Supplementary for Task2Sim: Towards Effective Pre-training and Transfer from Synthetic Data

Samarth Mishra	extsuperscript{1} Rameswar Panda	extsuperscript{2} Cheng Perng Phoo	extsuperscript{1,3} Chun-Fu (Richard) Chen	extsuperscript{2} Leonid Karlinsky	extsuperscript{2} Kate Saenko	extsuperscript{1,2} Venkatesh Saligrama	extsuperscript{1} Rogerio S. Feris	extsuperscript{2}

\textsuperscript{1}Boston University \textsuperscript{2}MIT-IBM Watson AI Lab \textsuperscript{3}Cornell University

A. Task2Vec

We used Task2Vec \cite{1} representations for downstream tasks for our Task2Sim model. Task2Vec of a task consists of diagonal elements of the Fisher information matrix (FIM) of the outputs with respect to the network parameters over the distribution of downstream task examples (Refer to Sec. 2 of \cite{1} for more details). For this purpose, following \cite{1}, we used a single Imagenet pre-trained probe network with only the classifier layer trained on specific tasks (using the training set of examples for that task). In our experiments, a Resnet-18 probe network was used, resulting in a 9600-dimensional Task2Vec task representation.

How much downstream data do we need access to? In the case of models pre-trained using an approach that is not task-adaptive, there is no need to access any downstream data while pre-training. Given that task-adaptive approaches need a downstream task representation, in the main paper, we used Task2Vec. Here, following \cite{1} we used all labeled examples from the training set of the downstream task to represent its distribution (in computing the FIM). However, we show that the FIM can be estimated by using fewer examples from the downstream task and the resulting Task2Vec vectors can be used to train Task2Sim with no degradation in performance (see Table 1). This property also makes Task2Sim more practical since a user need not wait to collect labels for all data pertaining to their downstream application in order to generate pre-training data using Task2Sim.

B. Similarity between Learned Features

We used centered kernel alignment (CKA) \cite{13} to find the similarity between features learned by the Resnet-50 backbone pre-trained on different image sets containing 100k images from 237 classes. Figure 1, shows these similarities computed using the output features at different stages of the backbone (Stages 1-4 are intermediate outputs after different convolutional blocks in the resnet).

A few interesting phenomena surface: Task2Sim features (\textit{i.e.} features produced by a model pre-trained on Task2Sim generated dataset) are more similar to Imagenet features, than Domain Randomization. Thus Task2Sim in some manner, mimics features learned on real images better.

We can also see that features early on in the network are largely similar across all kinds of pre-training and they only start differentiating at later stages, suggesting high similarity in lower level features (\textit{e.g.} edges, curves, textures, \textit{etc.}) across different pre-training datasets. Also, as might be expected, features post downstream finetuning are more similar to each other than before, while still quite far away from being identical.

C. Additional Results

C.1. Effect of Different Backbones

In Figures 2 and 3, we show the average downstream performance over the seen and unseen tasks respectively, using different Resnet backbones (of different sizes). For this study, we used the same pre-training procedure across all backbones. We see that results are largely consistent
that Task2Sim’s features are more similar to ImageNet than those produced by pre-training with Domain Randomization.

Figure 1. CKA similarities between features from backbones trained on different pre-training datasets (with 100k images from 237 classes). Similarities have been computed using features output at different stages of the Resnet-50 model. We notice that features at earlier stages across all methods of pre-training are quite similar and only later in the Resnet, do they start differentiating. We also observe that Task2Sim’s features are more similar to ImageNet than those produced by pre-training with Domain Randomization.
with different backbones and for all of them Task2Sim performance is competitive with Imagenet pre-training and is much better than Domain Randomization. We also see that typically methods improve average downstream performance with the use of a larger backbone in the classifier. Moving from Resnet-50 to Resnet-101, Task2Sim performance breaks this trend and is lower indicating that the larger backbone could overfit in this case. This might be expected since Task2Sim was trained to optimize the performance of a Resnet-50 backbone.

C.2. Task2Sim Results—Linear Probing

Figure 4 shows the downstream accuracy with linear probing for different seen and unseen datasets where pre-training dataset has 100k images from all 237 classes. These complement Figures 3 and 4 in the main paper, where downstream evaluation used full network finetuning.

C.3. Varying Pre-training Data Size

Linear Probing. Figures 5, 6 and 7 are counterparts (with downstream evaluation done with linear probing) of Figures 6, 7 and 8 in the main paper respectively. We see that primarily similar findings as the main paper hold and in Figure 5, different backbones improve at a similar rate with more classes (and images for pre-training). In Figure 6, we see that both methods of synthetic pre-training improve their features with more object models, with Domain Randomization improving at a slightly higher rate.

