Transfer learning

Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

Why use transfer learning?

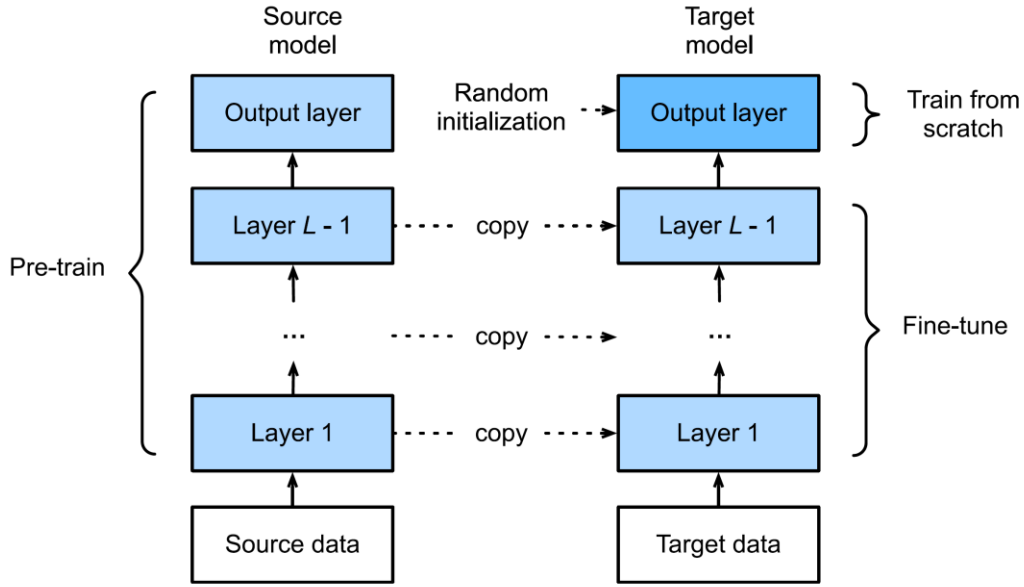
- Having plenty of unlabeled data and little labeled data is common. Building a large unlabeled dataset is often cheap (e.g., a simple script can download millions of images off the internet), but labeling those images (e.g., classifying them as cute or not) can usually be done reliably only by humans. Labeling instances is time-consuming and costly, so it’s normal to have only a few thousand human labeled instances

Transfer learning

- It is generally not a good idea to train a very large DNN from scratch: instead, you should always try to find an existing neural network that accomplishes a *similar task* to the one you are trying to tackle, then reuse the lower layers of this network. This technique is called *transfer learning*
  - It will not only speed up training considerably, but also require significantly less training data
- Suppose you have access to a DNN that was trained to classify pictures into 100 different categories, including animals, plants, vehicles, and everyday objects. You now want to train a DNN to classify specific types of vehicles. These tasks are very similar, even partly overlapping, so you should try to reuse parts of the first network

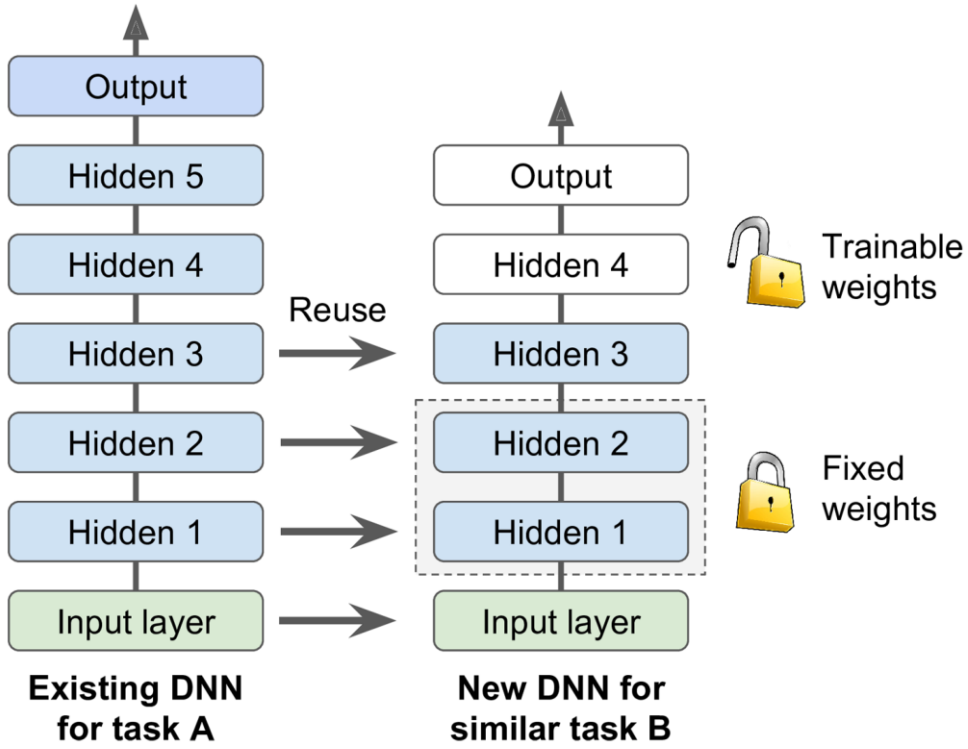
Transfer learning

- Transfer learning work best when the inputs have similar low-level features
  - The output layer of the original model should be replaced because it is most likely not useful for the new task, and it may not have the right number of outputs for the new task
  - Similarly, the upper hidden layers of the model are less likely to be as useful as the lower layers, since the high-level features that are most useful for the new task may differ significantly from the ones that were most useful for the original task
  - If we treat lower layer frozen we can often speed up training while still obtain good performance for vision or NLP tasks
    - There exist complex variants such as adapter



