TLDChoiceNet: Quantitatively Choosing a Transfer Learning Dataset

Jing Ning[1] and James D. Braza[1]

[1]Department of Computer Science, Stanford University

arXiv:2608.09091v1 [cs.CV] 10 Aug 2026

Abstract

Transfer learning is particularly useful in settings with limited training data, and within image classification it is common to transfer learn upon massive datasets like ImageNet,[2] CIFAR-100,[9] or COCO.[12] Qualitatively, it seems a transfer learning dataset should have both more classes and more examples per class than the fine tuning dataset; however, a quantitative method to choose the best transfer learning dataset does not currently exist. In this paper, we design TLDChoiceNet, a model to choose the best transfer learning dataset given a fine tuning dataset by predicting the test-set accuracy after fine-tuning. A simple version 1 achieves 0.154 MSE on the test dataset, while a version 2 leveraging an ImageNet pre-trained ResNet50 v2 embedding with per-class information attains a 5X lower MSE of 0.031. We further design two metrics that enable an unsupervised method of choosing an optimal transfer learning dataset: distribution distance (DD), which linearly regresses against fine-tune accuracy with an R2 of 0.89, and average class correlation (ACC), which improves the R2 to 0.97. Our results underscore that a dataset's low-level statistics can explain the transfer learning effect, and that using a pre-trained ImageNet can embed different classes further apart in latent feature space.

1 Introduction

Consider the following scenario. You have a small dataset for a target task that could be used for fine tuning. You are aware of three open-source datasets A, B, and C (all with similar example and class count) and pre-trained model weights for A, B, C to use, but only have the resources to fine-tune with one. Which pre-trained dataset would you choose? We designed a model TLDChoiceNet which takes datasets A, B, C, and/or pre-trained weights and predicts the test-time accuracy after fine-tuning. Figure 1 gives a high-level overview of this pipeline.

TLDChoiceNet can also provide:

- Pre-trained model to select the best subset class dataset for joint fine-tuning
- Summary statistic(s) illustrating why dataset B is desirable.
- Insights into whether dataset or pre-trained weights have the most impact on fine-tune performance.

We believe answers to these questions could aid the machine learning community at large, and help practitioners speed their transfer learning process in novel areas.

Contributions. This paper makes the following contributions:

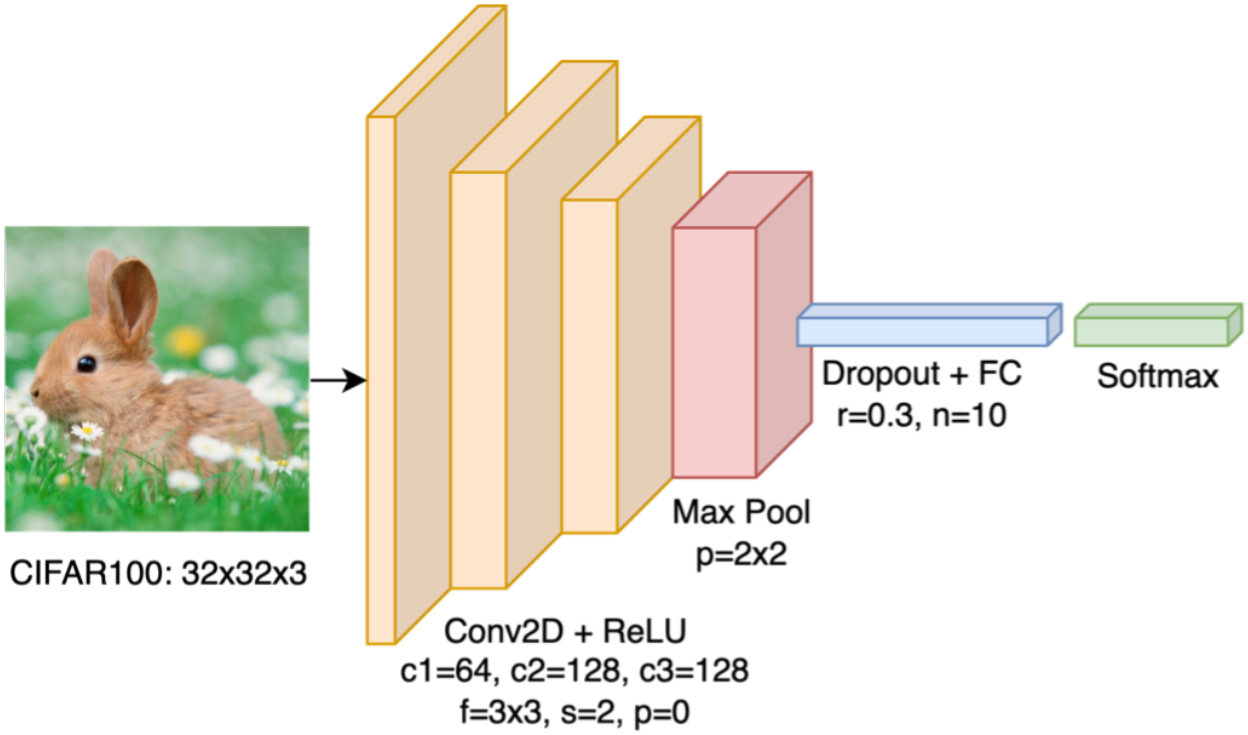


Figure 1: **Overview of TLDChoiceNet.** Given a target fine-tuning dataset and one or more candidate transfer learning datasets (and/or their pre-trained models), TLDChoiceNet embeds each input, reduces it through a learned LoRA-similar layer, and predicts the post-fine-tuning test-set accuracy. This lets a practitioner choose the best transfer learning dataset *without* actually fine-tuning on every candidate.

