice with index i is then given by:

$$Z^{-1}\sum_{m=1}^{n} e^{\boldsymbol{Q}(\mathcal{W}_i^L)\cdot\boldsymbol{K}_I(\mathcal{I}_m^L)}\boldsymbol{V}_I(\mathcal{I}_m^L) + Z^{-1}\sum_{m=1}^{N_w} e^{\boldsymbol{Q}(\mathcal{W}_i^L)\cdot\boldsymbol{K}_W(\mathcal{W}_m^L)}\boldsymbol{V}_W(\mathcal{W}_m^L), \tag{3}$$

where $Z := \sum_{m=1}^{n} e^{\boldsymbol{Q}(\mathcal{W}_i^L)\cdot\boldsymbol{K}_I(\mathcal{I}_m^L)} + \sum_{m=1}^{N_w} e^{\boldsymbol{Q}(\mathcal{W}_i^L)\cdot\boldsymbol{K}_W(\mathcal{W}_m^L)}$. It is easy to see that with a proper choice of query and key matrices attending only to prepended ξ and μ tokens, the second term in equation 3 can be made negligible, while $\boldsymbol{Q}(\mathcal{W}_i^L)\cdot\boldsymbol{K}_I(\mathcal{I}_m^L)$ can make the softmax function only attend to those components of $\boldsymbol{V}_I$ that correspond to samples with label i. Choosing $\boldsymbol{V}_I(\mathcal{I}_m^L)$ to be proportional to $\boldsymbol{e}_m$, we can then recover the first term in $\delta\boldsymbol{W}$ in equation 2. The second term in $\delta\boldsymbol{W}$ can be produced, for example, with the help of a second head that generates identical attention weights for all samples, thus summing up their embeddings.

B Model Parameters

Here we provide additional detail of the model parameters used in our experiments.

Feature extractor parameters. For Omniglot dataset, we used the same image augmentations that were originally proposed in MAML. For miniImageNet and tieredImageNet datasets, however, we used ImageNet-style image augmentations including horizontal image flipping, random color augmentations and random image cropping. This helped us to avoid model overfitting on the miniImageNet dataset and possibly on tieredImageNet.

[5] here we use only local features $h_{\boldsymbol{\phi}_l}(e_i)$ of the sample embedding vectors e_i

The dimensionality d of the label encoding ξ and weight slice encoding μ was typically set to 32. Increasing d up to the maximum number of weight slices plus the number of per-episode labels, would allow the model to fully disentangle examples for different labels and different weight slices, but can also make the model train slower.

Transformer parameters. Since the weight tensors of each layer are generally different, our per-layer transformers were also different. The key, query and value dimensions of the transformer were chosen to be equal to a pre-defined fraction ν of the input embedding size, which in turn was a function of the label embedding and feature dimensionality as well as the size of the weight slices. The inner dimension of the final fully-connected layer in the transformer was also chosen using the same approach. In our MINIIMAGENET and TIEREDIMAGENET experiments, ν was chosen to be 0.5 and in OMNIGLOT experiments, we used $\nu = 1$. Each transformer typically contained 2 or 3 Encoder layers and used 2 or 8 heads for OMNIGLOT and MINIIMAGENET, TIEREDIMAGENET, correspondingly.

Learning schedule. In all our experiments, we used gradient descent optimizer with a learning rate in the 0.01 to 0.02 range. Our early experiments with more advanced optimizers were unstable. We used a learning rate decay schedule, in which we reduced the learning rate by a factor of 0.95 every 10^5 learning steps.

C Additional Supervised Experiments

While the advantage of decoupling parameters of the weight generator and the generated CNN model is expected to vanish with the growing CNN model size, we compared our approach to two other methods, LGM-Net (Li et al., 2019b) and LEO (Rusu et al., 2019), to verify that our approach can match their performance on sufficiently large models.

For our comparison with the LGM-Net method, we used the same image augmentation technique that was used in Li et al. (2019b) where it was applied both at the training and the evaluation stages (Ma, 2019). We also used the same CNN architecture with 4 learned 64-channel convolutional layers followed by two generated convolutional layers and the final logits layer. In our weight generator, we used 2-layer transformers with local feature extractors that relied on 48-channel convolutional layers and did not use any global features. We trained our model in the end-to-end fashion on the MINIIMAGENET 1-shot-5-way task and obtained a test accuracy of $69.3\% \pm 0.3\%$ almost identical to the 69.1% accuracy reported in Li et al. (2019b).

