Differentiable Weight Masks for Domain Transfer

Akash Velu
Stanford University
avelu@stanford.edu

Samar Khanna
Stanford University
samar.khanna@stanford.edu

Skanda Vaidyanath
Stanford University
svaidyan@stanford.edu

Abstract

One of the major drawbacks of deep learning models has been their lack of ability to generalize to new, but similar domains while maintaining their performance on the original domain they were trained on. Specifically, in vision models, shifting from a domain such as "sketch images" to "painting images" can lead to a drastic drop in performance on the original domain, even for the same task when we fine-tune on the new domain. In some fields such as continual learning, this problem is referred to as catastrophic forgetting. In this work, we provide methods to analyse the weights of a neural network that are important for the source domain and provide a fine-tuning method that can improve performance on the target domain while maintaining performance on the source domain. We also analyse the weights that need to be changed the most when going from one domain to another, which can provide insightful information to train vision models in the future.

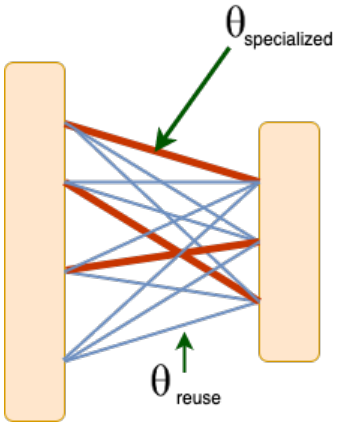

Figure 1. We hypothesize that after training a vision model on a source domain, certain weights (denoted $\theta_{\text{specialized}}$ are highly specialized to the source domain and should not be edited when fine-tuned in the source domain, whereas the other weights (θ_{reuse} can be modified.

1. Introduction

Deep learning algorithms have proven to be extremely successful in a wide range of supervised learning tasks [2] [19] in vision and language. These methods are capable of utilizing large datasets to learn models which can generalize to new datapoints which are within the training data distribution. Although some particularly large models such as GPT-3 and DALLE-2 have exhibited emergent qualities of generalizing to *new* data distributions, smaller models in vision and language tend to suffer from *distribution shift*.

Often, when practitioners want to improve the performance of their model on a new domain that is out of distribution from the one they trained on, a natural idea is to fine-tune the weights of the model with data from this new domain. This often tends to improve the performance on the new (target) domain but tends to lose performance on the original (source) domain. However, we'd like models to be able to maintain their performance on the source domain while improving on the target domain. For instance, a model used in a self-driving car which is adapted to a new driving environment should retain strong performance in the original environment it was trained on, while achieving competitive performance in the new environment. Another example is when a image classifier is trained one domain like paintings and when this is used to fine-tune on a new domain with cartoon images, it loses its performance on the original painting domain. This is related to the *catastrophic forgetting* problem in continual learning where a model trained on a new task tends to forget older tasks it learned.

In this project, we examine methods by which models can be adapted to a new domain *without* experiencing significant catastrophic forgetting. One way to achieve this could be by modularizing the network and splitting the weights as being specific to the source domain and others that can be edited to improve performance on the target domain without a drop in the source domain. We take inspiration from methods that edit the weights of a trained model as a means of adapting its performance. Specifically, we focus on differentiable weight masks [5] and model editor networks literature such as [17], methods which either directly modify or mask the weight matrices of a trained neu-

ral network in order to analyze the network's properties or to edit its performance on specific datapoints. In our setting, we use methods similar to these to identify weights that can be modified to retain source domain performance while achieving improved target domain performance.

Overall, our contributions are as follows:

1. We provide algorithms that can modularize a pretrained network by classifying the weights as those important to the source domain and those that can be fine-tuned for the target domain.

2. We demonstrate that these methods do not deteriorate much in performance in the source domain while improving performance on the target domain.

3. We also analyse the weights that get modified in this process and offer explanations and insights for the same.

2. Related Work

Transfer Learning Transferring a model trained on one dataset to a new dataset has been a common paradigm in both computer vision [6] and natural language processing [7, 3]. In these settings, the learning algorithm typically has learned a particular task utilizing large set of training data for that task, and must use the knowledge gained from this task to transfer to a new task (from which the learning algorithm has a limited set of data). Our setting is very similar to this; however, our setting assumes that at the learning algorithm must adapt to a distribution shift (typically in the inputs) and not a new prediction task. Furthermore, we explicitly care about performance in the first task in addition to the model's performance on the transfer task.

Domain Adaptation Domain adaptation takes a similar setup to transfer learning, with the primary difference being that the learning algorithm has access to training data from *multiple* source domains and must "transfer" to a target domain. Many domain adaptation methods explicitly take advantage of the multiplicity of source domains to which they have access; for instance, domain adversarial feature learning methods [9] aim to learn representations that are invariant across all domains, and mixture of experts methods use ensembling style approaches to combine models from multiple domains [10]. The setting in which we operate is similar to this setting, but we assume access to only one source domain.