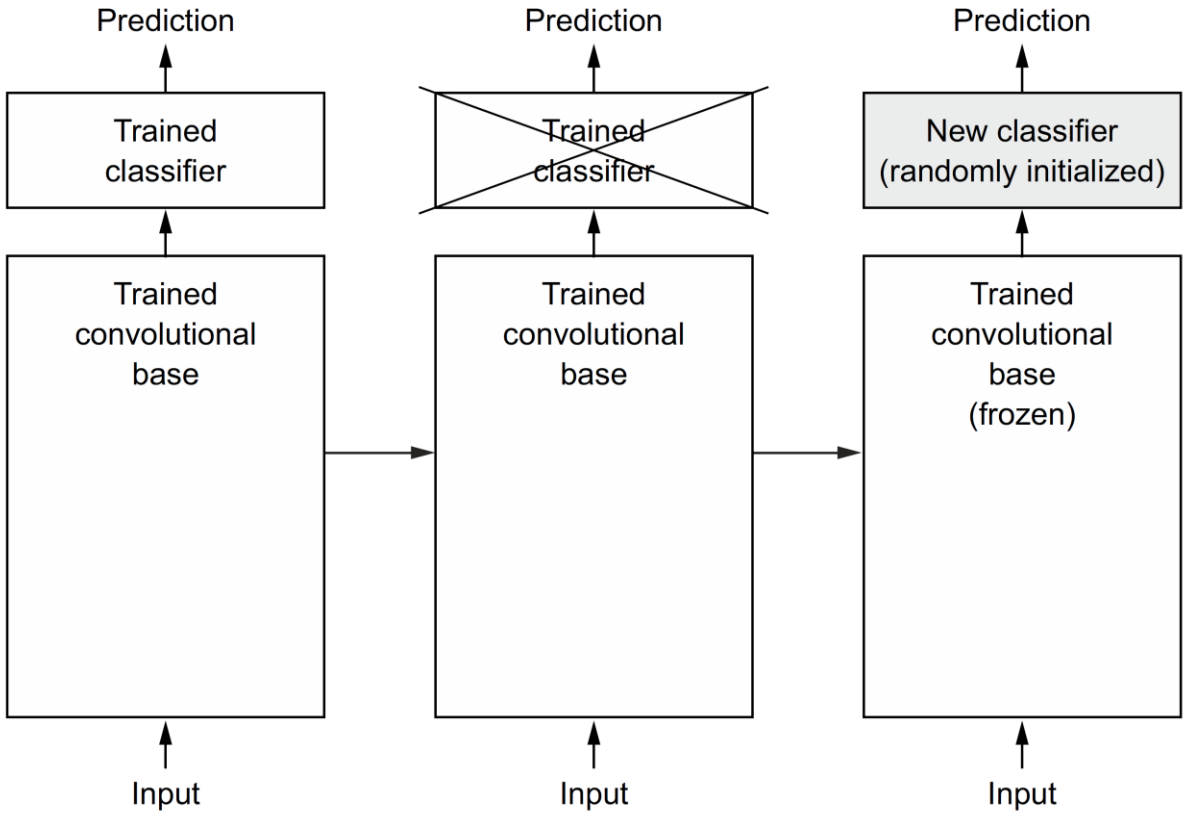
Transfer learning

- You want to find the right number of layers to reuse
  1. Try freezing all the reused layers first, then train your model and see how it performs. Then try unfreezing one or two of the top hidden layers to let backpropagation tweak them and see if performance improves
  2. The more training data you have, the more layers you can unfreeze. It is also useful to reduce the learning rate when you unfreeze reused layers
     - If you have plenty of training data, you may even try to add more hidden layers
  3. If you still cannot get good performance, and you have little training data, try dropping the top hidden layer(s) and freezing all the remaining hidden layers again. You can iterate until you find the right number of layers to reuse



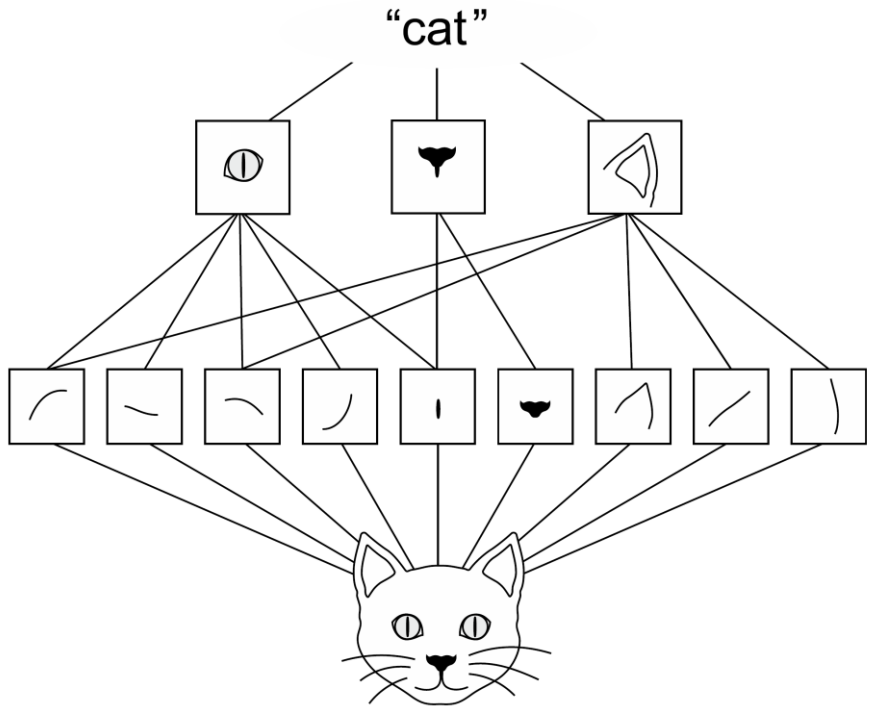
Using pretrained model – feature extraction

- *Feature extraction* with a pretrained model is often useful in visual task
  - Such portability of learned features across different problems is a key advantage of deep learning compared to traditional learning approaches and is effective for small-data problems



