Automatic fine-tuning

Nipun Agarwala
Stanford University
nipuna1@stanford.edu

Haider Suleman
Stanford University
hs605@stanford.edu

Neal Vaidya
Stanford University
nealv@stanford.edu

Abstract

As it becomes increasingly arduous to train modern deep-learning pipelines from scratch, the practice of utilizing pretrained foundation models and fine-tuning them on a new task has become an important part of machine learning. Fine-tuning is particularly relevant when the target task is out of distribution relative to the source. Several fine-tuning schemes have been proposed in the literature. The common theme in existing works is that they propose some freezing/unfreezing scheme that better preserves the learned structure than full fine-tuning. Here, we take the perspective that adapting to distribution shifts while preserving learned features reduces to choosing a separate learning rate for each layer during fine-tuning.

In this paper, we propose three novel methodologies for choosing how much tune each layer during fine-tuning. **(1)** RELATIVE GRADIENT NORM – based on the fundamental intuition that the gradient encodes the new information contained in a target dataset, we fine-tune more the layers with a larger change in gradient between the source and target data. We also look to the second-order gradient to extract richer information about the optimal learning rate for each layer. **(2)** LAYERWISE SURPRISE – using the pretrained network itself as a feature extractor, we measure the shift between the features in the source and target data by comparing the distributions of the activations at each layer. The layers which exhibit a greater divergence are then adapted more during fine-tuning. This reflects the intuition that the nature of the distribution shift is important for determining which layers should be fine-tuned. **(3)** MODULE CRITICALITY – uses the notion that the criticality of a layer to the network's prediction should drive the learning rate weights during fine-tuning. We measure criticality by measuring the increase in loss on either the source or target data due to layer removal. Layer removal is modelled through either randomization, or the identity mapping.

We evaluate the empirical performance of our fine-tuning strategies against existing methods in experiments across four image classification tasks containing either input, feature or output level distribution shifts. Across input-level and feature-level shifts, MODULE CRITICALITY with random initialization performs better than our other methods and full fine-tuning. For output-level shifts, RELATIVE GRADIENT NORM with second order gradients performs best. We can conclude that finding important layers for fine-tuning and weighting them does provide performance benefits. Specifically, incorporating gradient and loss curvature information is beneficial. But we also note that methods like MODULE CRITICALITY and LAYERWISE SURPRISE required hyper-parameter tuning, which requires careful exploration in itself for best results.

In the future, we hope to investigate our methods on larger and more challenging datasets, along with modern neural architectures like Transformers. We also would like to apply these methods to other domains like NLP.

1 Introduction

Pre-training deep neural network architectures on tasks with large datasets, and fine-tuning them to make predictions on tasks with smaller, out-of-distribution (OOD) datasets (a form of transfer learning) is common in modern deep learning. The power of fine-tuning stems from leveraging pre-trained networks, which have learned useful representations, to generalize from an original source task to a related target task with little compute and few new fine-tuning examples. However, a significant challenge in fine-tuning is effectively preserving the generic structure learned in the pre-training stage, while adapting to the distribution shifts present in the new task.

The general strategy is to either fine tune the entire network or manually select parts of the network to be fine-tuned, often the first or last layer. Manual layer selection has significant costs including expensive cross-validation, search space investigation and hyper-parameter tuning. Our objective is to instead create automatic fine-tuning strategies which mitigate the need for discretionary modelling decisions, deep understanding of distribution shifts between source and target tasks, and large a large amount of computational resources. We evaluate the empirical performance of our fine-tuning strategies against existing methods in experiments across four image classification tasks containing diverse distribution shifts.

In this paper, we make the following contributions:

1. Propose an optimization and gradient based finetuning strategy to better incorporate the loss curve of the model on target set
2. Propose a data driven strategy that explicitly measures distribution shifts across the source and target sets to fine-tune layers
3. Use a theoretical framework to find important layers in a network and finetune them based on this importance
4. Compare and contrast these methods against one another along with manual and automatic tuning baselines

2 Related Work

In attempts to preserve pre-trained information and prevent overfitting to the target data, existing approaches to fine-tuning use various learning-rate schemes ([Li+20];[RC21]), layer freezing ([LAV21];[LTL19]) or regularization ([LGD18];[LCK20]). These methods either fail to account for the nature of the distributional shift or require significant manual experimentation. Nonetheless, they provide important benchmarks for our methodologies.

Recent work in automatic fine-tuning [Lee+22] provides two baselines for gradient-based selection: “Relative Gradient Norm" (AUTO-RGN) and “Signal-to-Noise Ratio" (AUTO-SNR). While both of these metrics use a different normalization, the core idea is similar – layers which have gradients with a relatively larger magnitude on average should be fine-tuned more. The intuition is that larger gradients contain more information about the target task. We propose extensions to these baselines through improved normalization, relating for example to the curvature of the loss surface or to aspects of the training process.

The existing feature importance literature provides an approach to evaluating which input features of networks drive performance. $\mathcal{V}$-information [Xu+20] has been shown to be a useful metric for reasoning about which input features to a model are important to the final prediction. By doing a series of ablation tests, authors of [OA21] were able to find what kinds of context features in language inputs were most important to the predictions. We use the theoretical framework proposed in [Xu+20] as a foundation for finding which layers are the most important for performance. In addition, [ZBS19] and [CNS20] have shown that randomization/re-initialization of layers can be used to determine which layers or modules are the most important to a network’s generalisation.