- We introduce the **Transfer Learning Dataset (TLDS)**, a dataset of (transfer learning dataset, transfer-learned model, resultant test accuracy) 3-tuples spanning similar, dissimilar, random, and no-transfer-learning cases.
- We design **TLDChoiceNet (v1 and v2)**, a network with a novel dataset embedding and reduction pipeline that predicts post-fine-tuning test-set accuracy, reducing test MSE 5X from v1 to v2.
- We propose two **unsupervised metrics**, distribution distance (DD) and average class correlation (ACC), that select a transfer learning dataset without any additional training, reaching an R^2 of 0.97 against fine-tune accuracy.
- We provide **insights into what is transferred**, showing that pre-trained ImageNet weights embed dissimilar classes further apart in latent feature space.

2 Related Work

Transfer learning training techniques have been extensively studied, such as selection of fine-tuning layers, fine-tuning robustness, and amenable network architectures.[11] Krizhevsky et al.[10] showed via training of a Restricted Boltzmann Machine and Deep Belief Net that a network is capable of extracting useful features when trained on other image datasets (e.g. hand-written digits, faces).

Another study[6] focused on using pre-training to improve model robustness and uncertainty.

There is also another research area focusing on how to choose the best data subset for joint fine-tuning. Weifeng et al.[4] detail a method to choose the subset of images possessing low-level characteristics similar to the target task, and how to jointly fine-tune shared convolutional layers for both tasks. Some other interesting approaches[3] seek to optimally group transfer learning tasks for joint fine-tuning by examining gradients.

Some other areas of research focus on analyzing the fundamental similarity of neural networks. Kornblith et al.[8] developed a quantitative metric called centered kernel alignment to determine the similarity between neural networks. Peters et al.[14] researched fine-tuning and feature extraction performance, showing strong correlation between closely aligned tasks. Neyshabur et al.'s work[13] has a similar objective as our work: discovering what enables a successful transfer, and which part of the network is responsible. They also presented that some transfer learning benefits come from the low-level data statistics of the data, and showed that pre-trained weights remain within the same loss landscape basin after fine-tuning.

In this paper, we use supervised and unsupervised training techniques, and investigate with multiple custom metrics. Moreover, we also provide insights into what is being transferred, and leverage pre-trained model activations as an embedding method.

3 Method

3.1 The ChoiceNet Algorithm

We first summarize the full TLDChoiceNet v2 forward pass in 1; the remaining subsections detail each component. Version 1 follows the same pipeline with class-agnostic embeddings and a concatenation (rather than additive) combination step. Let D_f be the fine-tuning dataset and D_t a candidate transfer-learning dataset with transfer-learned model M_t. We write $g(\cdot)$ for the headless ImageNet-pretrained ResNet50 v2 activation map, with $g(x) \in \mathbb{R}^{2048}$, and $\phi_{M_t}(\cdot)$ for the last-Conv2D activations of M_t, with $\phi_{M_t}(x) \in \mathbb{R}^{1152}$.

The learnable parameters are $\theta = \{A_f, A_t, W_1, w_2\}$, and the model predicts $\hat{y} = \text{ChoiceNet}_\theta(D_f, D_t, M_t)$. Given a TLDS of N tuples $\{(D_f, D_t^{(n)}, M_t^{(n)}, y_n)\}_{n=1}^N$, where y_n is the observed post-fine-tuning test accuracy, we train by minimizing the mean-squared error

$$\theta^\star = \arg\min_\theta \ \frac{1}{N} \sum_{n=1}^{N} \left(\text{ChoiceNet}_\theta(D_f, D_t^{(n)}, M_t^{(n)}) - y_n\right)^2,$$

using the Adam optimizer with a learning rate of 10^{-4} for up to 150 epochs, with early stopping (patience of 20 epochs, keeping the best weights).

3.2 Transfer Learning Dataset Creation

We created a novel dataset that we call the Transfer Learning Dataset (TLDS). This dataset consisted of 3-tuples of (transfer learning dataset, transfer learned model, resultant test accuracy).

First, we created a straightforward CNN called the TransferModel, whose hyperparameters are detailed in Figure 2a. The TransferModel was small to ensure fast training, simple to enable

Algorithm 1 ChoiceNet (v2): predict post-fine-tuning test accuracy

Require: Fine-tuning dataset D_f with classes $\mathcal{C}_f$; candidate transfer-learning dataset D_t with classes $\mathcal{C}_t$ and transfer-learned model M_t

Require: Learned parameters $\theta = \{A_f,\, A_t,\, W_1,\, w_2\}$

Fine-tuning dataset embedding:

for each class $c \in \mathcal{C}_f$ **do**

  $e_f^c \leftarrow \dfrac{1}{|D_f^c|} \sum_{x \in D_f^c} g(x)$ ▷ $g(x) \in \mathbb{R}^{2048}$

end for

$E_f \leftarrow$ rows e_f^c for the 10 classes with largest $\|e_f^c\|_1$ ▷ $E_f \in \mathbb{R}^{10 \times 2048}$

Transfer-learning embedding:

for each class $c \in \mathcal{C}_t$ **do**

  $e_t^c \leftarrow \dfrac{1}{|D_t^c|} \sum_{x \in D_t^c} \phi_{M_t}(x)$ ▷ $\phi_{M_t}(x) \in \mathbb{R}^{1152}$

end for

$E_t \leftarrow [\, e_t^1;\ \ldots;\ e_t^{10}\,]$ ▷ $E_t \in \mathbb{R}^{10 \times 1152}$