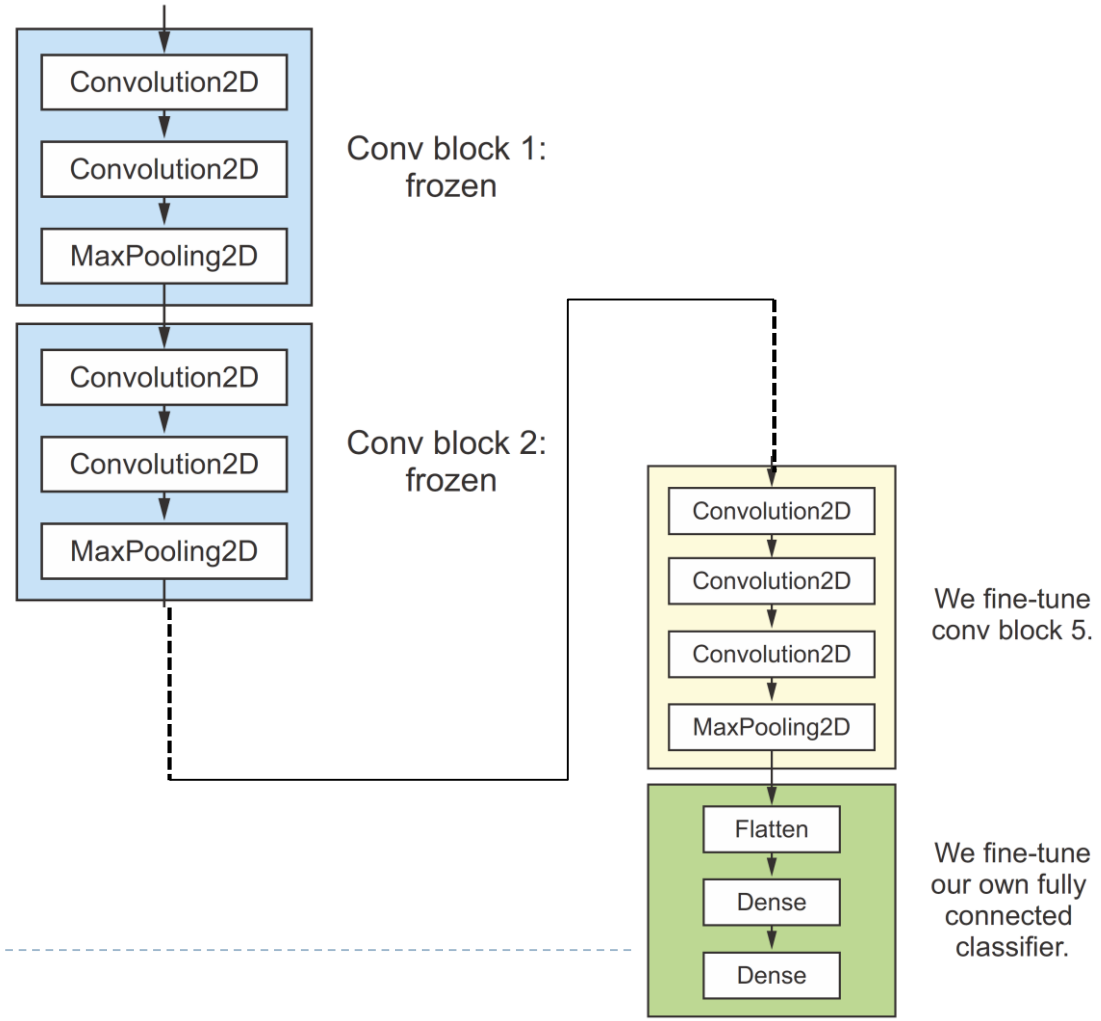
Using pretrained model – feature extraction

- Convnets start with a series of pooling and convolution layers, and they end with a densely connected classifier. The first part is called the *convolutional base* of the model
  - In the case of convnets, feature extraction consists of taking the convolutional base of a previously trained network, and training a new classifier on top of the output
- Note that the level of generality of the representations extracted by specific convolution layers depends on the depth of the layer
  - Layers that come earlier in the model extract local, highly generic feature maps (such as visual edges, colors, and textures), whereas layers that are higher up extract more-abstract concepts (such as “cat ear” or “dog eye”)



Using pretrained model – fine-tuning

- Another widely used technique for model reuse, complementary to feature extraction, is fine-tuning
  - Fine-tuning consists of unfreezing a few of the top layers of a frozen model base used for feature extraction, and jointly training both the newly added fully connected classifier and these top layers
  - When fine-tuning a model that includes BatchNormalization layers, it is recommended leaving these layers frozen. Otherwise they will keep updating their internal mean and variance, which can interfere with the very small updates applied to the surrounding Conv2D layers



Supervised pre-training

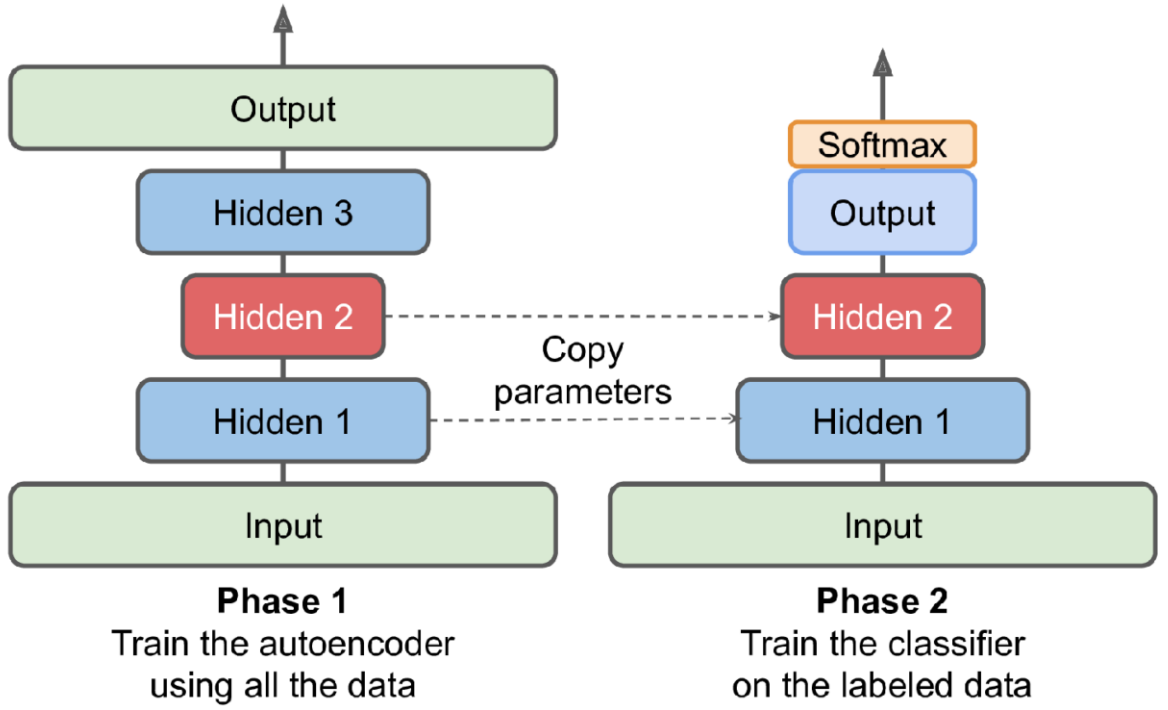
- The pre-training task may be supervised or unsupervised; the main requirements are that it can teach the model basic structure about the problem domain and that it is sufficiently similar to the downstream fine-tuning task
  - The notion of task similarity is not rigorously defined, but in practice the domain of the pre-training task is often more broad than that of the fine-tuning task
  - For example, it is very common to use the ImageNet dataset to pretrain CNNs, which can then be used for an a variety of downstream tasks and. Imagenet has 1.28 million natural images, each associated with a label from one of 1,000 classes. The classes constitute a wide variety of different concepts, including animals, foods, buildings, musical instruments, clothing, and so on
  - Another non-vision application of transfer learning is to pre-train a speech recognition on a large English-labeled corpus before fine-tuning on low-resource languages