Multi-Task Learning Multi-task learning is another setting that tackles the problem of simultaneously learning multiple learning tasks at once. Here, methods typically assume simultaneous access to all training tasks of interest, whereas domain adaptation and transfer learning settings typically Analyzing the *modularity* of networks in this setting has been an approach adopted by many different works [20, 22] – in our work, we hope to also analyze and utilize the modularity of networks, but in the setting of domain adaptation from 1 source domain.

Continual Learning As with transfer learning and multi-task learning, continual learning is a field in which a model must learn multiple tasks at once. However, in continual learning, tasks are presented in a sequential manner; once a task's data is presented to the model for training, the model no longer has access to that task's data and is presented with a new task's data. However, at the end of training, the model is evaluated on *all tasks*, and must hence retain predictive performance on previous tasks. Common methods in continual learning gate hidden units in the network [16], or examine at synaptic stability of weights [23]. Our setting is highly similar to a 2-task continual learning setup, in which the source domain is the first task presented to the model, and the target domain is the second task presented.

3. Problem Setup

Consider a source domain task $\mathcal{S}$ and a model trained to performance inference on this domain f_θ. We have a target domain task $\mathcal{T}$ we would like to generalize to. We would like to modify f_θ to obtain a model $f_{\theta'}$ that performs well on both the source and target domains. This modification must be achieved with a dataset $\mathcal{D}_\mathcal{T}$ from the target domain and the original dataset from the source domain was $\mathcal{D}_\mathcal{S}$. *We no longer have access to $\mathcal{D}_\mathcal{S}$ once the model has been pre-trained.* We would like to achieve this by modularizing the weights of f_θ by splitting the parameters θ into $\theta_{\text{specialize}}$ and θ_{reuse} as described in the [5] paper.

4. Datasets

We are conducting our initial experiments on the PACS dataset [15]. PACS is an image dataset for domain generalization. It consists of four domains, namely Photo (1,670 images), Art Painting (2,048 images), Cartoon (2,344 images) and Sketch (3,929 images). Each domain consists of the same seven classification categories: dog, elephant, giraffe, guitar, house, and person. We would like to study the generalization performance on each pair of domains from this set. For instance, we could first train on the Photos dataset as the source domain, and evaluate performance on the Cartoon dataset.

5. Method

As described in the previous sections, the core idea is that modularizing the weights of a pretrained network into θ_{reuse} and $\theta_{\text{specialize}}$ can help in alleviating the catastrophic forgetting issue. The idea is that we can freeze the $\theta_{\text{specialize}}$ weights when fine-tuning on the target domain so we don't

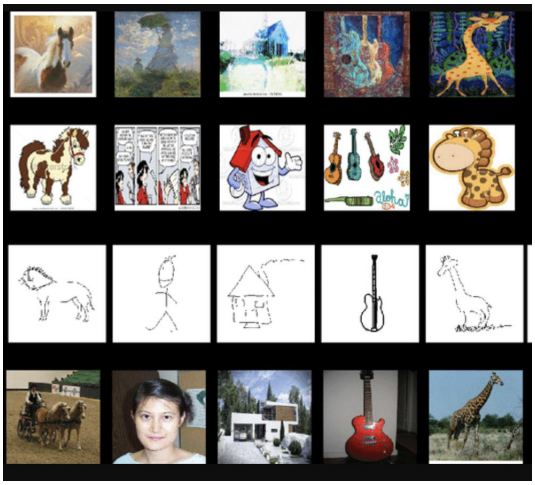

Figure 2. Images from the PACS dataset. The PACS dataset consists of 4 domains – art, cartoon, sketch, and photos. The domains consist of the same categories of images.

change the weights that are important to the source domain in the process of transfer learning.

There are several methods one can think of to split the weights into $\theta_{\text{specialize}}$ and θ_{reuse}. One common theme we can think of is a training scheme that outputs a binary output for each weight on whether we are allowed to modify that weight or not. In other words, for a given layer of weights $\mathcal{W} \in \mathbb{R}^D$, we would like to learn a binary mask $\mathcal{M} \in \{0,1\}^D$, where if the mask has a value of 1 then the weight must be frozen in the fine-tuning step and if its 0 then the weight is allowed to change. In all our experiments, for computational reasons we only mask the final layer of the pre-trained model.

5.1. Naive masking

First, we try a naive method to learn a mask $\mathcal{M}$ from the weights $\mathcal{W}$. To this end, we first analyze the distribution of the weights in $\mathcal{W}$ and noticed the distribution looked normal as shown in figure 4.