Learned reduction (LoRA-similar):

$R_f \leftarrow E_f A_f, \qquad R_t \leftarrow E_t A_t$ ▷ $A_f \in \mathbb{R}^{2048 \times 256},\ A_t \in \mathbb{R}^{1152 \times 256}$

$H \leftarrow R_f + R_t$ ▷ $H \in \mathbb{R}^{10 \times 256}$; v1 uses $H = [R_f \,\|\, R_t]$

Prediction head:

$\hat{y} \leftarrow w_2^\top \,\mathrm{Dropout}_{0.3}\big(\mathrm{ReLU}(W_1\,\mathrm{vec}(H))\big)$

return predicted test-set accuracy $\hat{y}$

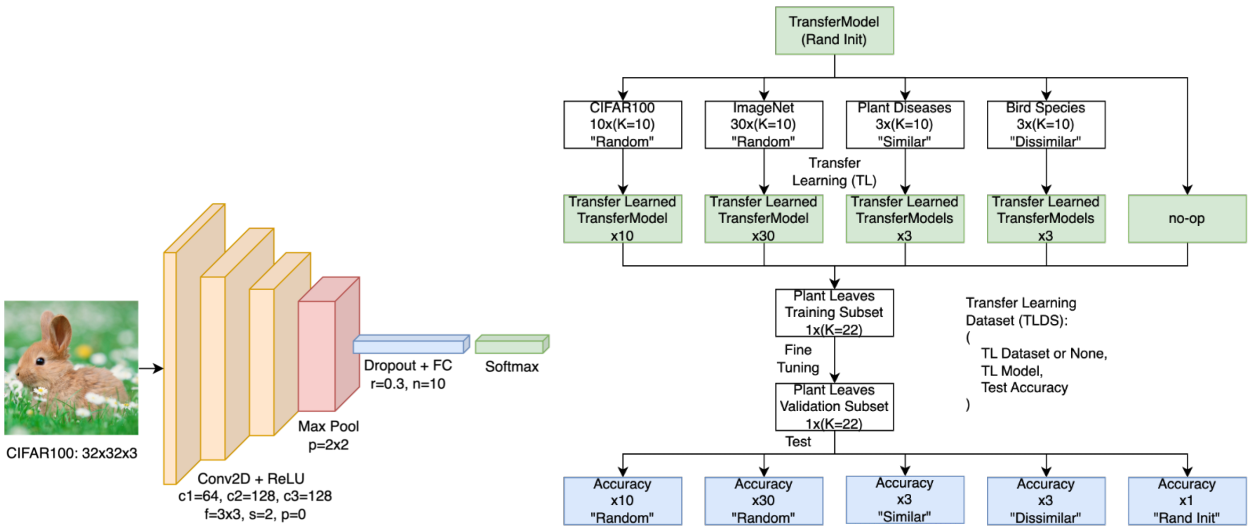
understandable embedding methods, and custom to avoid over-designing for a particular CNN architecture.

We designed the TLDS to have four categories of transfer learning dataset, such that our conclusions were somewhat generalized:

- Similar TL dataset: the TL dataset was of plant diseases,[7] possessing enough classes for three subsets of 10 randomly-sampled classes.
  - Note, our fine-tuning dataset was 22-classes of plant leaves[1] from Kaggle.
- Dissimilar TL dataset: the TL dataset was of bird species,[5] possessing enough classes for three subsets of 10 randomly-sampled classes.
- Random TL datasets: we randomly sampled classes from CIFAR-100[9] and ImageNet[2] to create 40 subsets of 10 classes.
- No TL dataset: the TransferModel's random initialization was considered the transfer-learned model.

We created a script that codified the process detailed in Figure 2b. Each invocation of the script creates 57 data points, and different seeds enable acquisition of more datapoints. Here is the gist:

1. Store the random initialization of the TransferModel, as an experimental control.
2. Train (Adam optimizer with learning rate of 0.001 for 15 epochs) upon each TL dataset.
3. Freeze the TransferModel's first two Conv2D blocks and re-create the head Dense layer. Note the fine-tuning dataset had a different number of classes, standard in transfer learning.



(a) TransferModel Architecture (b) Dataset Generation Process

Figure 2: The Transfer Learning Dataset (TLDS) was created with a custom CNN ("TransferModel") that was repeatedly trained in a full-stack transfer learning, fine-tuning, and testing pipeline.

4. Run fine-tuning (Adam optimizer with learning rate of 0.0001 for 15 epochs) and then test on a validation to attain the test accuracy.
5. Record the transfer learning dataset subset, post-transfer learning weights, and test accuracy.

3.3 Transfer Learning Dataset Choice Network: TLDChoiceNet

The core design of TLDChoiceNet (see all versions in Figure 3) is a neural network:

- Input 1: embedded fine-tuning dataset.
- Input 2: embedded transfer learning dataset, transfer learned model, or both.
- Output: predicted test-time accuracy, if transfer learning upon the TL dataset.