Unsupervised pretraining

- In case you want to tackle a complex task for which you don't have much labeled data, but unfortunately you can't find a model trained on a similar task
  - If you can gather plenty of unlabeled training data, you can try to use it to train an unsupervised model, such as an autoencoder or a generative adversarial network
  - You can then reuse the lower layers of the autoencoder or GAN's discriminator, add the output layer for your task on top, and finetune the final network with the labeled training examples

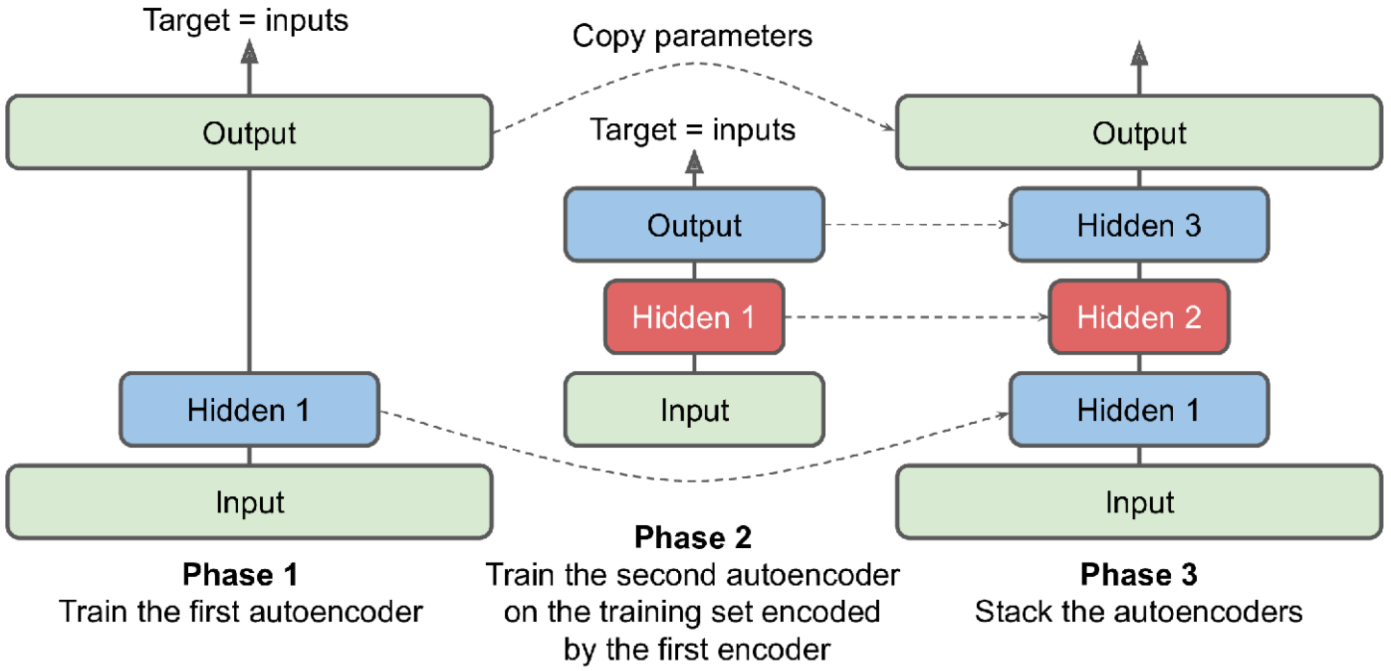
Unsupervised pretrained

- If you have a large dataset but most of it is unlabeled, you can first train a stacked autoencoder using all the data
  - Then reuse the lower layers to create a neural network for your actual task and train it using the labeled data, you may also want to freeze the pretrained layers
  - When an autoencoder is neatly symmetrical, a common technique is to tie the weights of the decoder layers to the weights of the encoder layers. This halves the number of weights in the model, speeding up training and limiting the risk of overfitting



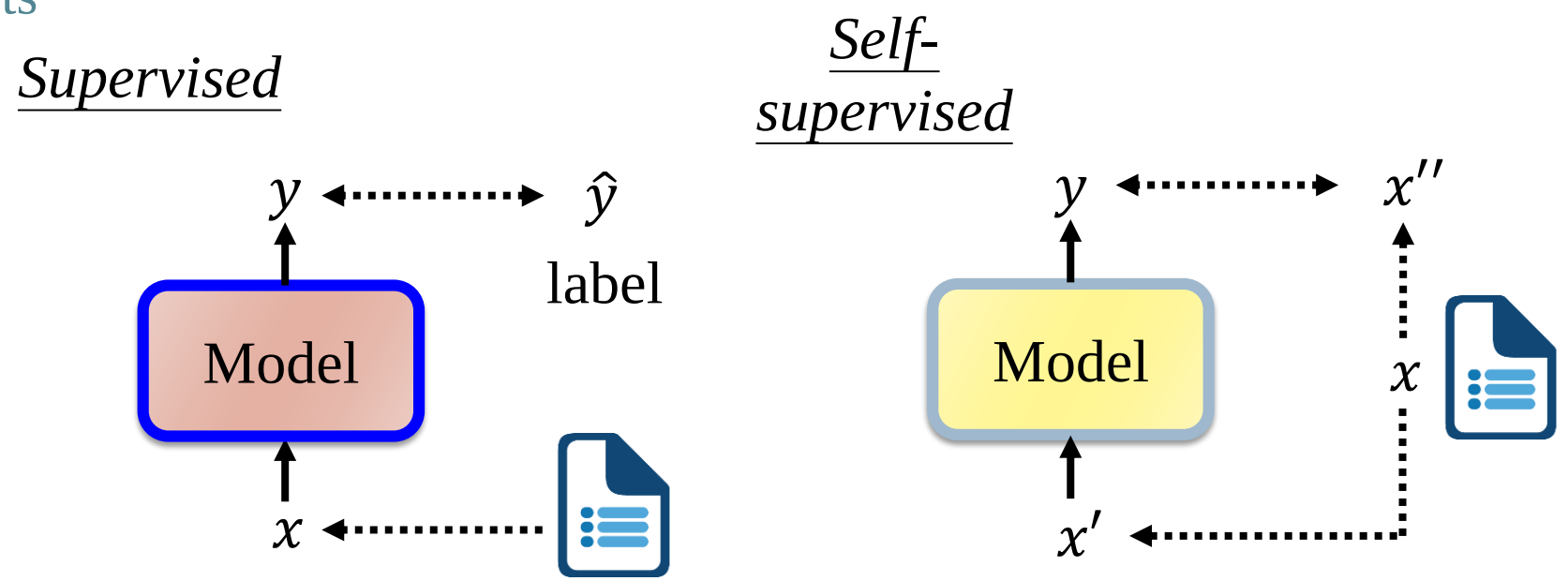
Training One Autoencoder at a Time

- It is possible to train one shallow autoencoder at a time, then stack all of them into a single stacked autoencoder called “greedy layerwise training”
  - During the first phase of training, the first autoencoder learns to reconstruct the inputs. Then we encode the whole training set using this first autoencoder, and this gives us a new (compressed) training set. We then train a second autoencoder on this new dataset. This is the second phase of training
  - Finally, we build a big sandwich using all these autoencoders, This gives us the final stacked autoencoder