Several methods exist for detecting and categorizing distribution shift in general, which can be useful for understanding when and how fine-tuning needs to be performed. These methods are often built around some form of dimensionality reduction on the feature space followed by a statistical test to quantify the distance between the two data distributions [RGL19]. Black Box Shift Detection [LWS18] leverages the learned representation of a black box model trained on a different task as the reduced dimension representation of the original features. Black Box Shift Detection is generally

performed on only the final layer of the network, utilizing the softmax outputs of a classifier network as the feature representation.

3 Problem formulation

Here, we adapt the setup of [Lee+22], whereby the goal is to allow for different learning rates for each layer during fine-tuning. We can denote the full pre-trained model as $f = f_{\mathcal{L}} \circ \ldots \circ f_1$, where each layer f_ℓ has parameters θ_ℓ, and the subset of the model up to layer ℓ as $f_{1:\ell}$. We'll also define $L(f, \mathcal{D})$ as the loss of the model f on the dataset $\mathcal{D}$. Our fine-tuning setup aims to minimize the target loss by allowing for a tuned learning rate α_ℓ at each layer during fine-tuning:

$$\underset{\theta_\ell : i \in \{1,\ldots,n\}}{\operatorname{argmin}} \; L(f, \mathcal{D}_{\text{tgt}})$$
$$\text{with} \quad \theta_\ell^{(t)} = \theta_\ell^{(t-1)} - \alpha_\ell^{(t)} \nabla_{\theta_\ell} L(f, \mathcal{D}_{\text{tgt}})$$

where we have $t \in \{1, \ldots, T\}$ as the number of epochs and $\alpha_\ell \in \{\alpha_1, \ldots, \alpha_{\mathcal{L}}\}$ as the per layer learning rates. Note that this also encompasses the case where we only fine-tune a subset of the layers – just set $\alpha_\ell = 0$ for the layers we want to freeze during fine-tuning. Below, we primarily attempt to determine α_ℓ based on the distribution shift between the source and target data. We also investigate whether we can optimally determine the learning rates by using the notion of layer importance on the training data (formalized by the ideas of module criticality and $\mathcal{V}$-information).

We evaluate the performance of our fine-tuning strategies across two benchmarks and four tasks. The tasks have been selected to ensure that our proposed strategies are evaluated on at least one input-level, feature-level and output-level shift:

- **CIFAR-10 → CIFAR-10C** ([HD19];[Kri09]) We classify images into one of 10 categories, where the images in the target data have low-level corruptions w.r.t the source data. We run experiments over 14 corruptions (frost, gaussian blur, gaussian noise, glass blur, impulse noise, jpeg compression, motion blur, pixelate, saturate, shot noise, snow, spatter, speckle noise, and zoom blur), and report the average test set accuracy across all corruptions. There are 5 severity levels of corruption available, and we evaluate on the most severe
- **BREEDS (Living17 and Entity30)** [STM21] BREEDS is a methodology for deriving benchmarks for subpopulation shifts. The classes across the source and target datasets are the same, but the subpopulations are disjoint. For example, the source dataset could only contain black bears for the class "Bear", while the target dataset would contain only polar bears for the same class. Living17 and Entity30 are created by applying the BREEDS methodology to ImageNet. The former is an image classifcation task over 17 animal categories, while the latter is an image classification task over 30 entities
- **CIFAR-10 → CIFAR-10F** The task here is again a 10-way image classification problem. While the distributions of the input data are the same across the source and target, we simulate an output-level shift by flipping the labels from the source to the target. I.e. if the label for a particular class is y in the source dataset, then it is $9 - y$ in the target dataset

4 Automatic Fine-Tuning Strategies

We summarise our proposed methodologies below and explain the baselines that we used. Note that some of them require access to at least a subset of the training set, which is not always possible.

4.1 Baselines

In our experiments, we benchmark our novel methodologies against three baselines for determining layer weights during fine-tuning:

(a) FULL FINE-TUNING: all layers are weighted equally
(b) AUTO-RGN: at each fine-tuning step, layers with a higher relative gradient norm get a higher learning rate (see Section 4.2)
(c) CROSS VALIDATION: fine-tune a small, contiguous subset of layers and freeze the rest. The best subset to fine-tune is determined using performance on a validation set

4.2 Relative Gradient Norm

[Lee+22] showed that for fine-tuning, the learning rate for each layer can be weighted by the value of their proposed AUTO-RGN method. Explicitly, for a layer ℓ of a pretrained network f_θ:

$$\text{Auto-RGN}_\ell = \frac{\|\nabla_{\theta_\ell} L(f, \mathcal{D}_{\text{ft}})\|_2}{\|\theta_\ell\|_2}$$

Motivated by some desirable properties of second order optimization methods (e.g. Newton step) over first order methods (e.g. SGD) (see [Yao+20]), we extend AUTO-RGN by using the Newton Step:

$$\theta_\ell^{(t)} = \theta_\ell^{(t-1)} - \nabla_\theta^2 L^{-1} \nabla_\theta L$$

before aggregating into layer-wise metrics. Second order methods account for the curvature of the loss surface, which leads to a better descent direction and automatically incorporates information about the best learning rate for each parameter. Our hope is that preconditioning with the inverse Hessian will prevent the AUTO-RGN metric from being skewed by an *ill-conditioned* gradient, thereby providing a cleaner estimate for which layers contain the most information about the target task. To reduce the cubic or quadratic computational complexity, we use the LBFGS inverse hessian approximation ([Byr+95]), which has a linear time and memory complexity.