We also carried out a comparison with LEO by using our method to generate a fully-connected layer on top of the TIEREDIMAGENET embeddings pre-computed with a WideResNet-28 model employed by Rusu et al. (2019). For our experiments, we used a simpler 1-layer transformer model with 2 heads that did not have the final fully-connected layer and nonlinearity. We also used L_2 regularization of the generated fully-connected weights setting the regularization weight to 10^{-3}. As a result of training this model, we obtained $66.2\% \pm 0.2\%$ and $81.6\% \pm 0.2\%$ test accuracies on the 1-shot-5-way and 5-shot-5-way TIEREDIMAGENET tasks correspondingly. These results are almost identical to 66.3% and 81.4% accuracies reported in Rusu et al. (2019).

D Dependence on Parameters and Ablation Studies

Most of our parameter explorations were conducted for OMNIGLOT dataset. We chose a 16-channel model trained on a 1-shot-20-way OMNIGLOT task as an example of a model, for which just the logits layer generation was sufficient. We also chose a 4-channel model trained on a 5-shot-20-way OMNIGLOT task for the role of a model, for which generation of all convolutional layers proved to be beneficial. Figures 5 and 6 show comparison of training and test accuracies on OMNIGLOT for different parameter values for these two models. Here we only used two independent runs for each parameter value, which did not allow us to sufficiently reduce the statistical error. Despite of this, in the following, we try to highlight a few notable parameter dependencies. Note here that in some experiments with particularly large feature or model sizes, training progressed beyond the target number of steps and there could also be overfitting for very large models.

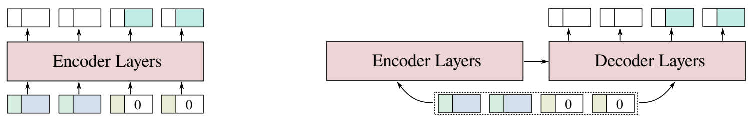

Figure 4: Two transformer-based approaches to weight generation studied in our experiments: only encoder layers (left), encoder and decoder layers on the same input sequence (right).

Number of transformer layers. Increasing the number of transformer layers is seen to be particularly important in the 4-channel model. The 16-channel model also demonstrates the benefit of using 1 vs 2 transformer layers, but the performance appears to degrade when we use 3 transformer layers.

Local feature dimension. Particularly low local feature dimension can be seen to hurt the performance in both models, while using higher local feature dimension appears to be advantageous cases except for the 32-dimensional local feature in the 4-channel model.

Embedding dimension. Particularly low embedding dimension of 16 can be seen to hurt the performance of both models.

Number of transformer heads. Increasing the number of transformer heads leads to performance degradation in the 16-channel model, but does not have a pronounced effect in the 4-channel model.

Shared feature dimensions. Removing the shared feature, or using an 8-dimensional feature can be seen to hurt the performance in both cases of the 4- and 16-channel models.

Transformer architecture. While the majority of our experiments were conducted with a sequence of transformer encoder layers, we also experimented with an alternative weight generation approach, where both encoder and decoder transformer layers were employed (see Fig. 4). Our experiments with both architectures suggest that the role of the decoder is pronounced, but very different in two models: in the 16-channel model, the presence of the decoder increases the model performance, while in the 4-channel model, it leads to accuracy degradation.

Inner transformer embedding sizes. Varying the ν parameter for different components of the transformer model (key/query pair, value and inner fully-connect layer size), we quantify their importance on the model performance. Using very low ν for the value dimension hurts performance of both models. The effect of key/query and inner dimensions can be distinctly seen only in the 4-channel model, where using $\nu = 1$ or $\nu = 1.5$ appears to produce the best results.

Weight allocation approach. Our experiments with the "spatial" weight allocation in 4- and 16-channel models showed slightly inferior performance (both accuracies dropping by about 0.2% to 0.4% in both experiments) compared to that obtained with the "output" weight allocation method.