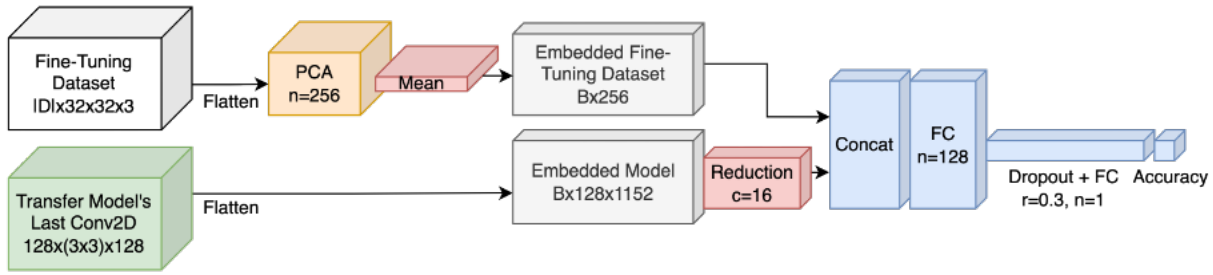
3.3.1 TLDChoiceNet Training Regime

Since ChoiceNet performs regression, we use mean-squared error (MSE) as the loss function. Training was done up to 150 epochs with an Adam optimizer with a learning rate of 0.0001, early-stopping if MSE loss stopped decreasing for 20 epochs (keeping the best weights).

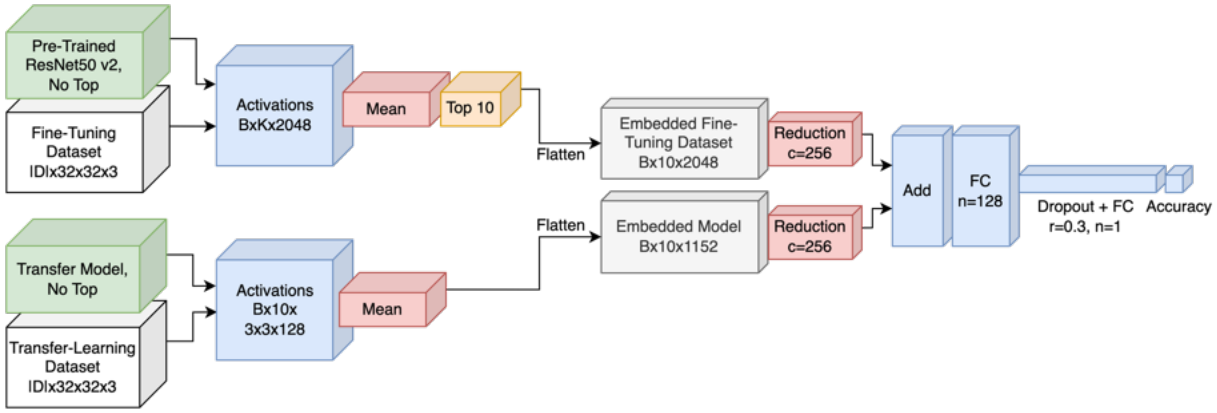
Our TLDS creation process uses 15000 images in the transfer learning dataset and 1000 images in the fine-tuning dataset. We observed a similar TL dataset tends to lead to higher test-set accuracy. The less fine-tuning examples present, the more a similar TL dataset stands out. The more fine-tuning examples, the less effect transfer learning had on test accuracy, which makes sense. We found 1000 fine-tuning images strikes a reasonable balance between the two extremes.

3.3.2 Version 1 Architecture

Our first-pass at ChoiceNet only embedded the transfer-learned model, leaving the transfer learning dataset untouched. It has a simple design (shown in Figure 3a) that provided a baseline performance.



(a) TLDChoiceNet v1 Embeddings and Architecture



(b) TLDChoiceNet v2 Embeddings and Architecture

Figure 3: Each version of TLDChoiceNet builds upon the previous. All figures show the embedding process (left side) and the actual model architecture (right side).

The fine-tuning dataset was embedded in two steps. First was principal component analysis (PCA) on the entire dataset, to reduce each image from a full RGB image to a vector with 256 elements. Next, all examples were averaged to lead to a 256-element vector. The embedding of the transfer-learned model (note: before fine-tuning) was simply each filter in the last Conv2D flattened, which is a matrix of shape $(128, 1152)$.

TLDChoiceNet v1 first performs a LoRA-similar reduction of the embedded TL model. Instead of two trainable matrices (A and B), we just use one trainable A matrix. This learned reduction further reduces the embedded TL model's dimensionality to $(126, 16)$. Next, TLDChoiceNet v1's head concatenates both embeddings and passes them through two fully-connected layers ($n = 128, 1$) with a 30% dropout layer in between.

3.3.3 Version 2 Architecture

The 2nd version of ChoiceNet attempted to solve a major issue: both embedding processes ignored class-specifics. The fine-tuning embedding averaged across all classes, and the transfer learning model's weights general to all classes. This was fixed with two fundamental changes:

- The embedding process was done on a per-class basis.
- Post-embedding reduction was done by a trained aspect of ChoiceNet.

The transfer learning embedding process begins with the transfer-learned TransferModel's last

Conv2D's activations. We chose activations because it combines learned weights (not specific to class) with dataset (all examples). We reduced the dimensionality by averaging activations across each class, resulting in a $(10, 1152)$ matrix. The 10 corresponds with the number of classes and the 1152 is the flattened size of the last Conv2D. One final comment is TLDChoiceNet v2 can't use the randomly initialized model since there's not an accompanying dataset to create activations.

The embedding process of the fine-tuning dataset involved using a ResNet50 v2 pre-trained on ImageNet. We took the model activations before the head layers and averaged on a per-class basis. Then, since the number of classes was greater than 10, we kept the 10 classes with the highest average absolute value activation. This left us with a $(10, 2048)$-sized matrix, where the 2048 comes from the ResNet50 v2's last convolutional layer size.

