PAUL AGGARWAL

Big Transfer

QUESTION 1:

Transfer learning offers a solution that requires a large amount of task-specific data and compute which these per-task requirements can make new tasks prohibitively expensive. The transfer learning solution is that task-specific data and compute are replaced with a pre-training phase. A network is trained once on a large, generic dataset, and its weights are then used to initialize subsequent tasks which can be solved with fewer data points, and less compute. We revisit a simple paradigm: pre-train on a large supervised source dataset, and then fine-tune the weights on the target task. This eventually provides foundation for the recipe that uses the minimal number of tricks yet attains excellent performance on many tasks. We call this recipe “Big Transfer” (BiT).

With the effectiveness of scale (during pre-training) in the context of transfer learning, including transfer to tasks with very few datapoints, we can that the combination of GN and WS is useful for training with large batch sizes, and has a significant impact on transfer learning.

One can add a step at which the trained model is fine-tuned to the test resolution. The latter is well-suited for transfer learning; we include the resolution change during our fine-tuning step.

When we revisit classical transfer learning, where a large pre-trained generalist model is fine-tuned to downstream tasks of interest. A simple recipe is provided which exploits large scale pre-training to yield good performance on all of these tasks. BiT uses a clean training and fine-tuning setup, with a small number of carefully selected components, to balance complexity and performance.

When transfer to Downstream, we avoid expensive hyperparameter search for every new task and dataset size; we try only one hyperparameter per task. We use a heuristic rule—which we call BiT-HyperRule—to select the most important hyperparameters for tuning as a simple function of the task’s intrinsic image resolution and number of datapoints. We found it important to set the following hyperparameters per-task: training schedule length, resolution, and whether to use MixUp regularization. We use BiT-HyperRule for over 20 tasks, with training sets ranging from 1 example per class to over 1M total examples. (Refer to 1912.11370.pdf Section 3.3 for BiT-HyperRule settings). During fine-tuning, the following standard data pre-processing: resize the image to a square, crop out a smaller random square, and randomly horizontally flip the image at training time. At test time, only resize the image to a fixed size. In some tasks horizontal flipping or cropping destroys the label semantics, making the task impossible. An example is if the label requires predicting object orientation or coordinates in pixel space. In these cases omit flipping or cropping when appropriate. Recent work has shown that existing augmentation methods introduce inconsistency between training and test resolutions for CNNs. Therefore, it is common to scale up the resolution by a small factor at test time. As an alternative, one can add a step at which the trained model is fine-tuned to the test resolution. The latter is well-suited for transfer learning; we include the resolution change during our fine-tuning step. It is found that MixUp is not useful for pre-training BiT, likely due to the abundance of data. However, it is sometimes useful for transfer. It is most useful for mid-sized datasets, and not for few-shot transfer (Refer to 1912.11370.pdf Section 3.3). Surprisingly, we do not use any of the following forms of regularization during downstream tuning: weight decay to zero, weight decay to initial parameters, or

dropout. Despite the fact that the network is very large—BiT has 928 million parameters—the performance is surprisingly good without these techniques and their respective hyperparameters, even when transferring to very small datasets. We find that setting an appropriate schedule length, i.e. training longer for larger datasets, provides sufficient regularization.

What is Transfer Learning?

Keeping in mind that in practice, very few people train an entire Convolutional Network from scratch. In transfer learning we first train a base network on a base dataset and task, and then we repurpose the learned features, or transfer them, to a second target network to be trained on a target dataset and task. This process will tend to work if the features are general, that is, suitable to both base and target tasks, instead of being specific to the base task.

Transfer learning is the machine learning challenge of using information (or representations) learned on one task to aid learning on another task. For transfer learning to work, the two tasks should be from related domains. Image processing is an example of a domain where transfer learning is often used to speed up the training of models across different tasks. Transfer learning is appropriate for image processing tasks because low-level visual features, such as edges, are relatively stable and useful across nearly all visual categories. Furthermore, the fact that CNN models learn a hierarchy of visual feature, with the early layers in CNN learning functions that detect these low-level visual features in the input, makes it possible to repurpose the early layers of pretrained CNNs across multiple image processing projects. For example, imagine a scenario where a project requires an image classification model that can identify objects from specialized The Future of Deep Learning 237 categories for which there are no samples in general image datasets, such as ImageNet. Rather than training a new CNN model from scratch, it is now relatively standard to first download a state-of-the-art model (such as the Microsoft ResNet model) that has been trained on ImageNet, then replace the later layers of the model with a new set of layers, and finally to train this new hybrid-model on a relatively small dataset that has been labeled with the appropriate categories for the project. The later layers of the state-of-the-art (general) model are replaced because these layers contain the functions that combine the low level features into the task specific categories the model was originally trained to identify. The fact that the early layers of the model have already been trained to identify the low-level visual features speeds up the training and reduces the amount of data needed to train the new project specific model. The increased interest in unsupervised learning, generative models, and transfer learning can all be understood as a response to the challenge of annotating increasingly large datasets. (ref. Deep Learning by John D. Kelleher)

ConvNet features are more generic in the early layers and more original-dataset specific in the later layers, here are some common rules of thumb for navigating the four major scenarios:

Depending on both the size of the new dataset and the similarity of the new dataset to the original dataset, the approach for using transfer learning will be different.

1. The *target* dataset is **small** and **similar** to the *base* training dataset.
   Since the target dataset is small, it is not a good idea to fine-tune the ConvNet due to the risk of overfitting. Since the *target* data is similar to the *base* data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, we:
   - Remove the fully connected layers near the end of the pretrained *base* ConvNet

- Add a new fully connected layer that matches the number of classes in the *target* dataset
- Randomize the weights of the new fully connected layer and freeze all the weights from the pre-trained network
- Train the network to update the weights of the new fully connected layers

2. The *target* dataset is **large** and **similar** to the *base* training dataset.
   Since the *target* dataset is large, we have more confidence that we won\u2019t overfit if we try to fine-tune through the full network. Therefore, we:
   - Remove the last fully connected layer and replace with the layer matching the number of classes in the *target* dataset
   - Randomly initialize the weights in the new fully connected layer
   - Initialize the rest of the weights using the pre-trained weights, i.e., unfreeze the layers of the pre-trained network
   - Retrain the entire neural network
3. The *target* dataset is **small** and **different** from the *base* training dataset.
   Since the data is small, overfitting is a concern. Hence, we train only the linear layers. But as the *target* dataset is very different from the *base* dataset, the higher level features in the ConvNet would not be of any relevance to the *target* dataset. So, the new network will only use the lower level features of the *base* ConvNet. To implement this scheme, we:
   - Remove most of the pre-trained layers near the beginning of the ConvNet
   - Add to the remaining pre-trained layers new fully connected layers that match the number of classes in the new dataset
   - Randomize the weights of the new fully connected layers and freeze all the weights from the pre-trained network
   - Train the network to update the weights of the new fully connected layers
4. The *target* dataset is large and different from the *base* training dataset.
   As the *target* dataset is large and different from the *base* dataset, we can train the ConvNet from scratch. However, in practice, it is beneficial to initialize the weights from the pre-trained network and fine-tune them as it might make the training faster. In this condition, the implementation is the same as in case 3.

(ref. https://www.hackerearth.com/practice/machine-learning/transfer-learning/transfer-learning-intro/tutorial/)