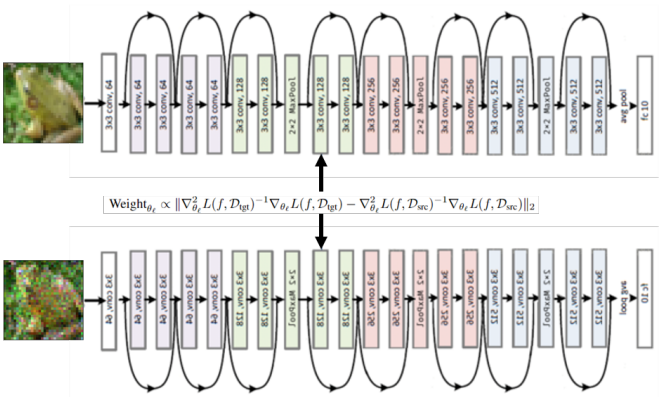


Figure 1: Illustration of AUTO-RGN+HESSIAN+CHANGE

Algorithm 1: AUTO-RGN+HESSIAN+CHANGE

Data: $\mathcal{D}_{\text{src}}, \mathcal{D}_{\text{tgt}}^{\text{support}}, \mathcal{D}_{\text{tgt}}^{\text{query}}$
Model : f pretrained on $\mathcal{D}_{\text{src}}$
Result: f^* fine-tuned on $\mathcal{D}_{\text{tgt}}^{\text{support}}$
$\alpha \leftarrow 1 \cdot 10^{-5}$;
$f^* \leftarrow f$;
for $\ell \in 1 \dots \mathcal{L}$ **do**
  $X \leftarrow$ `NewtonStep(`f_{θ^*}, ℓ,D_{src}`)`;
  $Y \leftarrow$ `NewtonStep(`f_{θ^*}, ℓ,$D_{tgt}^{support}$`)`;
  $w \leftarrow \|X - Y\|_2 / \|\theta_\ell^*\|_2$;
  $\alpha_\ell \leftarrow w \cdot \alpha$;
end
for $\ell \in 1 \dots \mathcal{L}$ **do**
  $\theta_\ell^* \leftarrow \theta_\ell^* - \alpha_\ell \cdot \nabla_{\theta_\ell} L(f, \mathcal{D}_{\text{tgt}}^{\text{support}})$
end

Furthermore, while [Lee+22] use the gradient on the fine-tuning set alone to determine how to fine-tune layers, we propose using the *change* in gradient between the training and fine-tuning sets. The intuition is that a large gradient alone is not sufficient to indicate the presence of new information in the fine-tuning set if the gradient is also large for the original training set (which might occur, for example, if early stopping was used).

Putting it all together, we introduce a weighted scheme called AUTO-RGN+HESSIAN+CHANGE (illustrated in Figure 1 and Algorithm 1), which extends AUTO-RGN by measuring the norm of the change in the preconditioned gradient:

$$\text{AUTO-RGN+HESSIAN+CHANGE}_l = \frac{\|\nabla^2_{\theta_\ell} L(f, \mathcal{D}_{\text{tgt}})^{-1} \nabla_{\theta_\ell} L(f, \mathcal{D}_{\text{tgt}}) - \nabla^2_{\theta_\ell} L(f, \mathcal{D}_{\text{src}})^{-1} \nabla_{\theta_\ell} L(f, \mathcal{D}_{\text{src}})\|_2}{\|\theta_\ell\|_2}$$

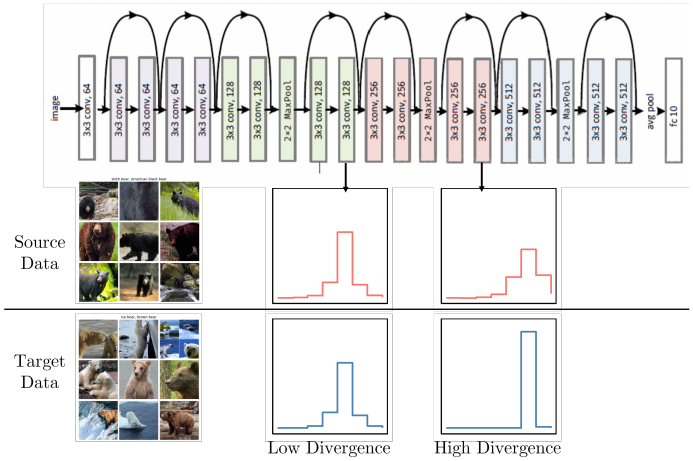
In our experiments, we also evaluated the cases of extending AUTO-RGN exclusively with either the "Hessian" or "Change" elements, and we denote these specifications as AUTO-RGN+HESSIAN and AUTO-RGN+CHANGE respectively.

4.3 Layerwise surprise

Our second approach also builds on the notion that the best layers for fine-tuning correspond to the type of distribution shift [Lee+22]. We try to explicitly model the type of distribution shift between the source and target datasets. Leveraging the idea that different parts of a neural network act as feature extractors – and in particular the idea that the layers of a network represent a hierarchy of the complexity of those features [Bau+20] – we propose adding a drift detector to each layer of our

network. With that, our goal is to determine which specific features are causing the data distribution to shift, and which parts of the network these features are represented by.