Now within TLDChoiceNet v2, we used a much larger learned reduction (same LoRA-similar reduction as ChoiceNet v1) to reduce to two $(10, 256)$-sized matrices. We add both together (instead of concatenation) and share the same head design as TLDChoiceNet v1. Overall TLDChoiceNet v2 has nearly four times the number of trainable parameters as version 1.

3.4 Unsupervised Metrics

3.4.1 Distance Metric Using Dataset Distribution

The goal of this experiment was to find metrics which could correlate the performance of transfer learning with fine-tuning task, in an unsupervised fashion. We designed an evaluation metric called distribution distance (DD) to characterize the difference in location and other variabilities between transfer learning and fine-tune datasets.

DD is a combination of mean, skew and kurtosis between a transfer learning dataset's normalized pixel values and a fine-tune dataset's normalized pixel values. For skew and kurtosis we take absolute value, since we wanted to treat left-side vs right-side variability equally. Let us define i as a transfer learning dataset and j as a fine-tuning dataset, with means μ_i, μ_j, standard deviations σ_i, σ_j, and number of samples n, m. Within a given image, each pixel is denoted x_{ik} and x_{jk}. We first define the (absolute) standardized third and fourth moments:

$$s_i = \left| \frac{1}{\sigma_i^3} \frac{1}{n} \sum_{k=1}^{n} (x_{ik} - \mu_i)^3 \right|, \qquad \kappa_i = \left| \frac{1}{\sigma_i^4} \frac{1}{n} \sum_{k=1}^{n} (x_{ik} - \mu_i)^4 \right|,$$

and analogously s_j, κ_j for dataset j. We then define the distance metric $\mathrm{DD}(i, j)$ as the square root of the summed squared errors of mean, absolute skew, and absolute kurtosis:

$$\mathrm{DD}(i, j) = \sqrt{(\mu_i - \mu_j)^2 + (s_i - s_j)^2 + (\kappa_i - \kappa_j)^2}.$$

We then calculate the DD metric for each transfer learning dataset and linearly regress it with test-set accuracy to analyze the correlation.

3.4.2 Correlation Metric Using a Pre-Trained Network

In this experiment, instead of just considering dataset distributions, we utilize ImageNet pre-trained weights to yield low-level statistical information on the dataset.

We do a forward pass through an ImageNet pre-trained headless ResNet50 v2 for all TL dataset images to get the last convolutional layer's output as our embedding latent feature. Similar to ProtoNet's methodology, we calculate class mean embedding as shown in Figure 4. Each TL dataset is now represented by 10 average class activations.

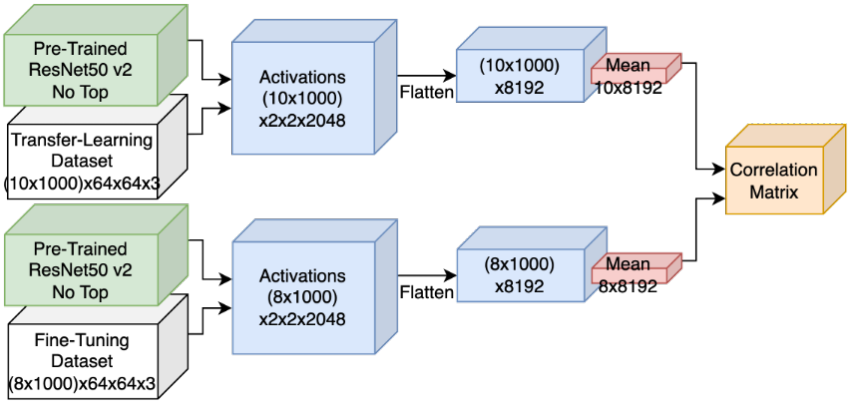


Figure 4: Dataset Embedding via ResNet50 v2

Now, we can define a new metric: average class correlation (ACC). We have dataset i with n classes and dataset j of m classes. Let Cor_{kl} denote the correlation between the per-class embedding of class k in dataset i and class l in dataset j. The average class correlation metric $\text{ACC}(i, j)$ between two datasets is defined as:

$$\text{ACC}(i, j) = \frac{1}{n\,m} \sum_{k=1}^{n} \sum_{l=1}^{m} \text{Cor}_{kl}.$$

We then calculate the ACC metric for each transfer learning dataset and linearly regress it with test-set accuracy to analyze the correlation.

4 Experiments and Results

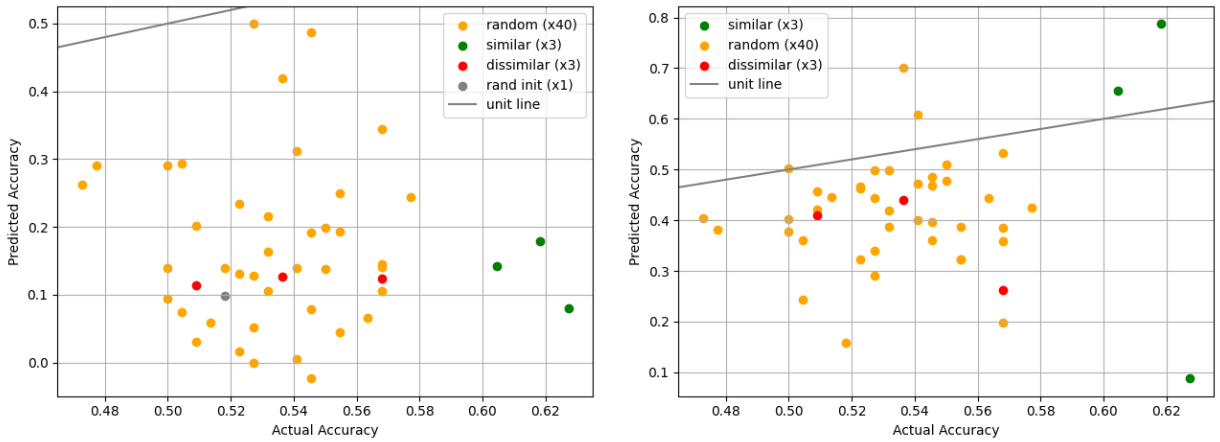
Model / Metric	**Result**
TLDChoiceNet v1 — test MSE	0.154
TLDChoiceNet v2 — test MSE	0.031
Distribution distance (DD) vs. accuracy — R^2	0.894
Average class correlation (ACC) vs. accuracy — R^2	0.974