E Attention maps of learned transformer models

We visualized the attention maps of several transformer-based models that we used for CNN layer generation. Figure 7 shows attention maps for a 2-layer 4-channel CNN network generated using a 1-head 1-layer transformer on MINIIMAGENET(labeled samples are sorted in the order of their episode labels). Attention map for the final logits layer ("CNN Layer 3") is seen to exhibit a "stairways" pattern indicating that a weight slice $W_{c,\cdot}$ for episode label c is generated by attending to all samples except for those with label c. This is reminiscent of the supervised learning mechanism outlined in Sec. 3.2. While the proposed mechanism would attend to all samples with label c and average their embeddings, another alternative is to average embeddings of samples with other labels and then invert the result. We hypothesize that the trained transformer performs a similar calculation with additional learned Transforer parameters, which may be seen to result in mild fluctuations of the attention to different input samples.

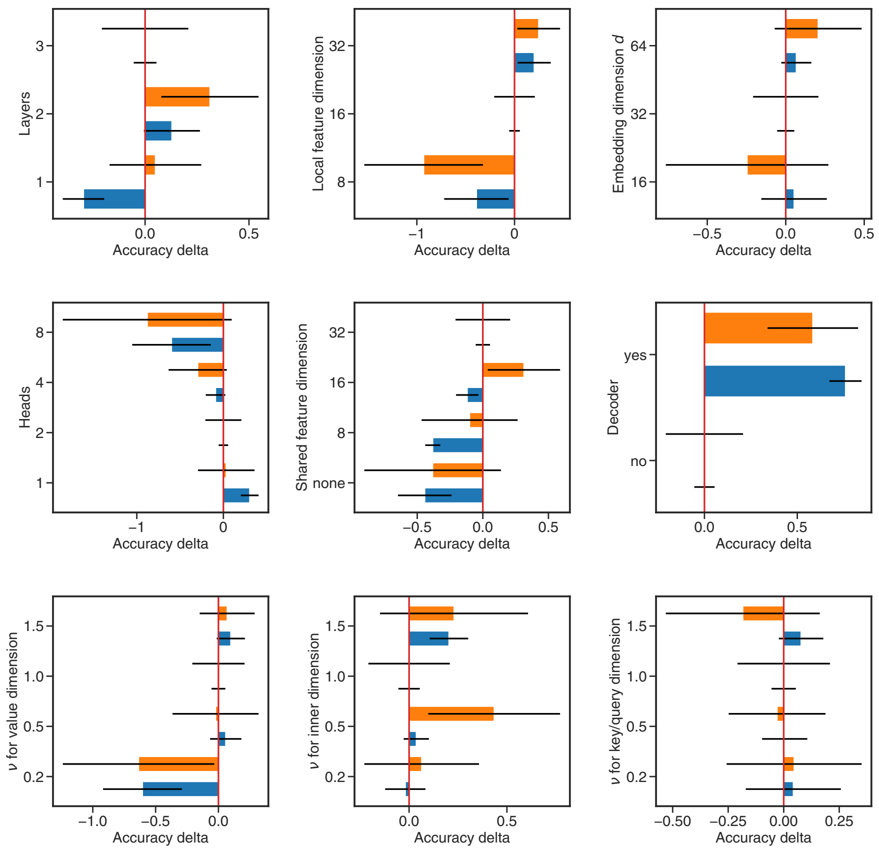

Figure 5: Change of the training (blue) and test (orange) accuracies on 1-shot-20-way OMNIGLOT task for a 16-channel model relative to the *base* configuration with 3-layer transformer, 16-dimensional local features, $\nu = 1.0$, $d = 32$, 2 heads and 32-dimensional shared features. Approximate confidence intervals are shown.