One naive idea is that the weights that are far away from the mean are the important weights and the ones close to the means are not very important for the task. Let the mean of the weights in $\mathcal{W}$ be μ and the standard deviation be σ. We set $\mathcal{M}$ such that the values in $\mathcal{M}$ are 1 if $(w > \mu + \sigma) \mid (w < \mu - \sigma)$ where w is each weight in $\mathcal{W}$. The idea is that the "extreme" weights are the ones important for the source task and represent $\theta_{\text{specialize}}$ and must be kept frozen for the fine-tuning process.

5.2. Editor networks based

Ideally we would like to learn the mask $\mathcal{M}$ using a learning process. However, learning a binary mask can prove to be challenging since discrete distributions cannot be differentiated through easily. One idea is to borrow concepts from model editor networks literature [17, 4] to learn an auxiliary $\Delta\mathcal{W} \in \mathbb{R}^D$ from which we can recover $\mathcal{M}$ using some simple thresholding. The idea is to pretrain a model on some source domain $\mathcal{S}$ to convergence. Let the last layer weights of this learned model be $\mathcal{W}$. We would like to learn a $\Delta\mathcal{W}$ such that we can edit the weights of the learned model by replacing $\mathcal{W}$ by $\mathcal{W} + \Delta\mathcal{W}$ *and we train on the same source domain.* We would like to make $\Delta\mathcal{W}$ such that we make as many edits as possible to the already tuned weights $\mathcal{W}$ without suffering too much of an increase in the cross-entropy loss (on the source domain). In other words, we are asking the question: "How much can I edit my pre-trained weights so that I still maintain my performance on the source domain?". We would like to edit as many weights as possible because if its possible to edit the weight while maintaining the performance on the source domain, then the weight probably does not belong to $\theta_{\text{specialize}}$. We would like to find a small set $\theta_{\text{specialize}}$ so that we can gain maximum performance on the new target domain without losing out on performance on the source domain. To ensure that our learned $\Delta\mathcal{W}$ tries to edit as many weights as possible in the learning process, we add a "L1 regularizer" that encourages the $\Delta\mathcal{W}$ values to be non-zero. Here we use "L1 regularizer" in quotes because this is doing the opposite job of a regularizer in the sense that it is increasing the values of the parameters instead of making them smaller. In this sense, it is more like an anti-regularizer. So the overall objective is to reduce the cross-entropy loss (in the same source domain) while also reducing this regularization loss.

$$\mathcal{L}_{\text{edit}}(\Delta\mathcal{W}) = \text{CE\_Loss}(\mathcal{W} + \Delta\mathcal{W}) + \beta \sum_i |\Delta\mathcal{W}_i|$$

In the above equation, β is a constant that controls the regularization strength and $\beta < 0$ to indicate that we want to increase the second term. Also note that the loss is only a function of $\Delta\ \mathcal{W}$ or in other words, there is a stop gradient over $\mathcal{W}$ and we are not fine-tuning these weights at this stage. Also note that this $\Delta\mathcal{W}$ is not the output of another model and is just a trainable parameter in implementation. In other words, $\Delta\mathcal{W}$ is not a function of a particular input but rather a function of the entire domain and the weights $\mathcal{W}$.

Now that we have our trained auxiliary mask, $\Delta\mathcal{W}$, we can recover a $\mathcal{M}$ from this by thresholding the values in $\Delta\mathcal{W}$ according to some heuristics.

Intuitively, values in $\Delta\mathcal{W}$ which have large magnitude indicate that the corresponding weights could be edited without drastically impacting the performance, whereas entries in $\Delta\mathcal{W}$ which are small in magnitude reflect that the original weights must be kept fairly intact in order to maintain performance.