Table 1: Summary of key quantitative results. TLDChoiceNet v2 reduces test MSE by 5X over v1, and the ACC metric explains fine-tune accuracy better than DD.

4.1 TLDChoiceNet Performance

Shown in Figure 5 we see the results of both TLDChoiceNets. The ideal network's predicted test-set accuracy matches the actual test-set accuracy, which means all points fall on the unit line.

Before discussing predictions, it's useful to comment on the actual test-set accuracy. Firstly, the spread from random initialization to similar transfer learned model is only about 15% accuracy. Ideally this is a much bigger gap, to underscore the relative effects of initialization, transfer learning, and fine tuning. Secondly, the dissimilar TL dataset tended to have roughly the same test-set accuracy as the random datasets, which suggests a larger spread of images may not play as big of a role. To better answer this question, we could redesign the TransferModel or utilize a more challenging test dataset to expose any nuances present.



(a) TLDChoiceNet v1 Performance

(b) TLDChoiceNet v2 Performance

Figure 5: TLDChoiceNet v1 attained a MSE loss of 0.154 and v2 attained a MSE loss of 0.031 on test datapoints. Note the test data shared the fine-tuning dataset with the training data.

Not only does TLDChoiceNet v2 have an upgraded architecture, but it has 1,147,137 trainable parameters compared with v1's 313,601 trainable parameters (nearly 4X increase). At training-time (94 datapoints), both TLDChoiceNet v1 and v2 attained a MSE loss below 0.02. However, at test time, the difference stands out: v1's MSE loss of 0.154 decreases to 0.031 with v2 ($1/5^{\text{th}}$ the size). In Figure 5, the test datapoints comparing actual accuracy with predicted accuracy are plotted. It's clear that both versions tend to underestimate test-set accuracy, but v2 is vastly closer to the unit line.

MSE numbers listed above were generated from a test subset that shared the same fine-tuning dataset as the training subset. When testing with both an unseen transfer learning and fine-tuning dataset, TLDChoiceNet v1 does much worse with a MSE of 0.46. Furthermore, test-set accuracy predictions were off by as much as 80%. To handle this removed constraint (shared fine-tuning dataset), we either need a more advanced version of TLDChoiceNet or a more general training set with multiple fine-tuning datasets included. This outstanding item is also documented in section 5.

Even though there is still room for improvement in a v3 network, the decrease in MSE loss by a factor of 5 at test-time from v1 to v2 is a substantial gain.

4.2 Low-Level Statistics and Metric Correlation

By analyzing the histogram of a normalized dataset's pixel distribution, there exists a distribution shift between dissimilar TL dataset classes and fine-tuning dataset classes. For similar TL dataset classes, the low-level statistics are more similar to fine-tune dataset, as shown in Figure 6 (Appendix).

After fitting DD with the test-set accuracy, it can explain the variation in accuracy with a R2 of 0.894 (see Figure 7 in the Appendix). This shows a strong correlation between the TL dataset's pixel distribution and fine-tune performance. A lower DD metric thus indicates better test-set accuracy after fine-tuning. Per-dataset DD values are reported in Table 2 (Appendix).

After embedding TL dataset and fine-tune dataset with a pre-trained ResNet50 v2, we found for 3 groups of similar TL datasets, the ACC metric is 0.501 (see Figure 8 in the Appendix). On the other hand, for dissimilar TL datasets the ACC metric is only 0.276. This is 45% lower. A higher ACC metric could indicate better test-set accuracy after transfer learning.

As illustrated in Figure 9 (Appendix), after fitting the correlation metric ACC with the test fine-tune accuracy, it can explain the variation in fine-tune accuracy with a R2 of 0.974. Thus we can say ACC can better explain the variation on test-set accuracy compared to DD, which only uses dataset distribution. Per-dataset ACC values are reported in Table 3 (Appendix).

To gain further understanding, we got rid of the pre-trained ResNet50 v2 weights, instead beginning with class means of the dataset's normalized pixels. We used this to calculate ACC in Figure 8. The average ACC for similar classes is 0.507 and for dissimilar classes it's 0.438, so the difference between similar and dissimilar ACC is 0.06. However, by adding the ResNet50 v2 pre-trained weights, the average ACC for similar classes is 0.49 and for dissimilar classes it's 0.27. We observe the difference between similar and dissimilar ACC has increased to 0.22. To summarize this finding: pre-trained weights are able to embed dissimilar class information further apart in latent feature space, providing increased feature separation for use in classification.