To quantify the amount of drift at each layer, we use LAYWERISE SURPRISE, adapted from [EMW22]. For each layer ℓ of the network, we want the distribution of the activations from both our source and target datasets, $p(f_\theta^\ell(\mathcal{D}_{tr})$ and $p(f_\theta^\ell(\mathcal{D}_{ft})$ respectively. We can determine how divergent – or "surprising" [IB09] – the target dataset is by calculating the Kullback-Leibler divergence D_{KL} [Kul59]. Without full knowledge of the parameters of these distributions, we can estimate D_{KL} by using the empirical distribution of f_θ^ℓ calculated on samples from $\mathcal{D}_{tr}$ and $\mathcal{D}_{ft}$ and binned to discrete intervals. To approximate the divergence for very high dimensional multivariate distributions, we perform a univariate divergence measurement on each of the K dimensions of our distribution [RGL19].



Algorithm 2: LAYERWISE SURPRISE

Data: $\mathcal{D}_{\text{src}}, \mathcal{D}_{\text{tgt}}^{\text{support}}, \mathcal{D}_{\text{tgt}}^{\text{query}}$
Model : f pretrained on $\mathcal{D}_{tr}$, and sub-models $f_{1:\ell} \in f_1 \dots f_{1:\mathcal{L}}$
Result: f_{θ^*} fine-tuned on $\mathcal{D}_{\text{tgt}}^{\text{support}}$
for $\ell \in 1 \dots \mathcal{L}$ **do**
  $Q_\ell \leftarrow$ distribution of activations $f_{1:\ell}(\mathcal{D}_{tr})$;
  $P_\ell \leftarrow$ distribution of activations $f_{1:\ell}(\mathcal{D}_{ft}^{\text{support}})$;
  $\alpha_\ell \leftarrow \alpha \cdot D_{KL}(P_\ell || Q_\ell)$;
end
for $t \in 1 \dots T$ **do**
  for $\ell \in 1 \dots \mathcal{L}$ **do**
    $\theta_\ell^* \leftarrow \theta_\ell - \alpha_\ell \cdot \nabla_{\theta_\ell} L(f_\theta, \mathcal{D}_{\text{tgt}}^{\text{support}})$
  end
end

Figure 2: Illustration of LAYERWISE SURPRISE

Our full procedure is to sample from both our source and target datasets and calculate $f_{1:\ell}(x)$ for each sampled point, where $f_{1:\ell}$ is the output of the activation function on the ℓ^{th} layer. We flatten the activations and for each dimension K, we bin them into one of 8 bins across the range of values for that dimension. The bin counts are normalized to give us probabilities $\pi_1^{k,src} \dots \pi_8^{k,src}$ and $\pi_1^{k,tgt} \dots \pi_8^{k,tgt}$. From these, we can calculate the LAYERWISE SURPRISE for each layer ℓ

$$s_\ell = \sum_{k \in K} \sum_{i=1}^{8} \pi_i^{k,tgt} \log \frac{\pi_i^{k,tgt}}{\pi_i^{k,src}} \tag{1}$$

s_ℓ is then used as a relative weighting factor for the learning rates of each layer. We also experiment with a modified surprise score, LAYERWISE SURPRISE DIFF, $s_\ell^* = \min(0, s_\ell - s_{\ell-1})$ to isolate the surprise due to a particular layer from the previous layers.

4.4 Module Criticality

The two important questions to answer when we want to automatically fine-tune are: which layers to fine-tune and how much to tune those layers. We have seen previous methodologies pick $\alpha_\ell \in \alpha_1 \dots \alpha_{\mathcal{L}}$. Here, we will see how to pick a layer $f_\ell \in f_1 \dots f_{\mathcal{L}}$ using $\mathcal{V}$-Information[Xu+20]. Specifically, we want to find the predictive influence of layer f_ℓ with parameters θ_ℓ on f. Let the input be X and predictions be Y. We also assume that each layer has minimized its own loss. We can re-purpose the formulation in [Xu+20] as the following:

$$I_{\mathcal{V}}(f_\ell \to f) = H_{\mathcal{V}}(Y|X;(f - f_\ell)) - H_{\mathcal{V}}(Y|X;f)$$

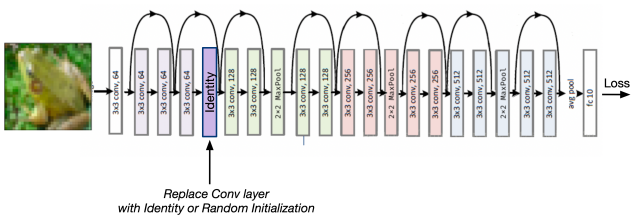


Figure 3: Illustration of MODULE CRITICALITY

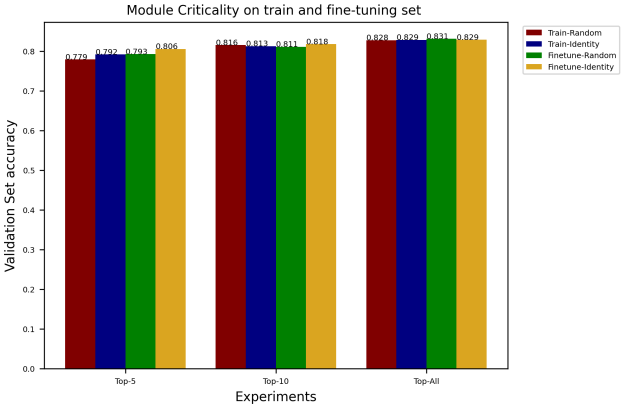


Figure 4: MODULE CRITICALITY hyperparameter investigation

Intuitively, we are removing the influence of f_ℓ from f and measuring what the loss in predictive information is. We can "remove" the influence of a layer by either putting an identity function I for the layer, or randomly initializing it. For convolution layers, $I = \delta$ i.e. the dirac delta function, which is the identity for convolution functions. Note that it might appear that random initialization adds noise, but if we initialize the layer close to its initial weights (using the same initialization mechanism), it can appear to be like the start of the loss curve for that layer. Adapting the above equation to our setup, we get:

$$I_{\mathcal{V}}(f_\ell \to f) = L(f(\theta_1, \ldots, \delta_\ell, \ldots, \theta_n), \mathcal{D}_{\text{src}}) - L(f(\theta_1, \ldots, \theta_n), \mathcal{D}_{\text{src}})$$