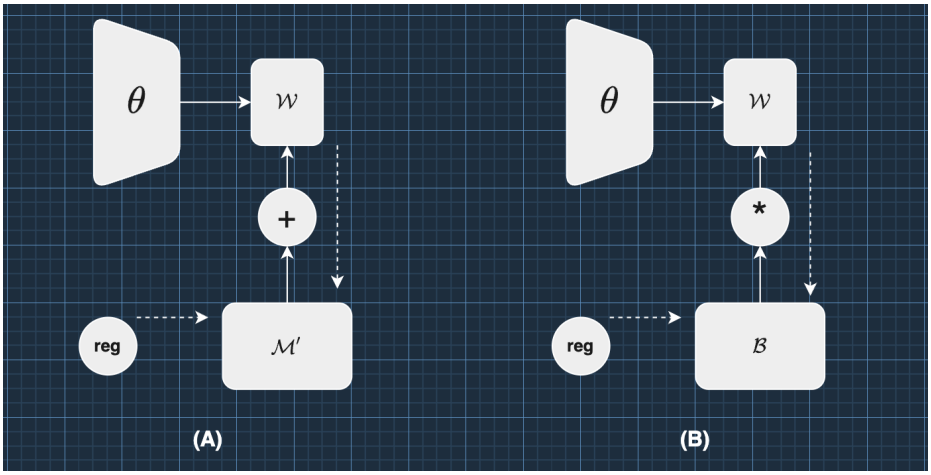

Figure 3. The figure above depicts the models described in sections 5.2 and 5.3. In figure (A), we show the model described in section 5.2 based on learning real-valued edits to the final layer $\mathcal{W}$ shown above and in (B), we show the model to learn a binary mask over $\mathcal{W}$ as in section 5.3. "reg" refers to regularization and the dotted lines represent gradients. Note that there is no gradient flowing into θ and we don't update $\mathcal{W}$ either.

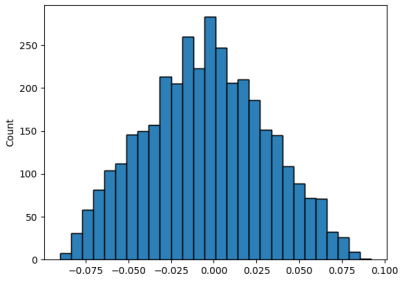

Figure 4. Distribution of weights after training on the Photo domain

Following this intuition, we first compute the magnitude of $\Delta\mathcal{W}$ by taking the absolute value of its entries (abs($\Delta\mathcal{W}$)); then, we freeze weight values which are less than 1 standard deviations than the mean of the absolute values. This results in a binary mask $\mathcal{B}$.

5.3. Binary masks

Our final goal is to recover a mask $\mathcal{M}$ such that we know which weights in $\mathcal{W}$ belong to $\theta_{\text{specialize}}$ and θ_{reuse}. In the previous two methods, we explored techniques where we have a real-valued mask from which we recover a binary mask via thresholding. Can we learn a binary mask directly as a part of the training process? In other words, given $\mathcal{W}$, can we directly learn $\mathcal{M}$ without having to go through auxiliary masks as we saw in the previous method? If we are able to learn a $\mathcal{M}$ directly as a part of the training process, we can simply freeze the weights indicated by a 1 in $\mathcal{M}$ and fine-tune the rest. Further, the weights that are getting masked out, i.e. 0 in $\mathcal{M}$ should not be very useful for the source task since the performance on the source task doesn't fall by much when we mask these weights.

In general, it is difficult to learn discrete masks like the binary mask $\mathcal{M}$ we desire with gradient based methods like neural networks since generally, we cannot differentiate through discrete sampling operations. Sampling from continuous distributions like normal distributions is still differentiable because of the famous re-parametrization trick such as in [14].

However, recent work [13] has shown that the Gumbel distribution can be used to generate discrete samples from a neural network in a differentiable fashion [5, 18]. The method in this section is based on the technique described in [5] and we describe it in detail below.

The training procedure is as follows: once we have a pre-trained model on the source domain, we would like to learn a binary weight mask $\mathcal{M}$ that will replace the final $\mathcal{W}$ of the model with $\mathcal{W} * \mathcal{M}$ where * represents an element-wise multiplication operation. We represent $\mathcal{M}$ as a parameter so once again, it is not the output of any model since we don't expect $\mathcal{M}$ to change with input but rather just be a function of the domain and $\mathcal{W}$. We train $\mathcal{M}$ as a real-valued logits parameter but during the sampling process, we sample binary masks using $\mathcal{M}$ using the Gumbel-sigmoid procedure which is derived from the Gumbel-softmax trick in [5]. Specifically, we can sample $s_i \in [0, 1]$ from the real-valued mask $\mathcal{M}$ as:

$$s_i = \sigma((l_i - \log(\log U_1 / \log U_2))/\tau)$$

Here, τ is the temperature parameter of the Gumbel distribution and we fix it as 1 for all our experiments. Typically τ tends to be harder to tune and often needs to be annealed during the training process [13]. However, for our experi-

ments, we found this setting to work well. U_1 and U_2 are drawn from $U(0, 1)$ which is the uniform distribution between 0 and 1. l_i is a single logit in the real mask $\mathcal{M}$.

We now binarize the s output to get a binary mask $\mathcal{B}$ with values $b_i \in \{0, 1\}$ as follows:

$$b_i = [\mathbb{1}_{s_i > 0.5} - s_i]_{\text{stop}} + s_i$$

Here, we have used the straight-through estimator [12, 1] to ensure that the binary samples are differentiable. $[\cdot]_{\text{stop}}$ represents a stop-gradient indicating we don't differentiate through that term.