In Figure 7 we see some differences: There is a more severe saturating behavior of downstream performance, which even decreases by a little after a certain point for the synthetic pre-training data. This is likely because the feature extractor overfits to the pre-training task and a linear classifier on these features cannot perform as well. Both from Figure 5 and from the curve for Imagenet-1K in Figure 7 we see that this saturating/overfitting behavior is somewhat alleviated by more classes in pre-training data. Another observation of note in Figure 7 is that the feature extractor pre-trained on Domain Randomization starts overfitting before it matches the performance of Task2Sim. With Fig 8 in the main paper, we mentioned that with more images a non-adaptive approach like domain randomization could improve its performance faster and sometimes equal a task-adaptive approach like Task2Sim. Figure 7 indicates that although a non-adaptive approach may improve faster, it may not always match the performance of its adaptive counterpart.

Unseen Tasks. Figures 9, 10 and 11 show the effect of the above variations averaged over unseen tasks. We can see that similar trends hold in this case, as in the case of seen datasets.
Figure 4. Performance of Task2Sim vs baselines on 12 seen tasks and 8 unseen tasks for 237 class / 100k image pre-training datasets evaluated with linear probing. Best viewed in color.

Figure 5. Avg performance with linear probing over 12 seen tasks at different number of classes for pre-training. All methods improve performance at similar rates with the addition of more classes.

Figure 6. Avg performance with linear probing over 12 seen tasks at different number of object meshes used per category for generating synthetic pretraining data. Both methods of synthetic data generation improve performance with addition of more objects with Domain Randomization improving at a slightly higher rate.

Figure 7. Task2Sim performance (avg over 12 seen tasks) vs other methods using linear probing for evaluation at different number of images for pretraining. Task2Sim is highly effective at fewer images. Increasing the number of images improves performance for all methods. Towards higher number of images in the case of linear probing we see methods not only reach a saturation but also exhibit some overfitting to pre-training data. Also, Domain Randomization stops improving in this case (evaluation with linear probing) before it can match Task2Sim performance.

C.4. Comparison with very large scale pretraining (CLIP).

CLIP [22] pre-trains on 400M image-text pairs. Such large datasets when curated from the web, are bound to have privacy and other ethical concerns, as discussed in the main paper. CLIP pre-training is also much more expensive than its counterparts using our synthetic data. We conducted an experiment finetuning a Resnet-50 model using pre-trained weights from CLIP on our tasks, while noting that this CLIP pre-trained Resnet-50 is different from the standard model used by us and uses more parameters (38M in CLIP Resnet50 vs 25M in standard Resnet50). The result was 77.33% avg. accuracy on seen tasks and 91.56% avg. accuracy on unseen tasks, which is comparable to the
best Task2Sim performance (79.10% over seen tasks and 91.50% over unseen tasks).

D. Synthetic Image Generation

We used Three-D-World (TDW) [7] for synthetic image generation. It is a platform built using the Unity3D engine, and besides a python interface, provides asset bundles which include 3D object models, interactive 3D scenes, and HDRi skyboxes (360° images of real scenes accompanied with lighting information). TDW is available under a BSD 2-Clause "Simplified" License.

For our implementation, we used all 2322 object models from 237 different classes available in TDW. We use a generator that imports one object into a simple scene with an HDRI-skybox background. It then, changes different properties of the scene/object/camera based on 8 simulation parameters as mentioned in Section 4.1 of the main paper. Whenever different variations corresponding to a simulation parameter are to be included, values are chosen uniformly at random within an appropriate range (via a careful choice of the extremes). Figure 8 has 8 rows corresponding to each of the simulation parameters used for Task2Sim. Each row shows using 5 images, the variations corresponding to its specific simulation parameter.

Generating 1M images using our generator with all 2322 objects, takes around 12 hours on an Nvidia Tesla-V100 GPU. Given the number of objects we used in our implementation, a bottleneck in image generation is the speed of loading object meshes into Unity3D. Hence, we used a subset of 780 objects from 100 classes with relatively simpler meshes, for generating the data used for training Task2Sim. The 8 parameters we used result in a total of \(2^8 = 256\) different possibilities and so we pre-generated these 256 sets of 40k images each for faster and smoother training of the Task2Sim model. Each of these 256 sets took \(\sim\)30 mins to generate on a Tesla-V100 GPU.

E. Training and Evaluation

We based our implementation of different classifiers for pre-training and downstream evaluation on pytorch-image-models [27]. For all experiments except those in Appendix C.1, we used a Resnet-50 backbone for our classifier. For all datasets while pre-training, we used the following parameters : we trained for 100 epochs using an AdamW optimizer, using a learning rate 0.001 and a batch size of 1024. The learning rate used a linear warmup for 20 epochs and a cosine annealing schedule following warmup. We use regularization methods like label-smoothing, cutmix [29] and mixup [30] following a training strategy from [27]. We used image augmentation in the form of RandAugment [6] while pre-training.