Algorithm 3: MODULE CRITICALITY on Train Set

Data: $\mathcal{D}_{\text{src}}, \mathcal{D}_{\text{tgt}}^{\text{support}}, \mathcal{D}_{\text{tgt}}^{\text{query}}$
Model : f pretrained on $\mathcal{D}_{\text{src}}$
Result: f^* fine-tuned on $\mathcal{D}_{\text{tgt}}^{\text{support}}$
$\alpha \leftarrow 1 \cdot 10^{-5}$;
for $\ell \in 1 \ldots \mathcal{L}$ **do**
  $f'_\ell \leftarrow \delta$ or Kaiming Initialization;
  $f' \leftarrow f_1 \circ \ldots \circ f'_\ell \circ \ldots f_n$;
  $w_\ell \leftarrow L(f', \mathcal{D}_{\text{src}})$;
end
TopK $\leftarrow \{\ell \mid w_\ell \in$ top K largest w values$\}$
for $t \in 1 \ldots T$ **do**
  $\mathcal{L}_{\theta_t^*} = \text{CE}(y, f_{\theta_t^*}(\mathcal{D}_{ft}))$;
  for $\ell \in 1 \ldots \mathcal{L}$ **do**
    if $\ell \in$ *TopK* **then**
      $\alpha_\ell \leftarrow \alpha \cdot w_\ell$
    else
      $\alpha_\ell \leftarrow 0$
    end
    $\theta_\ell^* \leftarrow \theta_\ell - \alpha_\ell \cdot \nabla_{\theta_\ell} L(f_\theta, \mathcal{D}_{\text{tgt}}^{\text{support}})$
  end
end

Algorithm 4: MODULE CRITICALITY on Target Set

Data: $\mathcal{D}_{\text{src}}, \mathcal{D}_{\text{tgt}}^{\text{support}}, \mathcal{D}_{\text{tgt}}^{\text{query}}$
Model : f pretrained on $\mathcal{D}_{\text{src}}$
Result: f^* fine-tuned on $\mathcal{D}_{\text{tgt}}^{\text{support}}$
$\alpha \leftarrow 1 \cdot 10^{-5}$;
for *each epoch* e *in finetuning* **do**
  $\{f_1 \ldots f_\mathcal{L}\}$ = train($\{f_1 \ldots f_\mathcal{L}\}$);
end
for $\ell \in 1 \ldots \mathcal{L}$ **do**
  $f'_\ell \leftarrow \delta$ or Kaiming Initialization;
  $f' \leftarrow f_1 \circ \ldots \circ f'_\ell \circ \ldots f_n$;
  $w_\ell \leftarrow L(f', \mathcal{D}_{\text{src}})$;
end
TopK $\leftarrow \{\ell \mid w_\ell \in$ top K largest w values$\}$
for $t \in 1 \ldots T$ **do**
  $\mathcal{L}_{\theta_t^*} = \text{CE}(y, f_{\theta_t^*}(\mathcal{D}_{ft}))$;
  for $\ell \in 1 \ldots \mathcal{L}$ **do**
    if $\ell \in$ *TopK* **then**
      $\alpha_\ell \leftarrow \alpha \cdot w_\ell$
    else
      $\alpha_\ell \leftarrow 0$
    end
    $\theta_\ell^* \leftarrow \theta_\ell - \alpha_\ell \cdot \nabla_{\theta_\ell} L(f_\theta, \mathcal{D}_{\text{tgt}}^{\text{support}})$
  end
end

The larger the $I_{\mathcal{V}}(f_\ell \to f)$, the more important f_ℓ is to f. We will weight the learning rate α_ℓ during the fine-tuning epochs based on this importance measure 3. We should note that the α_ℓ we get are based on importance measured from the training set. Our hope is that this importance transfers to the target set because our hypothesis is that the layers important to the initial training distribution will also be important to the target distribution. We also tried a few modifications to the above algorithm,

as seen in 4. The model is partially fine-tuned on the target set $\mathcal{D}_{\text{partial-tgt}}$ for a few epochs, after which we measure the importance of different layers. We then fine-tune only these specific layers of the pretrained model (before fine-tuning) using the weights obtained after a partial fine-tuning.

To understand the performance of the different hyper-parameters, we ran all of the combinations against the CIFAR-10C validation set 4 to understand trends and pick the best hyperparameter combination for identity and random initialization. We found tuning all layers best, but the trend (which remains fairly consistent and proportional across datasets) was meaningful to note. Additionally, we tuned our network on our fine-tuning set between 0 and 5 epochs to find the layer weights before rolling back our network to the pre-trained version, and chose the weights that gave us best validation set results.

5 Experimental setup

As we are extending the work of [Lee+22], we re-used large parts of their experimental setup.