Self-supervised learning

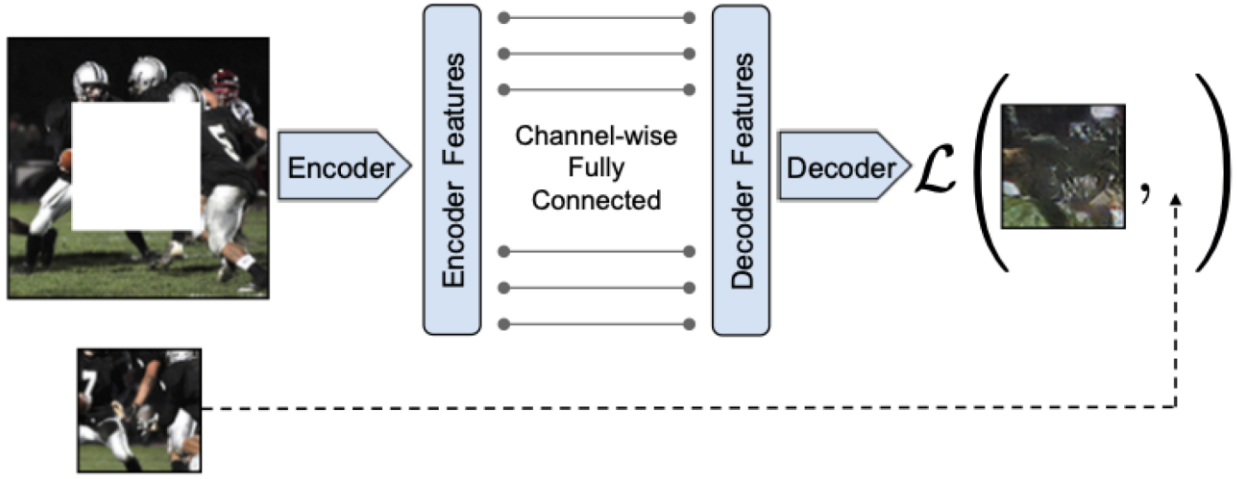
- Self-supervised learning is an active research field. Self-supervised learning is an approach to pre-training models using unlabeled data
  - This term is used because the labels are created by the algorithm, rather than being provided externally by a human, as in standard supervised learning. Both supervised and self-supervised learning are discriminative tasks, since they require predicting outputs given inputs



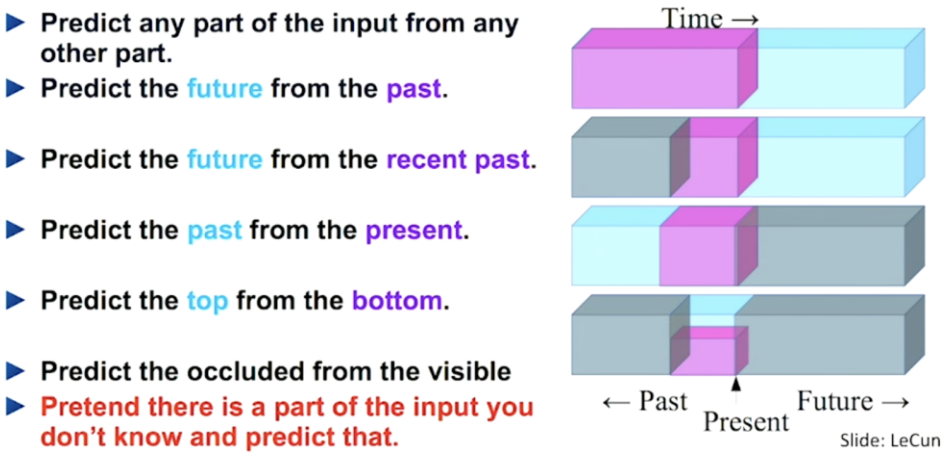
https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/bert_v8.pptx

Self-supervised learning - Pretrained using Imputation tasks

- One approach to self-supervised learning is to solve imputation tasks. In this approach, we partition the input vector x into two parts, $x = (x_h, x_v)$, and then try to predict the hidden part x_h given the remaining visible part, x_v, using a model of the form $\hat{x}_h = f(x_h = 0, x_v)$, . We can think of this as a "fill-in-the-blank" task; in the NLP community, this is called a cloze task



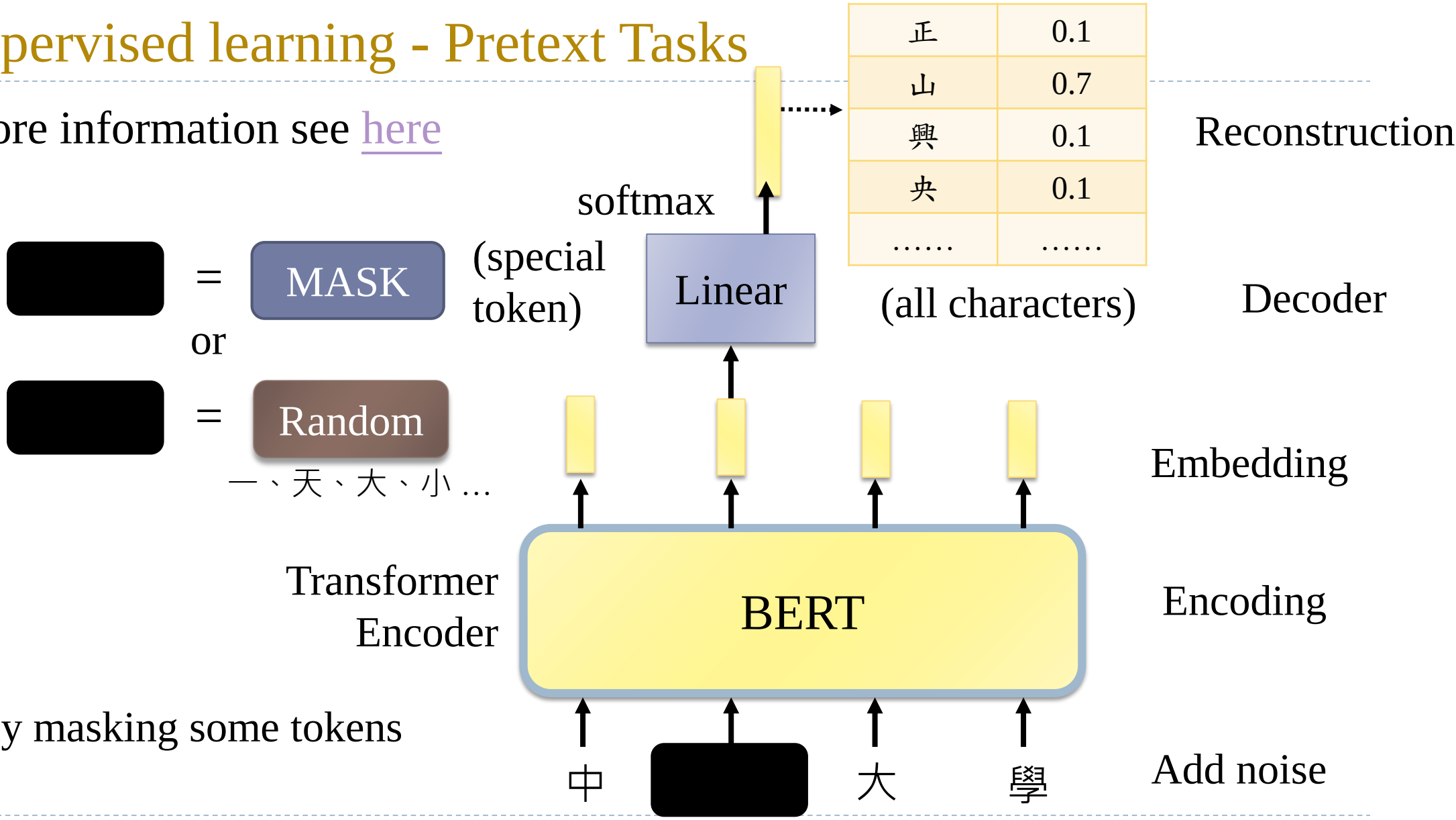
(a)