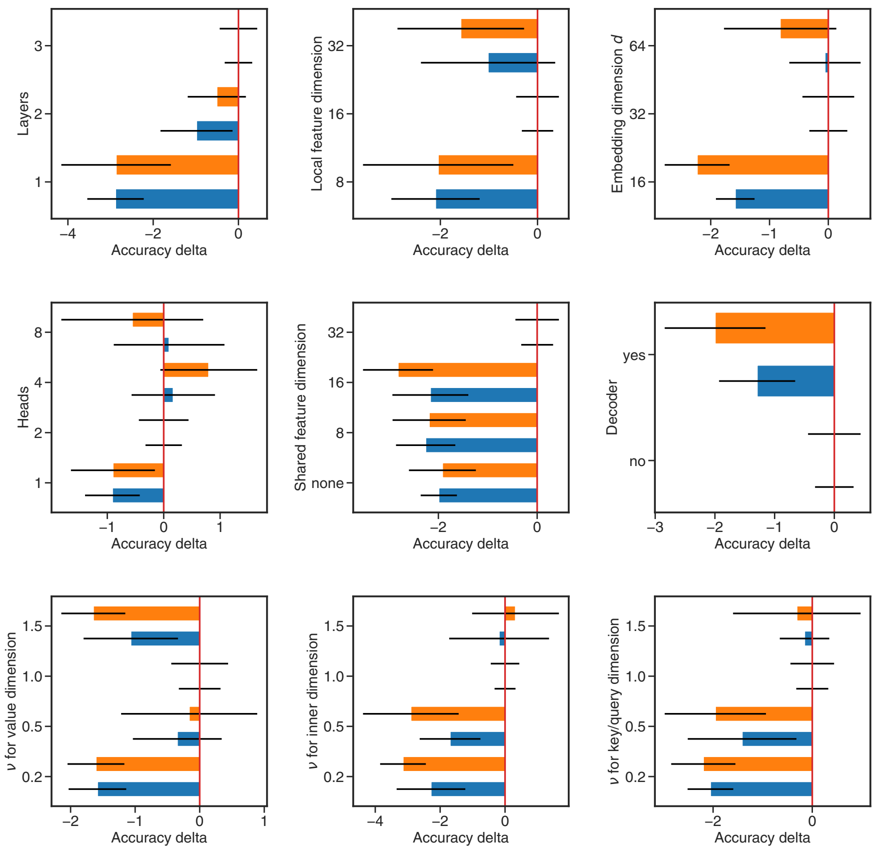

Figure 6: Change of the training (blue) and test (orange) accuracies on 5-shot-20-way OMNIGLOT task for a 4-channel model relative to the *base* configuration with 3-layer transformer, 16-dimensional local features, $\nu = 1.0$, $d = 32$, 2 heads and 32-dimensional shared features. Approximate confidence intervals are shown.

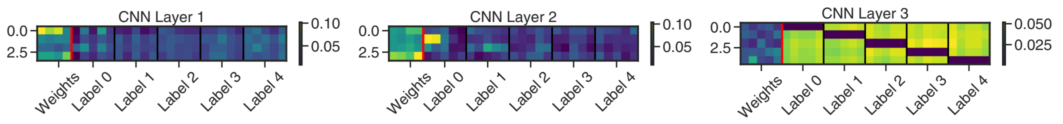

Figure 7: Learned attention weights for 2-layer 4-channel CNN network generated with 1 head, 1 layer transformer for 5-shot MINIIMAGENET.