- **CIFAR-10C** We used WideResNet-28-10 pretrained on CIFAR-10 [Cro+21]. For each corruption, we fine-tune on 1000 images for 25 epochs. The learning rate used was $10^{-5}\alpha_i$ (determined using our fine-tuning strategies). The weight decay was 10^{-4}. Reported test set accuracy is the average across the corruptions
- **Living17 and Entity30** We used MoCo-v2 ResNet-50, pretrained in an unsupervised way on ImageNet-1k[Che+20]. We then trained on the labeled source data for 5 epochs - 2 epochs of linear probing(LP) with a learning rate of 0.1, and 3 epochs of full fine-tuning(FT) with a learning rate of 10^{-5}, following the LP-FT scheme of [Kum+22]. Next, we fine-tuned on the labeled target data for 10 epochs with a learning rate of $10^{-5}\alpha_i$. No weight decay was used. For both Living-17 and Entity30, the target dataset contained 50 images per class.
- **CIFAR-10F** The model used is a WideResNet-28-10 pretrained on CIFAR-10 from [Cro+21]. We fine-tune on 1000 flipped-label images for 25 epochs. The learning rate used was $10^{-5}\alpha_i$. The weight decay was 10^{-4}

6 Results

Table 1: Test set accuracy after fine-tuning on various distribution shifts

Method	**Input-level shift**	**Feature-level-shift**		**Output-level-shift**
	CIFAR-10C	Living17	Entity-30	CIFAR-10F
Full fine-tuning	77.3	93.9	77.9	82.0
Cross validation	81.4	93.9	80.0	93.6
Auto-RGN	81.0	92.2	75.8	87.2
Auto-RGN+Hessian	80.8	93.4	77.3	**88.0**
Auto-RGN+Change	80.7	92.2	74.2	82.6
Auto-RGN+Hessian+Change	80.6	93.6	77.6	80.0
Layerwise Surprise	79.7	91.5	78.1	N/A
Layerwise Surprise Diff	77.6	90.3	67.2	N/A
Mod. Criticality-Identity	81.0	94.1	74.3	84.2
Mod. Criticality-Random	**81.1**	**94.5**	**78.4**	82.5

6.1 Baseline

As mentioned above, we use full fine-tuning, Auto-RGN and Cross Validation as baselines. Of these baselines, CROSS VALIDATION performs best across all distribution shifts. This is because it has the benefit of trying different specification across several runs. Therefore, we treat CROSS VALIDATION as the high-water mark for our methodologies. AUTO-RGN is also a pertinent baseline, as it is the only alternative automatic-fine tuning strategy that takes a similar approach to ours. Full fine-tuning is treated as a "naive" approach, and therefore constitutes a minimum requirement for the performance of our methodologies. Auto-RGN outperforms full fine-tuning on input and output level shifts, but actually underperforms on feature-level shifts.

6.2 Experiments on Distribution Shift Datasets

From the above table, we can make some observations across the various datasets.

6.2.1 Input-level Shift

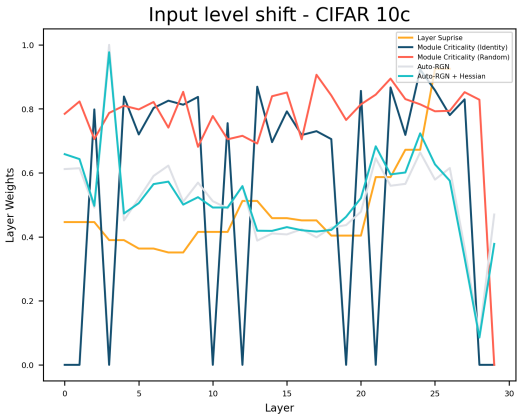


Figure 5: Layer weights for input level shift

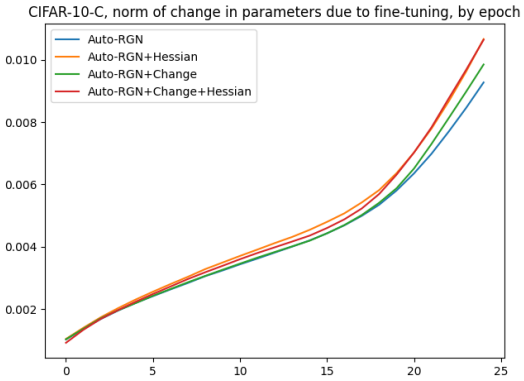


Figure 6: Change in parameters - Auto-RGN

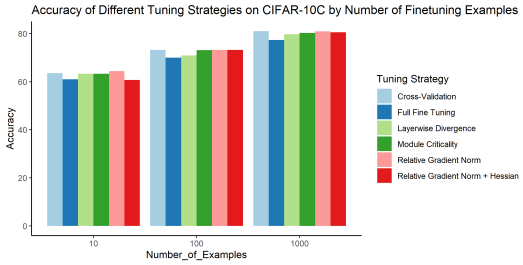


Figure 7: Performance by no. fine-tuning examples

Of our proposed methodologies, the MODULE CRITICALITY specifications perform best, followed closely by the Auto-RGN specifications. The performance of Layerwise Surprise lags the others. The performance of Module criticality is marginally better than the Auto-RGN baseline, and close to cross validation.

All our extensions to Auto-RGN underperform the vanilla specification of [Lee+22]. Figures 5 and 7 provide some clues as to why. Figure 5 shows that the layer weights of Auto-RGN+Hessian mimic those of Auto-RGN. This suggests that preconditioning with the Hessian does not provide useful additional information about the optimal learning rate for each layer. Furthermore, Figure 7 shows that the change in the norm of parameters during fine-tuning is the least for vanilla Auto-RGN. This suggests that Auto-RGN is more “regularizing" without our extensions, which perhaps helps it to generalize better to the test-set. Also, we note from Figure 5 that using the change in gradient between the source and target data does not affect the underlying principle that the Auto-RGN weights reflect the nature of the distribution shift - i.e. more weight is given to the earlier layers.