Now that we have this binary mask, we multiply this mask element-wise with the frozen model weights from the pre-trained model to get the output. Our objective is to decrease cross-entropy loss while masking out as many weights as possible.

$$\mathcal{L}_{\text{mask}}(\mathcal{M}) = \text{CE\_Loss}(\mathcal{B} * \mathcal{W}) + \beta \sum_i \mathcal{M}_i$$

Here, the regularization term is similar to the one we saw in the previous section but here $\beta > 0$ since we want to drive down the sum as much as possible since we want to mask as many weights as possible. The regularization term is crucial in recovering a fairly sparse mask at the end of training. During training, for stability, we sample multiple masks per batch as prescribed in [5]. At the end of training or during validation, we can get a binary mask from $\mathcal{M}$ as follows:

$$\mathcal{B} = \mathbb{1}_{\mathcal{M} > 0}$$

5.4. Using the trained masks for fine-tuning

Once we have trained our masks on the source domain $\mathcal{S}$, we can fine-tune our model on the new target domain $\mathcal{T}$, while not losing performance on the source domain. As mentioned, for a binary mask $\mathcal{B}$, $b_i = 1$ denotes that parameter i is important for the model to perform well in the source domain. Therefore, for all parameters θ_i such that $b_i = 1$ (i.e. $\theta_i \in \theta_{\text{specialize}}$), we let $\theta_i = \theta_i^{\mathcal{S}*}$, where $\theta^{\mathcal{S}*}$ denotes the fully trained weights on the source domain. For all parameters θ_j such that $b_j = 0$ (i.e. $\theta_j \in \theta_{\text{reuse}}$), we have a few options: (i) $\theta_j = \theta_j^{\mathcal{S}*}$, where we start updating θ_j from its final trained value on $\mathcal{S}$ (ii) $\theta_i^{\mathcal{S}}$, where $\theta^{\mathcal{S}}$ denotes the original initialisation of the weights for the model *before* it was trained on the source domain $\mathcal{S}$. The rationale is that each weight θ_i reached its values relative to other weights given its specific initalisation, and to allow the final $\theta_{\text{specialize}}$ weights to interact better with θ_{reuse}, it might be beneficial to reset θ_{reuse} to the original initialisation. This motivation is inspired from [8]. (iii) θ_i^{random}, where we also consider a fully random re-initialisation of θ_{reuse}. More details about these initialization strategies are in section 6.6. Our default initialization strategy is (ii).

6. Experiments

In this section we explore the effect of varying masking strategies with respect to performance on the target domain and the performance on the source domain after fine-tuning on the target domain. We would like to improve performance on the target domain and make sure we're not losing too much performance on the source domain.

For all our experiments we use a ResNet-18 backbone [11] initialized with ImageNet [21] weights. We then train a MLP head on the source domain and fine-tune on the target domain as described in section 5.4.

6.1. Baselines

In our plots in figures 5 and 6, we compare the performance of the various masking methods on both the source and target domain with the performance of models which are fine-tuned on the target domain but do not use any masking methods. Ideally, masking methods will surpass the performance of this baseline.

For the target domain, we compare against direct fine-tuning on the target domain without any masking (which acts as the oracle on target domain performance) and training on the target domain initialized with ImageNet weights.

6.2. How do the different masking strategies compare against each other?

The performance of the different masks on the target domain and the source domain (after fine-tuning on the target domain) are shown in figures 5 and 6 respectively. In each figure, each sub-plot represents the four different source domains and the three groups of bars represent the performance on the three other domains as target domains.

6.3. Source domain performance

In all the source domain plots, we see that the binary masking technique (green bar) does much better than the other strategies in alleviating the catastrophic forgetting issue. The blue line indicates the performance on the source domain after fine-tuning on the target domain. The edit-based masking strategy and the naive masking strategy are only able to beat the baseline in some settings. We suspect this is probably because of poor choices of thresholds for the real-valued masks that can probably be tuned better for each domain.

6.4. Target domain performance

In the target domain performance, we see that the performances of various masking strategies are very similar. However we do notice here that the real-valued masking