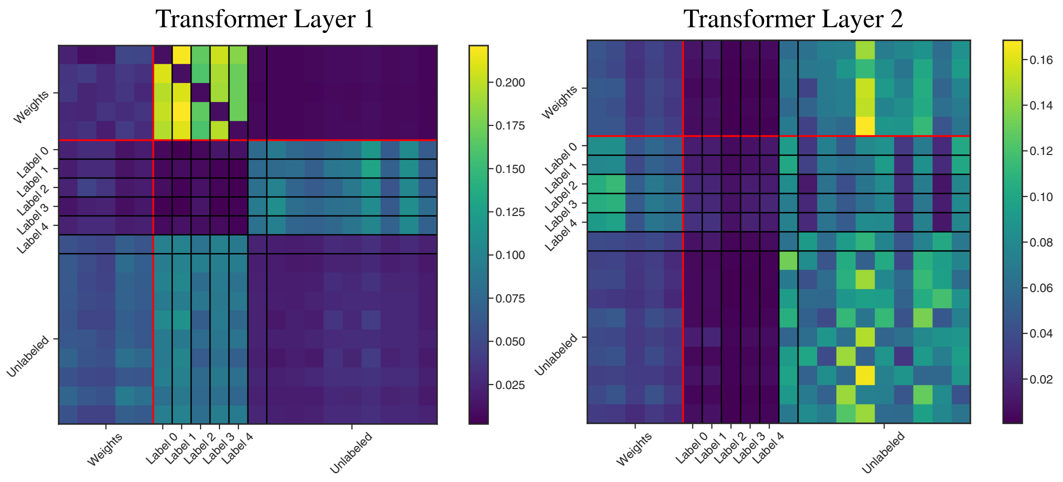

Figure 8: Learned attention weights for 4-layer 8-channel CNN network generated with 1-head, 2-layer transformer for 5-shot TIEREDIMAGENET with additional unsupervised samples (2 per class). Only the last layer of CNN is generated.

The attention maps for a semi-supervised learning problem with a 2-layer transformer is shown in Figure 8. One thing to notice is that a mechanism similar to the one described above appears to be used in the first transformer layer, where weight slices $W_{c,\cdot}$ attend to all labeled samples with labels $c_i \neq c$. At the same time, unlabeled samples can be seen to attend to labeled samples in layer 1 (see "Unlabeled" rows and "Label ..." columns) and the weight slices in layer 2 then attend to the updated unlabeled sample tokens (see "Weights" rows and "Unlabeled" columns in the second layer). This additional pathway connecting labeled samples to unlabeled samples and finally to the logits layer weights is again reminiscent of the simplistic semi-supervised learning mechanism outlined in Sec. 3.2.

The exact details of these calculations and the generation of intermediate convolutional layers is generally much more difficult to interpret just from the attention maps and a much more careful analysis of the trained model is necessary to draw the final conclusions.

F VISUALIZATION OF THE GENERATED CNN WEIGHTS.

Fig. 9 and 10 show the example of the CNN kernels that are generated by 1-head, 1-layer transformer for a simple 2-layer CNN model with 9x9 kernel with stride 4. The difference in figures consists in the way we re-assemble the weights after generation: using output allocation or spatial allocation (see Section 3.1 in the main text for more information). Notice that spatial weights produce more homogeneous kernels for the first layer comparing to the output. For both figures we show difference in final kernels for 3 variants: model with both layers generated, one generated and one trained and both trained.

Trained layers are always fixed for the inference for all the episodes, but the generated layers vary, albeit not significantly. In Fig.11 and 12 we show the generated kernels for two different episodes and, on the right, the difference between them. It looks like the values of the kernel change withing 10-15% form episode to episode.

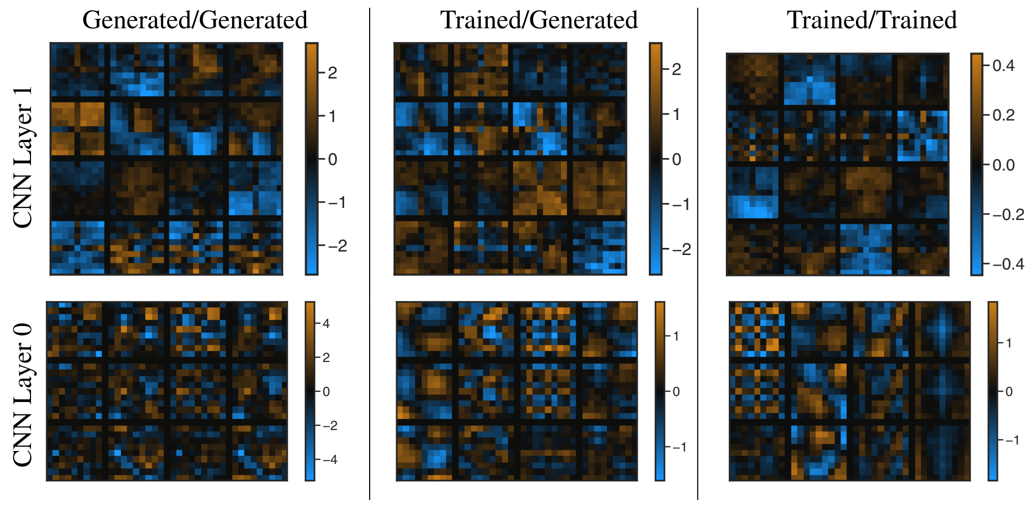

Figure 9: Visualizing convolutional kernels for two layer network with CNN kernel size 9x9 and stride 4 trained on MINIIMAGENET. Each column shows 1 model with two layers and MLP head that is always generated by the transformer. *Left:* both CNN layers are generated, *center:* first CNN layer is trained, second is generated, *right:* both CNN layers are trained. Layer weight allocation: ‘output‘