(b)

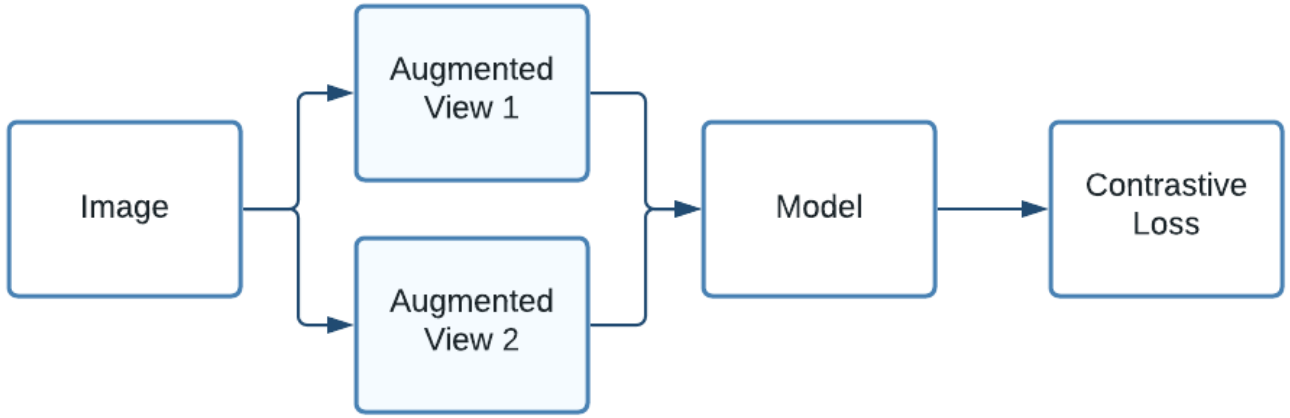
Self-supervised learning - Pretext Tasks

- For more information see here



Self-supervised learning - contrastive tasks

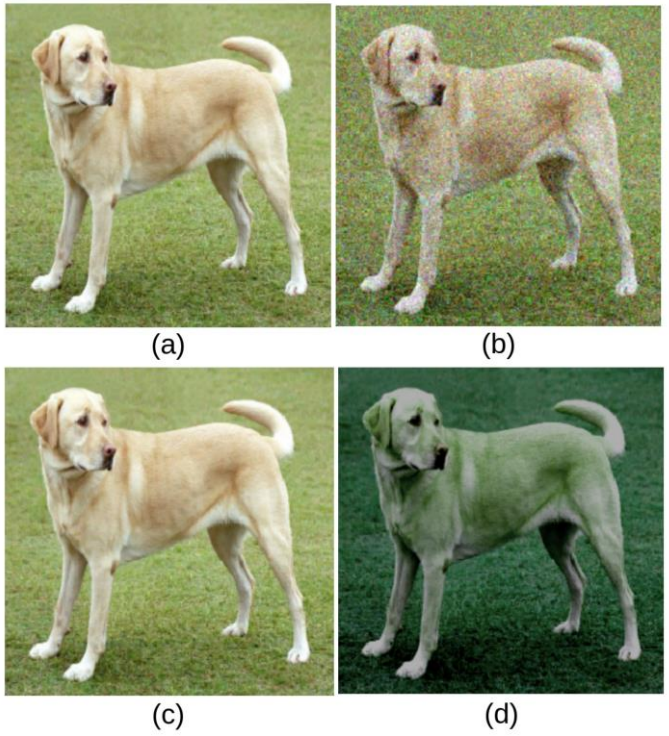
- Solve proxy tasks, also called pretext tasks
  - The basic idea is to create pairs of examples that are semantically similar to each other, using data augmentation methods
  - Train a self-supervised model to learn data representations by contrasting multiple augmented views of the same example. These learned representations capture data invariants, e.g., object translation, color jitter, noise, etc