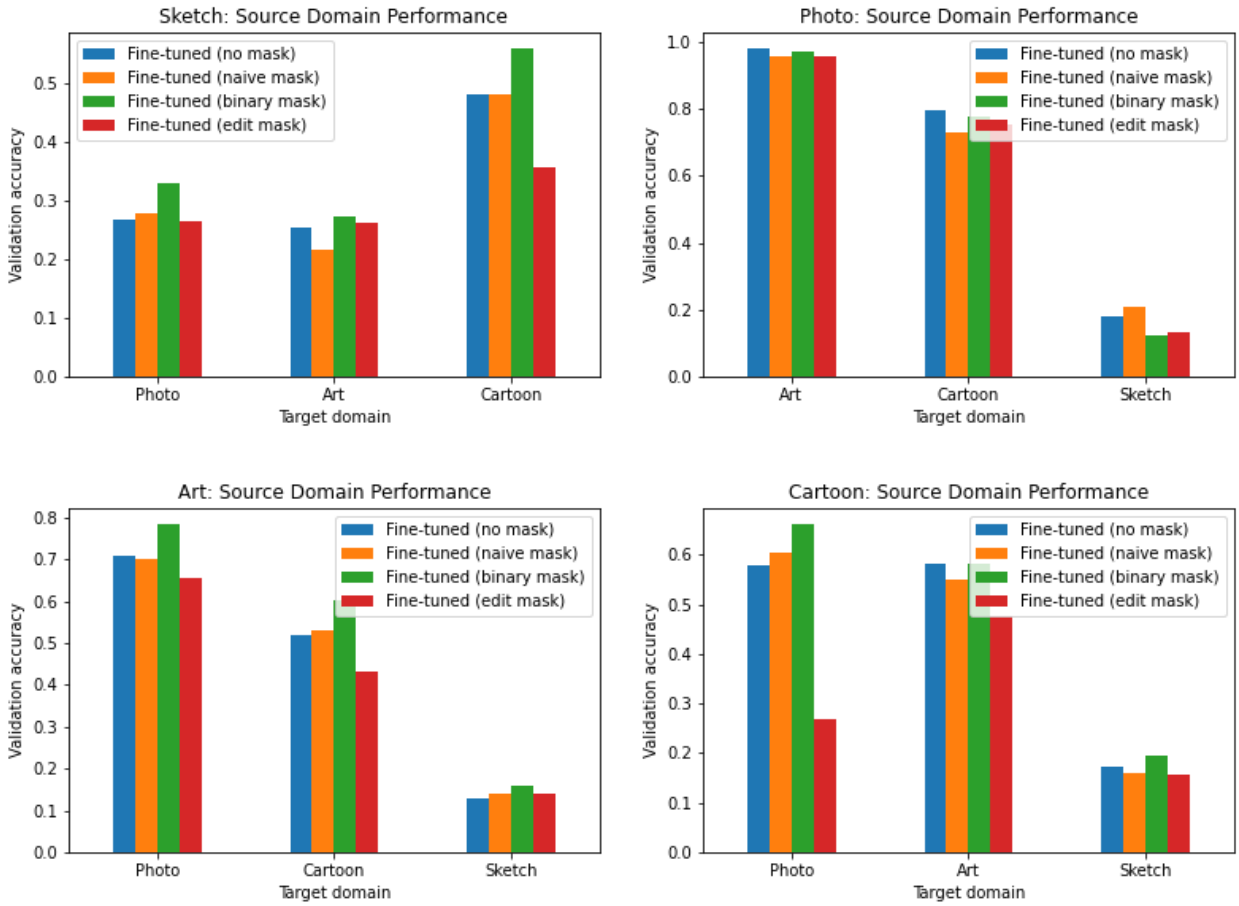

Figure 5. Performance of various masking strategies on the source domain after fine-tuning on the target domain

strategies outperform the binary masking strategy in several cases. The blue and purple bars show the performance when we fine-tune on the target domain without a masking strategy and where we fine-tune on the target domain starting from random weights respectively. As we will see in the next section, it is quite impressive that we are able to reach close to this performance on the target domain because we find that the masking process tends to freeze most weights. The binary masking strategy freezes the most weights compared to the real-valued masks and this explains why it shows better performance on the source domain and not as good performance on the target domain. Of course, the fact that real-valued masks do not freeze as many weights may be a thresholding issue. However, it is surprising that even with a small subset of weights, we are able to achieve close to optimal performance on the target domain.

6.5. How sparse are the trained masks using the binary masking strategy?

Figure 7 show the sparsity of the masks on the four domains when we use the binary masking strategy. In these figures, yellow represents weights that are frozen i.e. $\theta_{\text{specialize}}$ and purple represents weights that are fine-tuned i.e. θ_{reuse}. As we can see, the learned masks encourages more weights to be frozen. This is perhaps because of our initial bias in the training procedure (refer to section 5.3). It is quite surprising then that we are able to achieve close to the optimal performance on the target domain even when most of the weights are frozen.

6.6. How does the performance of the binary masking strategy vary with different weight initialization strategies on the target domain?

We compare the three different weight initialization strategies for the binary masking method as described in section 5.4. The results are presented in table 6.6. The performance is close across the different methods but as we can see, the initialization with the initial weights on the source domain as described in method (ii) in section 5.4 does the best and this is in line with the results in [8].