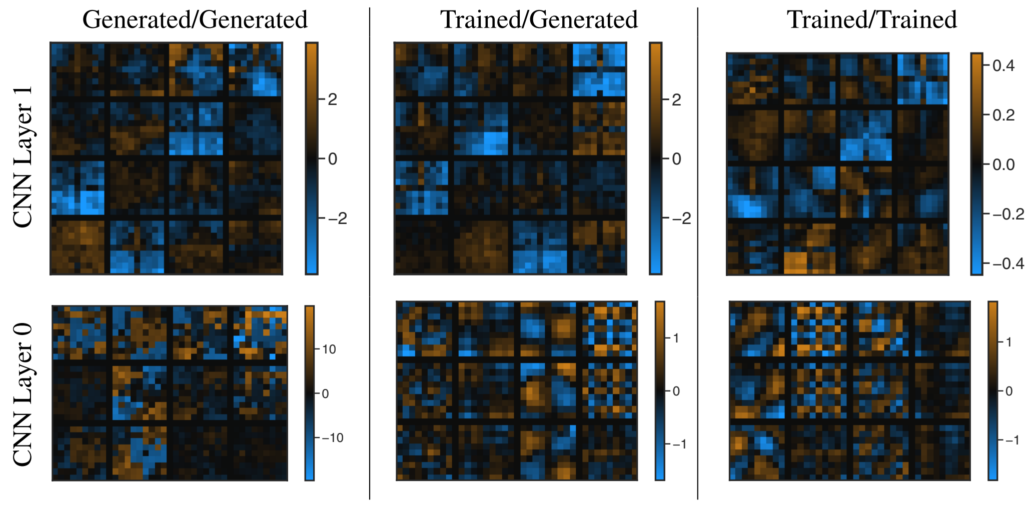

Figure 10: Visualizing convolutional kernels for two layer network with CNN kernel size 9x9 and stride 4 trained on MINIIMAGENET. Each column shows 1 model with two layers and MLP head that is always generated by the transformer. *Left:* both CNN layers are generated, *center:* first CNN layer is trained, second is generated, *right:* both CNN layers are trained. Layer weight allocation: ‘spatial‘

G ADDITIONAL TABLES AND FIGURES

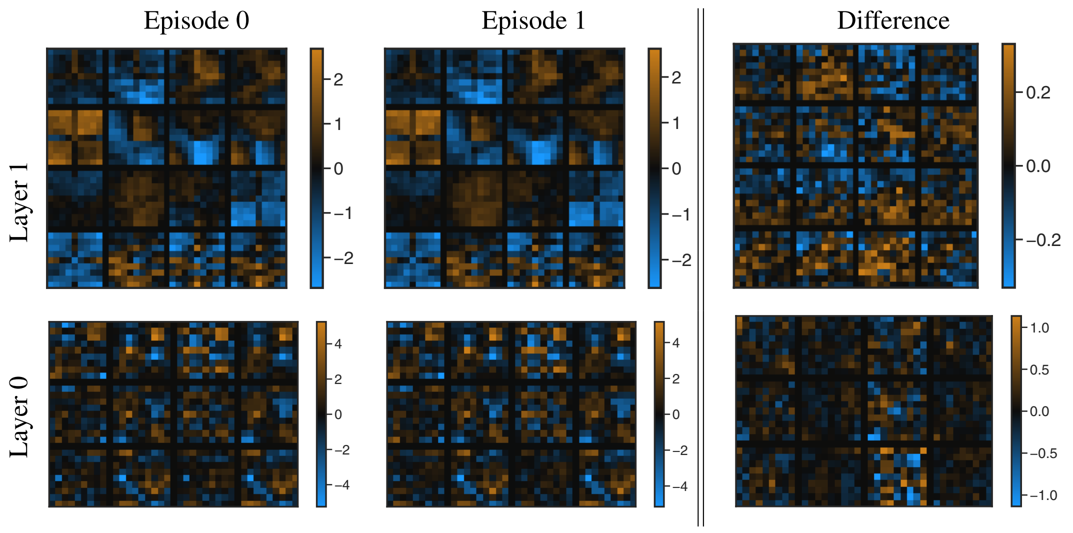

Figure 11: Visualizing generated convolutional kernels in a two layer model for two different episodes. *Left two plots:* kernels for two random episodes of 5 classes, *right:* the difference in generated kernels for two episodes. Layer weight allocation: ‘output‘.