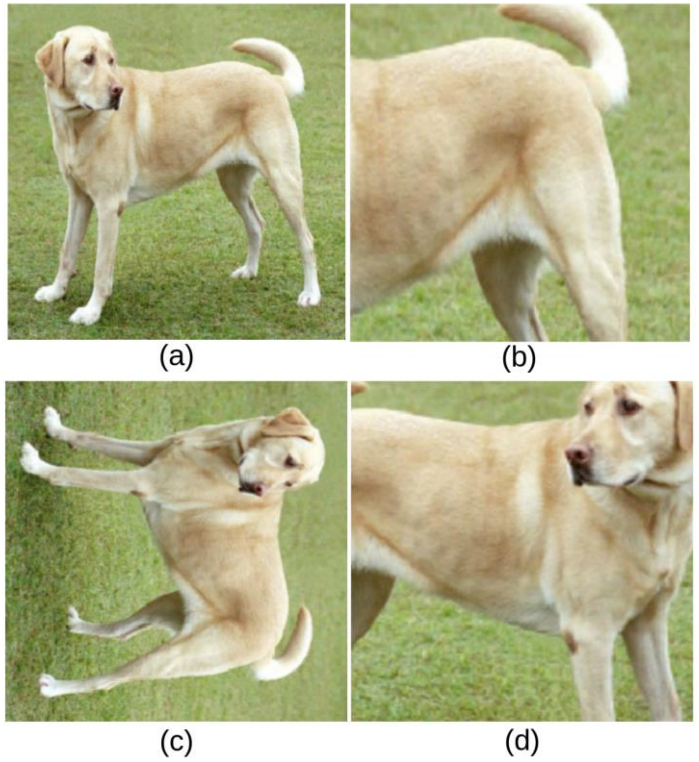


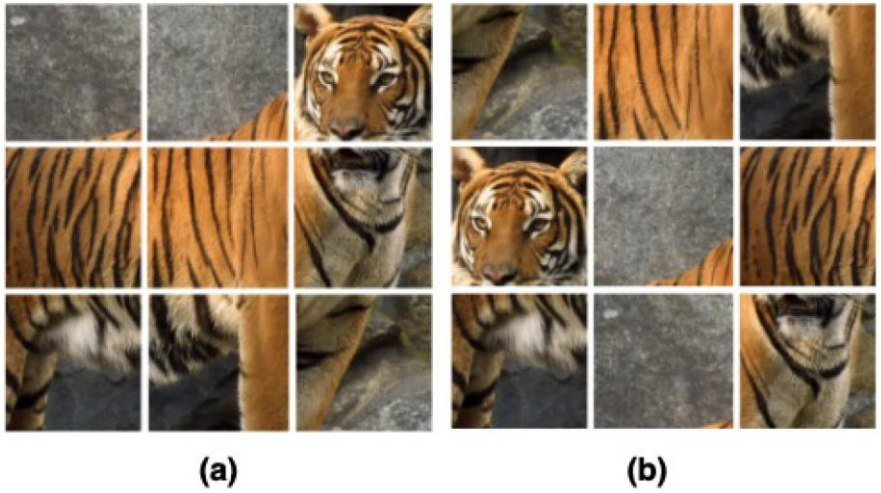
https://arxiv.org/abs/2011.00362

Self-supervised learning - contrastive tasks

- Color Transformation
- Geometric Transformation
- Jigsaw puzzle

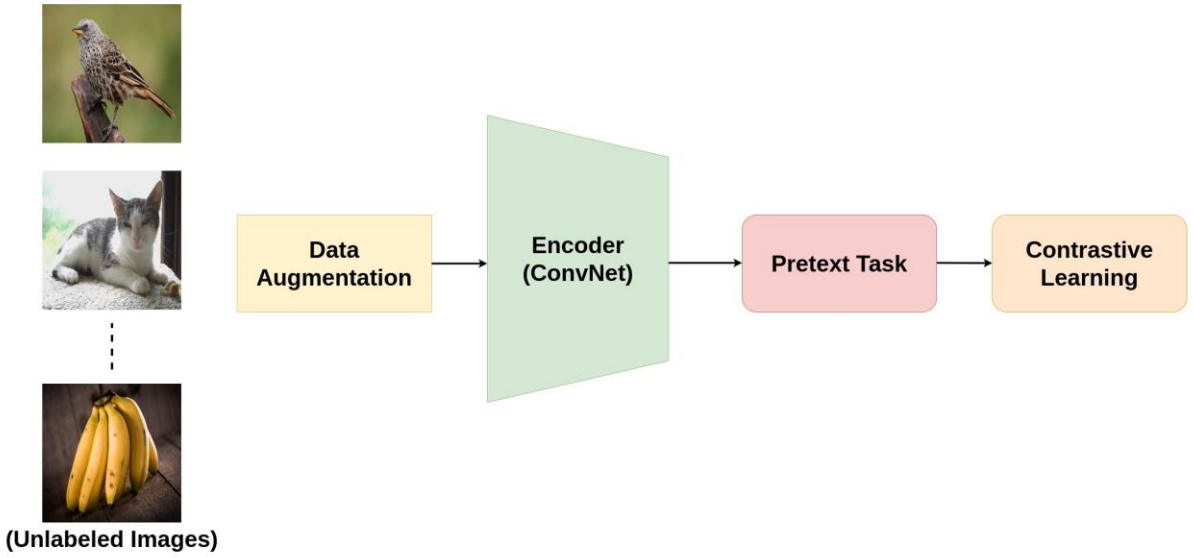


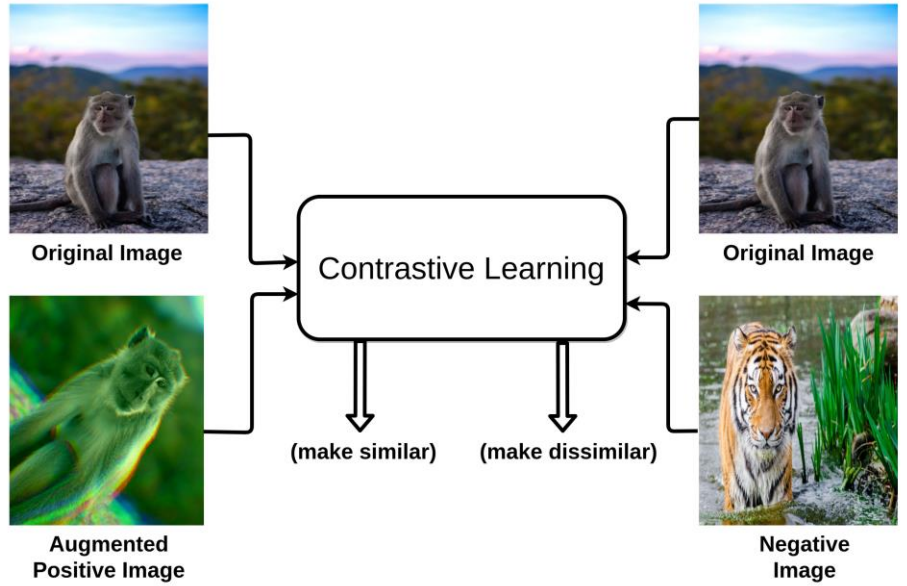




Self-supervised learning - contrastive tasks

- Pretext tasks are self-supervised tasks that act as an important strategy to learn representations of the data using pseudo labels
  - These pseudo labels are generated automatically based on the attributes found in the data
  - The original image acts as an anchor, its augmented version acts as a positive sample, and the rest of the images in the batch or in the training data act as negative samples