4.3 Pre-Trained Weights Across Fine-Tuning

To understand how pre-trained weights change across fine-tuning, we measured L2 norm between pre-trained weights and fine-tune weights across all fine-tuning epochs. Shown in Figure 10 (Appendix), L2 distance changes as a function of the number of epochs at fine-tune time. For a similar TL dataset's pre-trained weights, the L2 distance changes less than a dissimilar TL dataset's pre-trained weights. This could be due to more shared lower-level features for a similar TL dataset's pre-trained weights compared with a dissimilar TL dataset's weights.

Centered kernel alignment (CKA) is a distance metric invariant to linear transformation and also presents useful information about layers between different models. We also calculated the CKA as another measure to analyze the feature-space similarity between the pre-trained weights and fine-tuned weights. These results again show that for similar TL dataset, there's more correlation between weights after fine-tuning compared with a dissimilar TL dataset.

5 Conclusion and Future Work

In conclusion, we detail TLDChoiceNet v1 and v2, two networks featuring a novel dataset embedding and reduction pipeline, spanning from thousands of examples across many classes to one number: predicted test-set accuracy. We also designed two metrics, distribution distance (DD) and average class correlation (ACC), that can determine the optimal choice of transfer learning dataset. DD quantifies the distribution shift between two datasets and ACC proposes correlation between activations can be used as a high-quality substitute for test-set accuracy.

Several directions remain for future work:

1. One useful activity would be scrutinizing the inner workings of ChoiceNet. We could answer questions like "do pre-trained weights, pre-training dataset, or fine-tune dataset play the largest role?" or "what aspects of a dataset matter most for test-set accuracy?" TLDChoiceNet v2 addressed known issues from v1, but it also grew 4X in size. Perhaps TLDChoiceNet v3 could be vastly smaller but maintain the same performance.
2. Another area of investigation could be giving ChoiceNet more information on the fine-tuning process. Perhaps if the fine-tuning is short one pre-trained model is the right choice, but if there's many training epochs or examples, then a different model is the right choice.
3. We also need to test the generalizability of ChoiceNet to other domains and tasks. Our problem domain was image classification, so it would be good to test these conclusions in domains without image data or tasks besides classification. Similarly, we could test with multiple fine-tuning datasets, not just plant leaves.
4. Analyze and test the ability of other ImageNet pre-trained architectures such as Inception v3 and EfficientNet in further pushing apart feature-space embeddings.

Acknowledgements

We thank Chelsea Finn and Daniel Zeng for helpful discussions and feedback, including ideas on ImageNet embedding and measuring weight distance across training.

References

[1] csafrit. Plant leaves for image classification. `https://www.kaggle.com/datasets/csafrit2/plant-leaves-for-image-classification`. [Version 2].

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[3] Christopher Fifty, Ehsan Amid, Zhe Zhao, Tianhe Yu, Rohan Anil, and Chelsea Finn. Efficiently identifying task groupings for multi-task learning. *CoRR*, abs/2109.04617, 2021.

[4] Weifeng Ge and Yizhou Yu. Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning. *CoRR*, abs/1702.08690, 2017.

[5] Gerry. Birds 450 species- image classification. `https://www.kaggle.com/datasets/gpiosenka/100-bird-species`. [Version 59].

[6] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. *CoRR*, abs/1901.09960, 2019.

[7] David P. Hughes and Marcel Salath'e . An open access repository of images on plant health to enable the development of mobile disease diagnostics through machine learning and crowdsourcing. *CoRR*, abs/1511.08060, 2015.

[8] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey E. Hinton. Similarity of neural network representations revisited. *CoRR*, abs/1905.00414, 2019.

[9] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[10] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009.

[11] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution, 2022.

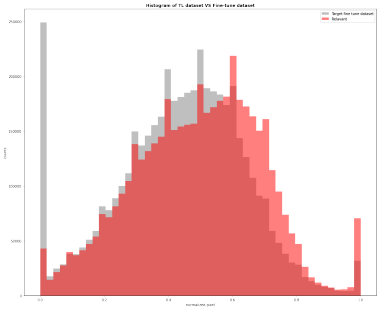
[12] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[13] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. What is being transferred in transfer learning? *CoRR*, abs/2008.11687, 2020.

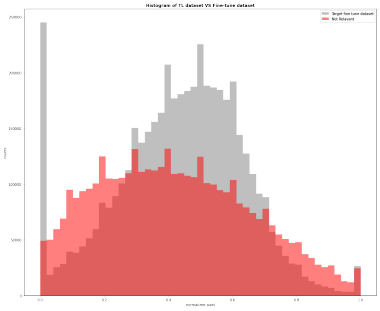
[14] Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. *CoRR*, abs/1903.05987, 2019.

A Additional Figures and Results

A.1 Dataset Distribution: Low-Level Statistics



(a) Similar TL Dataset vs Fine-Tune Dataset



(b) Dissimilar TL Dataset vs Fine-Tune Dataset

Figure 6: Histogram of Normalized Pixel Values.

TL Dataset	Distance Metric (DD)	Fine-tune Accuracy
Similar – Model1	0.375914939	89.0%
Similar – Model2	0.231030915	84.5%
Similar – Model3	0.328677488	84.5%
Dissimilar – Model4	0.687387568	15.5%
Dissimilar – Model5	0.584759946	18.5%
Dissimilar – Model6	0.653659486	20.3%
Dissimilar – Model7	0.540906724	25.4%
Dissimilar – Model8	0.612594293	20.3%
Dissimilar – Model9	0.611245781	22.4%

Table 2: Distance Metric DD vs. fine-tune accuracy per TL dataset.