For downstream evaluation, we followed a procedure similar to [11]. For both evaluations using linear probing and full-network finetuning, we used 50 epochs of training using an SGD optimizer with learning rate decayed by a tenth at 25 and 37 epochs. No additional regularizers or data augmentation approaches were used. For each downstream task, we did a coarse hyperparameter grid-search over learning rate \(\in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}\), optimizer weight decay \(\in \{0, 10^{-5}\}\) and training batch size \(\in \{32, 128\}\). We found by comparing backbones pre-trained on Imagenet and a large synthetic set generated with Domain Randomization, that with the above grid, for each specific downstream task and evaluation method, a particular set of hyperparameters worked best irrespective of the pre-training data. This was found using a separate validation split created from the downstream training set with 30% of the examples. Given this finding, we fixed these hyperparameters for a given downstream task and evaluation method for all remaining experiments.

F. Details of Downstream Tasks

Table 2 shows the number of classes in each of the 20 downstream tasks we used. It also shows the number of images in the training and test splits for each.

G. Limitations

In this paper, we constrained our demonstration to a relatively low number of datasets and simulation parameters, limited by data generation, pre-training and evaluation speed. If these processes can be made more efficient, in

<table>
<thead>
<tr>
<th>Category</th>
<th>Dataset</th>
<th>Train Size</th>
<th>Test Size</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural</td>
<td>CropDisease [18]</td>
<td>43456</td>
<td>10849</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>Flowers [20]</td>
<td>1020</td>
<td>6149</td>
<td>102</td>
</tr>
<tr>
<td></td>
<td>DeepWeeds [21]</td>
<td>12252</td>
<td>5257</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>CUB [24]</td>
<td>5994</td>
<td>5794</td>
<td>200</td>
</tr>
<tr>
<td>Satellite</td>
<td>EuroSAT [8]</td>
<td>18900</td>
<td>8100</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Resisc45 [2]</td>
<td>22005</td>
<td>9495</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>AID [28]</td>
<td>6993</td>
<td>3007</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>CactuAerial [16]</td>
<td>17500</td>
<td>4000</td>
<td>2</td>
</tr>
<tr>
<td>Symbolic</td>
<td>Omniplot [14]</td>
<td>9226</td>
<td>3954</td>
<td>1623</td>
</tr>
<tr>
<td></td>
<td>SVHN [19]</td>
<td>73257</td>
<td>26032</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>USPS [10]</td>
<td>7291</td>
<td>2007</td>
<td>10</td>
</tr>
<tr>
<td>Medical</td>
<td>ISIC [5]</td>
<td>7007</td>
<td>3008</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>ChestX [26]</td>
<td>18090</td>
<td>7758</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>ChestXPneumonia [12]</td>
<td>5216</td>
<td>624</td>
<td>2</td>
</tr>
<tr>
<td>Illustrative</td>
<td>Kaokore [23]</td>
<td>6568</td>
<td>821</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Sketch [25]</td>
<td>35000</td>
<td>15889</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>PACS-C [15]</td>
<td>2107</td>
<td>237</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>PACS-S [15]</td>
<td>3531</td>
<td>398</td>
<td>7</td>
</tr>
<tr>
<td>Texture</td>
<td>DTD [4]</td>
<td>3760</td>
<td>1880</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>FMD [31]</td>
<td>1400</td>
<td>600</td>
<td>10</td>
</tr>
</tbody>
</table>
Figure 8. Examples of variations using different simulation parameters. Best viewed in color and under zoom.
Figure 9. Downstream performance (avg over 8 unseen tasks) with different number of classes for pre-training. Best viewed in color.

Figure 10. Downstream performance (avg over 8 unseen tasks) with different number of objects for pre-training. Best viewed in color.

future work, we can expect to use more simulation parameters (with possibly more discrete options or even real-valued ranges), and use more datasets for training Task2Sim, allowing it to be more effective in deployment as a practical application.

H. Societal Impact

In the introduction, we discussed model pre-training using large real image datasets was what paved the way for a gamut of transfer learning research. Using real images is however riddled with curation costs and others concerns around privacy, copyright, ethical usage, etc. The fact that downstream performance on average correlates positively with the size of pre-training data, created a race for curating bigger datasets. Corporations with large resources are able to invest in such large-scale curation and create datasets for their exclusive use (e.g. JFT-300M [3, 9], or Instagram-3.5B [17]), which are unavailable to a range of research on downstream applications.

Using synthetic data for pre-training can drastically reduce these costs, because potentially infinite images can be rendered once 3D models and scenes are available, by varying various simulation parameters. In this paper, we demonstrated that the optimal use of such a simulation engine can be found in restricting certain variations, and that different restrictions benefit different downstream tasks. Our Task2Sim approach, can be used as the basis for a pre-training data generator, which as an end-user application can allow research on a wide range of downstream applications to have access to the benefits of pre-training on large-scale scale data. This does not create any direct impacts on average individuals, but could do so through the advancement in downstream applications. One particular case, as an example, could be the advancement in visual recognition systems in the medical domain, possibly making the diagnosis of illnesses faster and cheaper.

References

Figure 11. Downstream performance (avg over 8 unseen tasks) with different number of images for pre-training. Best viewed in color.

[22] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning...