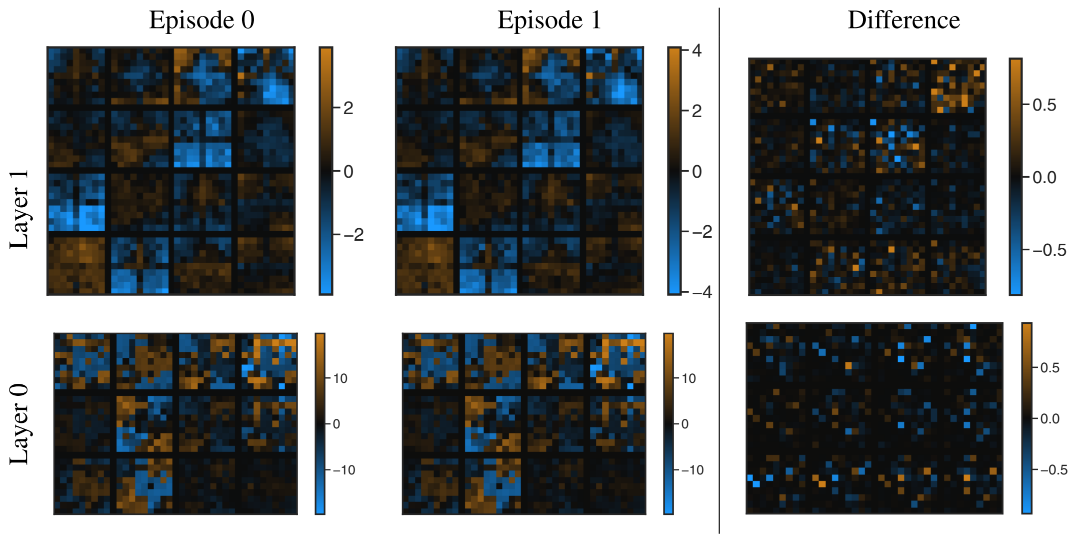

Figure 12: Visualizing generated convolutional kernels in a two layer model for two different episodes. *Left two plots:* kernels for two random episodes of 5 classes, *right:* the difference in generated kernels for two episodes. Layer weight allocation: ‘spatial‘.

Table 4: Average model test and training accuracies (separated by a slash) for the models of different sizes. "Logits" row shows accuracies for the model with only the fully-connected logits layer generated from the support set. It can be interpreted as a method based on a learned embedding. "All" row reports accuracies of the models with some or all convolutional layers being generated. We were not able to see a statistically significant evidence of an advantage of generating more than one convolutional layers.

	4-channel	6-channel	8-channel
Logits	77.9 / 79.2	90.0 / 91.4	94.4 / 95.8
All	82.0 / 83.4	90.7 / 92.0	94.6 / 96.0

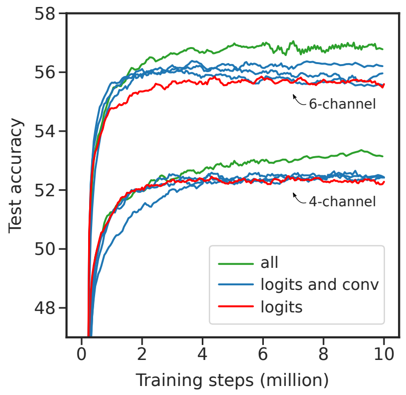

Figure 13: Test accuracies for the generated 4-channel and 6-channel CNN models on the 5-shot-5-way TIEREDIMAGENET task. Models with only the last logits layer generated (*red*) are characterized by the lower test accuracies compared to the models with some or all convolutional layers also being generated (*blue* and *green*).