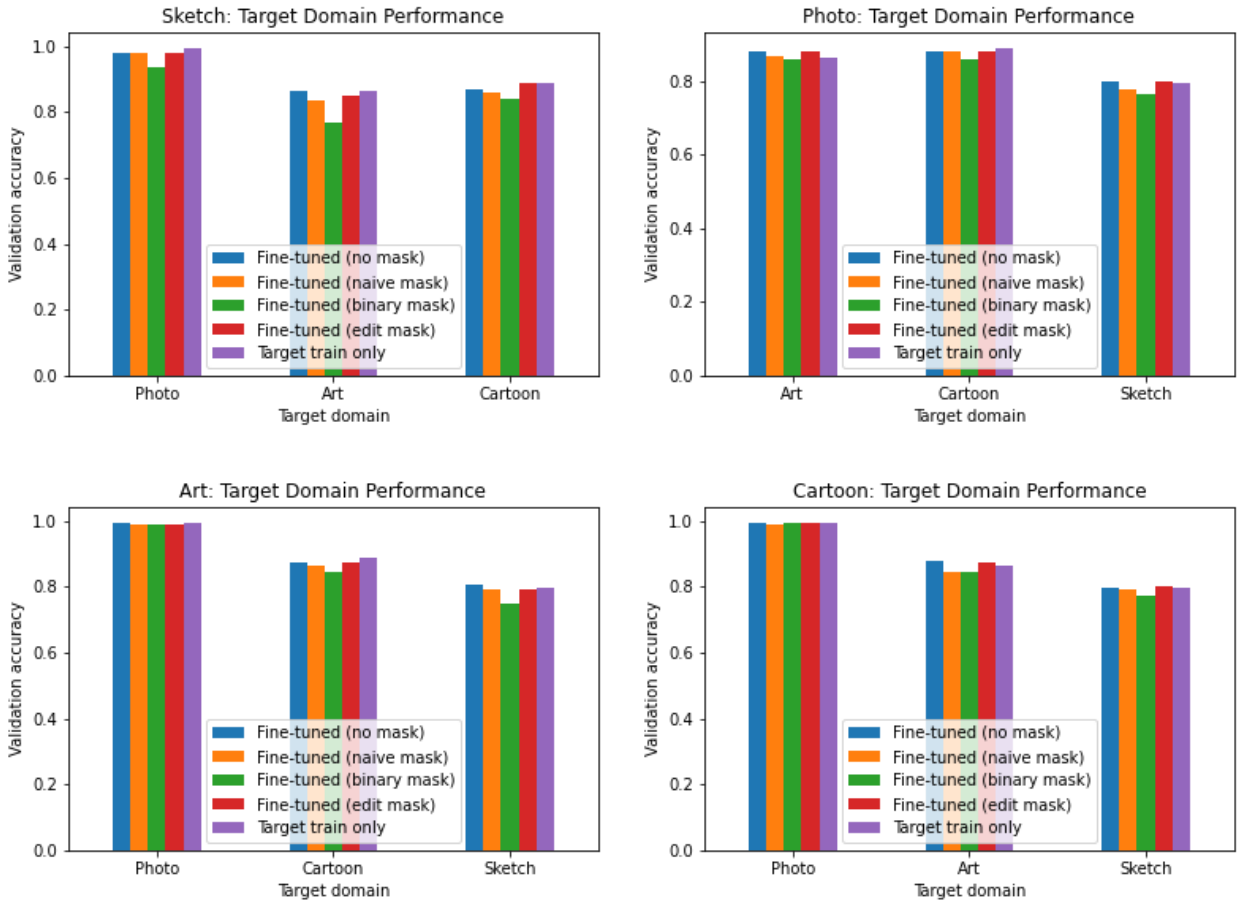

Figure 6. Performance of various masking strategies on the target domain after fine-tuning

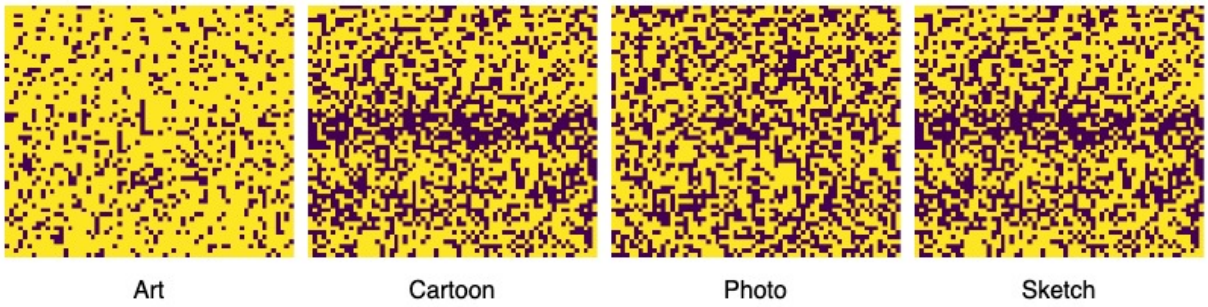

Figure 7. Visualizations of binary masks trained on the four domains. Yellow grids represent weights that are frozen (i.e., the weights are not updated during fine-tuning), whereas purple grids represent weights that can be updated during fine-tuning.