Counter to our expectations, Layerwise Surprise gives higher weight to the later layers of the network, indicating that this method does not capture the idea that input level shifts should be handled by the earliest layers in a network. This may explain why the Layerwise Surprise methods underperform the other approaches.

We also did an analysis of how each methodology scales with the number of shots. This was to help us to understand how productive each method was in using its examples to fine-tune the network. In general, as we expect, each method performs better as the number of shots is increased. We also see

that the relative performance of the methods remains consistent across methods except $K = 1$, where Module Criticality and Layerwise Surprise do better than Auto-RGN+Hessian. This is because 2^{nd} order gradients might be misleading the loss curve when only one example was provided.

6.2.2 Feature-level Shift

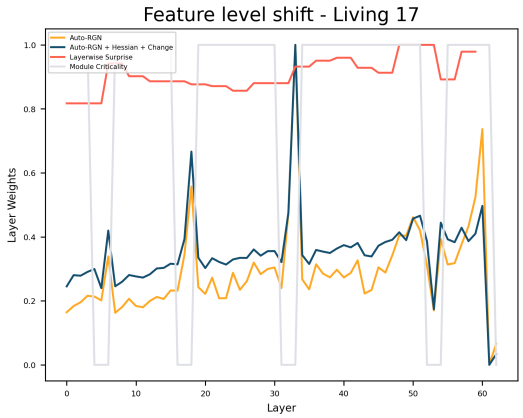


Figure 8: Layer weights for feature level shift

Table 1 shows that MODULE CRITICALITY with random initialization performs best on feature-level shifts, and even identity-based importance is close behind. This suggests that it is able to find important layers in the network that extract relevant features for output predictions even under distribution shifts. We also see that Auto-RGN+Hessian performs better than Auti-RGN, which tells us that Hessian is again providing us with useful curvative information for better autotuning. The layer weights in Figure 8 show that module criticality has 1s and 0s as its weights, because the weights that gave the best validation set accuracy was fine-tuning on the target set for 0 epochs before finding the weights. It is interesting to note that all the weights for the non-0 weighted layers are equal here. Auto-RGN have the most weight to the middle layer and some intermediate layers which shows that it detected the feature level shift and was correcting for it. Layerwise surprise had a uniformly high weights across the board, which probably led to its poor performance.

6.2.3 Output-level Shift

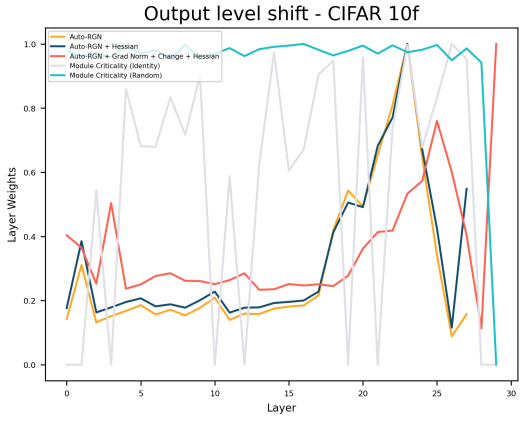


Figure 9: Layer weights for output level shift

Auto-RGN+Hessian performs the best, followed closely by Auto-RGN. This suggests that the 2^{nd} order methods did do better in optimizing the output shift given the same inputs. The layer weights

in 9 also indicate that the methods were able to adapt the final layers to the output shift, while keeping the main feature extractors similar to the pre-trained network. In fact, we can see how Auto-RGN+Hessian weights the last layer much more than Auto-RGN, probably resulting in the higher performance. Module criticality did not do as well since it was not able to find layers (and their weightings) important to flipped labels. The layer weights confirm this, where identity function leads to random weightings and a low weighting for the last layer. The random initializations weight each layer (except the all important last layer) relatively similar. Layerwise surprise was not run on this dataset because we thought it did not make sense to detect output shift, a primary characteristic of the final few layers, throughout the network.

7 Future work

Our current work only runs the above four datasets on ResNet type models, both of which are not fully representative of current state-of-the-art deep learning. We should run larger and more interesting datasets like CelebA and Waterbirds on the models we used. Additionally, we should also consider Transformer-based neural architectures on these datasets to see how different architectures perform with our finetuning methodologies.
One thing to note is that all our runs are with 1 seed, due to limited compute power. So running on multiple seeds and averaging the results would provide more noise-immune results. Finally, we should consider extending our methodologies to domains other than vision like NLP and speech, since feature extraction, layer importance and fine-tuning work differently in different domains.

8 Conclusions

We find that automatic fine-tuning does provide performance benefits over full fine-tuning, though it does not consistently do better than cross-validation. Module criticality does better for input-level and feature-level shifts, while Auto-RGN does better for Output-level shifts. We we also note that automatic fine-tuning did require non-trivial hyper-parameter tuning, especially for layerwise surprise and module criticality that required tuning of number of bins, number of fine-tuning layers, number of fine-tuning epochs etc. We also see that in general, Auto-RGN+Hessian performs better or relatively close to Auto-RGN, which suggests that accounting for the hessian provides useful information and hence should be incorporated in any fine-tuning strategy.

9 Team contributions

Haider Suleman implemented Section 4.2 - Gradient-based methods, Neal Vaidya implemented Section 4.3 - Layerwise surprise, and Nipun Agarwala implemented Section 4.4 - Module Criticality. We all contributed equally to code development and the final write-up.