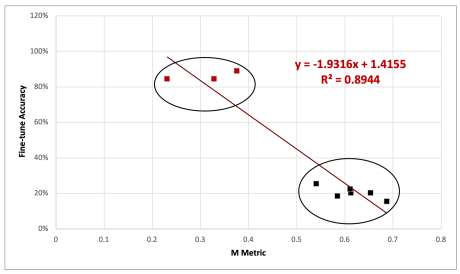


Figure 7: Distance Metric DD's Fitting Result.

A.2 Activation Map Correlations

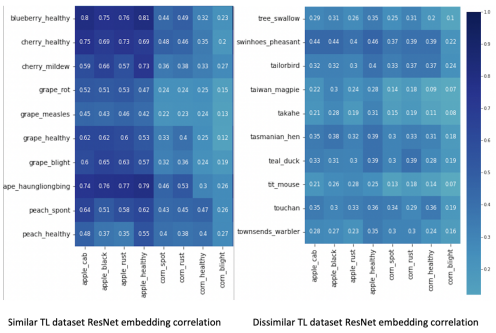
A.3 Pre-Trained Weights Across Fine-Tuning

B Source Code

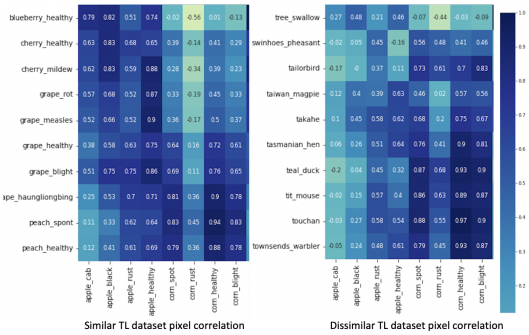
All source code for the project can be found on GitHub at CDC1688/TLDChoiceNet.

TL Dataset	Correlation Metric (ACC)	Fine-tune Accuracy
Similar – Model1	0.500714286	88.99%
Similar – Model2	0.5015	84.50%
Similar – Model3	0.49025	84.50%
Dissimilar – Model4	0.262	15.50%
Dissimilar – Model5	0.2975	18.50%
Dissimilar – Model6	0.2865	20.30%
Dissimilar – Model7	0.276125	25.40%
Dissimilar – Model8	0.252625	20.30%
Dissimilar – Model9	0.251625	22.43%

Table 3: Correlation Metric ACC vs. fine-tune accuracy per TL dataset.



(a) ResNet50 v2 Embedding's Correlation Matrix

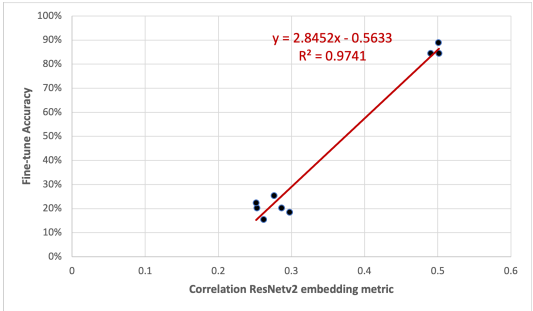


(b) Correlation Matrix of Dataset Pixels

Figure 8: Correlation Metric ACC vs Test-Set Accuracy.

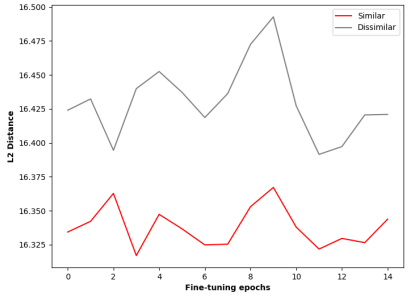
C Author Contributions

J.N. designed ChoiceNet v1 and built the transfer-learning and fine-tuning training pipeline used to train it quickly. J.N. also performed the experimental analyses: dataset pixel-distribution analysis, activation maps from a pre-trained ResNet50 v2, the class-correlation metric, monitoring of pre-trained weight-distribution changes during fine-tuning, and the CKA comparison. J.N. created the distribution distance (DD) and average class correlation (ACC) metrics and used them to fit linear regressions against fine-tune accuracy. J.D.B. created functions to load, preprocess, and automatically select subsets of all datasets, and built a comprehensive script to create the TLDS as a TensorFlow Sequence handling the similar, dissimilar, random-initialization, and random-class

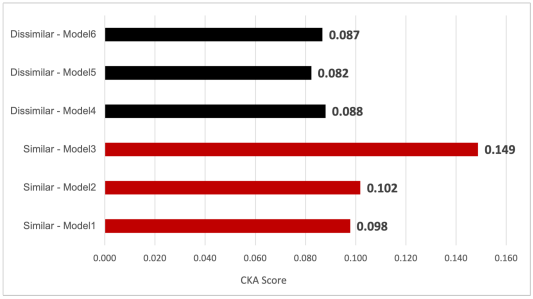


(a) Correlation Metric ACC's Fitting Result

Figure 9: Correlation Embedding Metric ACC.



(a) Pre-Trained L2 Distance During Fine-Tuning



(b) Centered Kernel Alignment Metric

Figure 10: Weight change and CKA.

cases, re-runnable to generate more datapoints per invocation. J.D.B. designed the ChoiceNet v2 embeddings and architecture, building upon the ChoiceNet v1 work, and created scripts to connect ChoiceNet v1 and v2 training with the TLDS.