7. Conclusion and Future Work

In this project, we compared the efficacy of 3 different masking strategies and their effect on alleviating the catastrophic forgetting problem on transfer learning settings. We showed that the binary masking strategy is effective at alleviating catastrophic forgetting while the real-valued masking strategies are better at target domain performance. We also analysed the sparsity of the learned masks and the effect of different weight initializations in the fine-tuning process.

There are several directions that we would like to take this project in as a part of future work.

We would like to study a few more masking strategies and how they compare with these techniques and what their properties are. For example, we could take inspiration from

Domain	Photo		Art		Cartoon		Sketch	
Method	$\mathcal{S}$ Gain	$\mathcal{T}$ Drop	$\mathcal{S}$ Gain	$\mathcal{T}$ Drop	$\mathcal{S}$ Gain	$\mathcal{T}$ Drop	$\mathcal{S}$ Gain	$\mathcal{T}$ Drop
$\mathcal{S}$ Start	-0.029	**-0.025**	0.062	**-0.030**	**0.035**	-0.019	**0.054**	**-0.056**
$\mathcal{S}$ End	**-0.014**	-0.027	0.051	-0.038	0.021	**-0.016**	0.043	-0.059
Random	**-0.014**	-0.027	**0.064**	-0.036	0.001	-0.021	0.023	-0.058

Table 1. Ablation study to determine best method of initialisation for θ_{reuse}. $\mathcal{S}$ Gain represents the average performance gain (in accuracy) on the source domain, and $\mathcal{T}$ drop represents the average performance drop on the target domains after using the masking strategy. Higher values are better.

the field of information theory to ask the question "which weights in my final layer give me the most information about my output logits?" or "mask weights such that the output distribution before and after masking are close". The first question can be modeled with a Mutual Information $\mathbb{I}$ based objective and the second question can be modeled using KL-divergence or Wasserstein distance.

We could also consider editing a low-rank approximation of the weights $\mathcal{W}$ such that the highest eigenvalues are kept fixed, while we allow edits on the lower eigenvalue decomposition. This could be useful for learning masks on high-dimensional weight vectors where learning an edit for each weight might be expensive.

We can also imagine that such a technique could be used for multi-task learning or continual learning problems where we are learning more than just two tasks (so we have more than just a source and target domain) and even after learning K tasks, we want to make sure that we remember the first task that we learned.

8. Acknowledgements

We would like to thank the course staff of CS231N for providing valuable feedback on this project and our methods.

References

[1] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. 5

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020. 1

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. J. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020. 2

[4] N. D. Cao, W. Aziz, and I. Titov. Editing factual knowledge in language models. In *EMNLP*, 2021. 3

[5] R. Csordás, S. van Steenkiste, and J. Schmidhuber. Are neural nets modular? inspecting functional modularity through differentiable weight masks. In *International Conference on Learning Representations*, 2021. 1, 2, 4, 5

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 2

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019. 2

[8] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018. 5, 6

[9] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *J. Mach. Learn. Res.*, 17(1):2096–2030, jan 2016. 2

[10] J. Guo, D. J. Shah, and R. Barzilay. Multi-source domain adaptation with mixture of experts. In *EMNLP*, 2018. 2

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015. 5

[12] G. Hinton. Neural networks for machine learning. coursera, video lectures, 2012. 5

[13] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. 4

[14] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013. 4

[15] D. Li, Y. Yang, Y.-Z. Song, and T. Hospedales. Deeper, broader and artier domain generalization. pages 5543–5551, 10 2017. 2

[16] N. Y. Masse, G. D. Grant, and D. J. Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, 115(44):E10467–E10475, 2018. 2

[17] E. Mitchell, C. Lin, A. Bosselut, C. Finn, and C. D. Manning. Fast model editing at scale. In *International Conference on Learning Representations*, 2022. 1, 3

[18] I. Mordatch and P. Abbeel. Emergence of grounded compositional language in multi-agent populations, 2017. 4

[19] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents, 2022. 1

[20] C. Rosenbaum, T. Klinger, and M. Riemer. Routing networks: Adaptive selection of non-linear functions for multi-

task learning. In *International Conference on Learning Representations*, 2018. 2

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge, 2014. 5

[22] R. Yang, H. Xu, Y. Wu, and X. Wang. Multi-task reinforcement learning with soft modularization, 2020. 2

[23] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3987–3995. JMLR.org, 2017. 2