References

[Kul59] S. Kullback. *Information Theory and Statistics*. Wiley publication in mathematical statistics. Wiley, 1959. URL: https://books.google.com/books?id=XeRQAAAAMAAJ.

[Byr+95] Richard H. Byrd et al. "A Limited Memory Algorithm for Bound Constrained Optimization". In: *SIAM Journal on Scientific Computing* 16.5 (1995), pp. 1190–1208. DOI: 10.1137/0916069.

[IB09] Laurent Itti and Pierre Baldi. "Bayesian surprise attracts human attention". en. In: *Vision Research*. Visual Attention: Psychophysics, electrophysiology and neuroimaging 49.10 (June 2009), pp. 1295–1306. ISSN: 0042-6989. DOI: 10.1016/j.visres.2008.09.007. URL: https://www.sciencedirect.com/science/article/pii/S0042698908004380 (visited on 12/14/2022).

[Kri09] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: 2009.

[LGD18] Xuhong Li, Yves Grandvalet, and Franck Davoine. "Explicit Inductive Bias for Transfer Learning with Convolutional Networks". In: *CoRR* abs/1802.01483 (2018). arXiv: 1802.01483. URL: http://arxiv.org/abs/1802.01483.

[LWS18] Zachary Lipton, Yu-Xiang Wang, and Alexander Smola. "Detecting and Correcting for Label Shift with Black Box Predictors". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 3122–3130. URL: https://proceedings.mlr.press/v80/lipton18a.html.

[HD19] Dan Hendrycks and Thomas Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HJz6tiCqYm.

[LTL19] Jaejun Lee, Raphael Tang, and Jimmy J. Lin. "What Would Elsa Do? Freezing Layers During Transformer Fine-Tuning". In: *ArXiv* abs/1911.03090 (2019).

[RGL19] Stephan Rabanser, Stephan Günnemann, and Zachary Lipton. "Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/hash/846c260d715e5b854ffad5f70a516c88-Abstract.html (visited on 11/27/2022).

[ZBS19] Chiyuan Zhang, Samy Bengio, and Yoram Singer. "Are all layers created equal?" In: (2019).

[Bau+20] David Bau et al. "Understanding the role of individual units in a deep neural network". In: *Proceedings of the National Academy of Sciences* 117.48 (Dec. 2020). Publisher: Proceedings of the National Academy of Sciences, pp. 30071–30078. DOI: 10.1073/pnas.1907375117. URL: https://www.pnas.org/doi/full/10.1073/pnas.1907375117 (visited on 12/14/2022).

[CNS20] Niladri S. Chatterji, Behnam Neyshabur, and Hanie Sedghi. "The intriguing role of module criticality in the generalization of deep networks". In: *ArXiv* abs/1912.00528 (2020).

[Che+20] Xinlei Chen et al. "Improved Baselines with Momentum Contrastive Learning". In: *arXiv preprint arXiv:2003.04297* (2020).

[LCK20] Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. "Mixout: Effective Regularization to Finetune Large-scale Pretrained Language Models". In: *ICLR*. 2020. URL: https://openreview.net/forum?id=HkgaETNtDB.

[Li+20] Hao Li et al. "Rethinking the Hyperparameters for Fine-tuning". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=B1g8VkHFPH.

[Xu+20] Yilun Xu et al. "A Theory of Usable Information Under Computational Constraints". In: *CoRR* abs/2002.10689 (2020). arXiv: 2002.10689. URL: https://arxiv.org/abs/2002.10689.

[Yao+20] Zhewei Yao et al. "ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning". In: *CoRR* abs/2006.00719 (2020). arXiv: 2006.00719. URL: https://arxiv.org/abs/2006.00719.

[Cro+21] Francesco Croce et al. “RobustBench: a standardized adversarial robustness benchmark”. In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021. URL: https://openreview.net/forum?id=SSKZPJCt7B.

[LAV21] Yuhan Liu, Saurabh Agarwal, and Shivaram Venkataraman. “AutoFreeze: Automatically Freezing Model Blocks to Accelerate Fine-tuning”. In: *CoRR* abs/2102.01386 (2021). arXiv: 2102.01386. URL: https://arxiv.org/abs/2102.01386.

[OA21] Joe O’Connor and Jacob Andreas. “What Context Features Can Transformer Language Models Use?” In: *CoRR* abs/2106.08367 (2021). arXiv: 2106.08367. URL: https://arxiv.org/abs/2106.08367.

[RC21] Youngmin Ro and Jin Young Choi. “AutoLR: Layer-wise Pruning and Auto-tuning of Learning Rates in Fine-tuning of Deep Networks”. In: *AAAI*. 2021.

[STM21] Shibani Santurkar, Dimitris Tsipras, and Aleksander Madry. “{BREEDS}: Benchmarks for Subpopulation Shift”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=mQPBmvyAuk.

[EMW22] Cian Eastwood, Ian Mason, and Christopher K. I. Williams. “Unit-level surprise in neural networks”. en. In: *I (Still) Can’t Believe It’s Not Better! Workshop at NeurIPS 2021*. ISSN: 2640-3498. PMLR, Feb. 2022, pp. 33–40. URL: https://proceedings.mlr.press/v163/eastwood22a.html (visited on 11/27/2022).

[Kum+22] Ananya Kumar et al. “Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=UYneFzXSJWh.

[Lee+22] Yoonho Lee et al. *Surgical Fine-Tuning Improves Adaptation to Distribution Shifts*. 2022. DOI: 10.48550/ARXIV.2210.11466. URL: https://arxiv.org/abs/2210.11466.