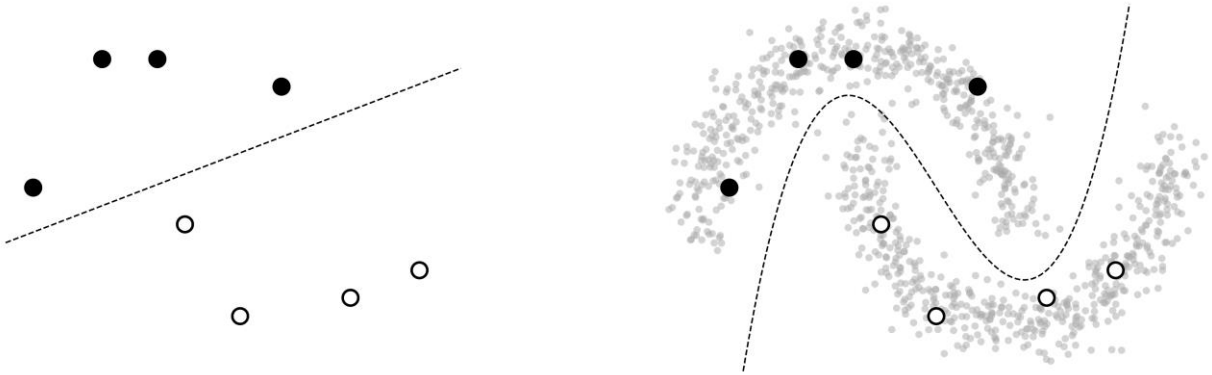
https://arxiv.org/abs/2011.00362

Self-supervised learning

- Identifying the right pre-text task
  - The choice of pretext task relies on the type of problem being solved
  - The main aim of a pre-text task is to compel the model to be *invariant* to these transformations while remaining discriminative to other data points
  - For instance, applying rotation may help with view-independent aerial image recognition but might significantly downgrade the performance while trying to solve downstream tasks such as detecting which way is up in a photograph for a display application
  - Similarly, colorization-based pretext tasks might not work out in a fine-grain classification represented in figure

Semi-supervised learning

- Semi-supervised learning can alleviate the need for labeled data by taking advantage of unlabeled data
  - The general goal of semi-supervised learning is to allow the model to learn the high-level structure of the data distribution from unlabeled data and only rely on the labeled data for learning the fine-grained details of a given task
  - Whereas in standard supervised learning we assume that we have access to samples from the joint distribution of data and labels $x, y \sim p(x, y)$, semi-supervised learning assumes that we additionally have access to samples from the marginal distribution of $x \sim p(x)$

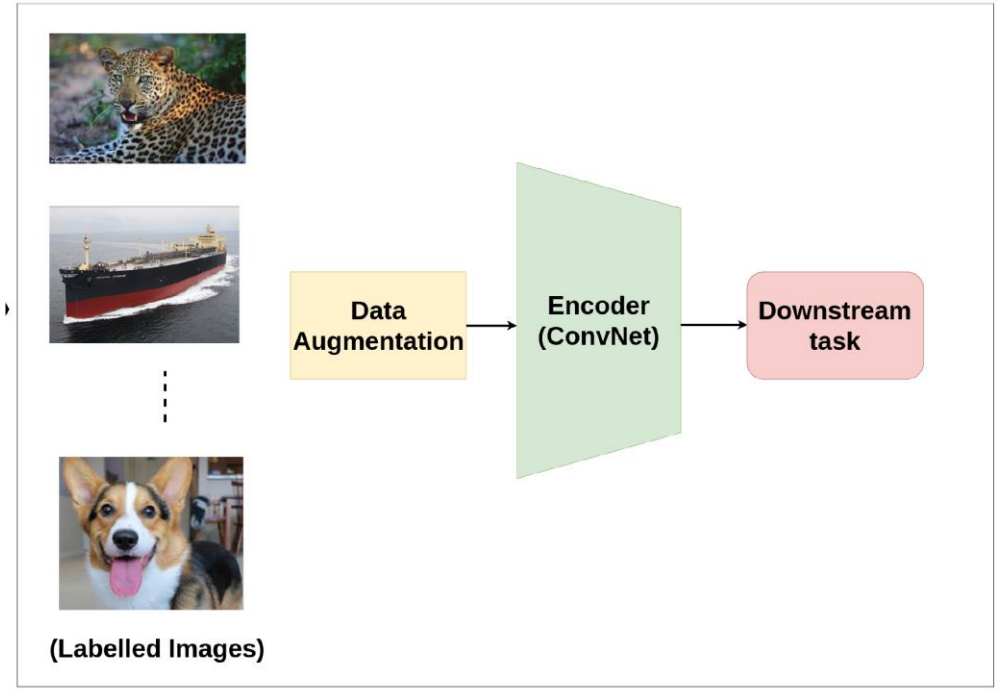


Self-training and pseudo-labeling

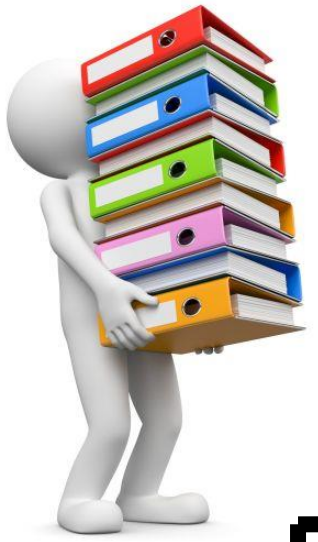
- A straightforward approach to *semi-supervised learning* is self-training
  - The basic idea behind self-training is to use the model itself to infer predictions on unlabeled data, and then treat these predictions as labels for subsequent training
  - Recently, it has become common to refer to this approach as “pseudo-labeling” because the inferred labels for unlabeled data are only “pseudo-correct” in comparison with the true, ground-truth targets used in supervised learning
  - A common strategy is to use a “selection metric” which tries to only retain pseudo-labels that are correct. For example, assuming that a model outputs probabilities for each possible class, a frequently-used selection metric is to only retain pseudo-labels whose largest class probability is above a threshold
  - Also refer to noisy student approach, some recent paper advocate self-training approach rather than supervised or self-supervised way, see here

Downstream task

- Downstream tasks are application-specific tasks that utilize the knowledge that was learned during the pretext task
  - Training a classifier on top of the frozen representations is easier and requires fewer labels because the pre-trained model already produces meaningful and generally useful features
  - The learned parameters serve as a pretrained model and are transferred to other downstream computer vision tasks by fine-tuning
  - The encoder can then be used to produce embedding or latent space



Downstream task



Self-supervised Learning

Pre-train

BERT

- Masked token prediction

Fine-tune

Model for Task 1

Model for Task 2

Model for Task 3

Downstream Tasks

- The tasks we care
- We have a little bit labeled data.

https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/bert_v8.pptx

Domain adaptation

Training Data

Testing Data

99.5%

57.5%

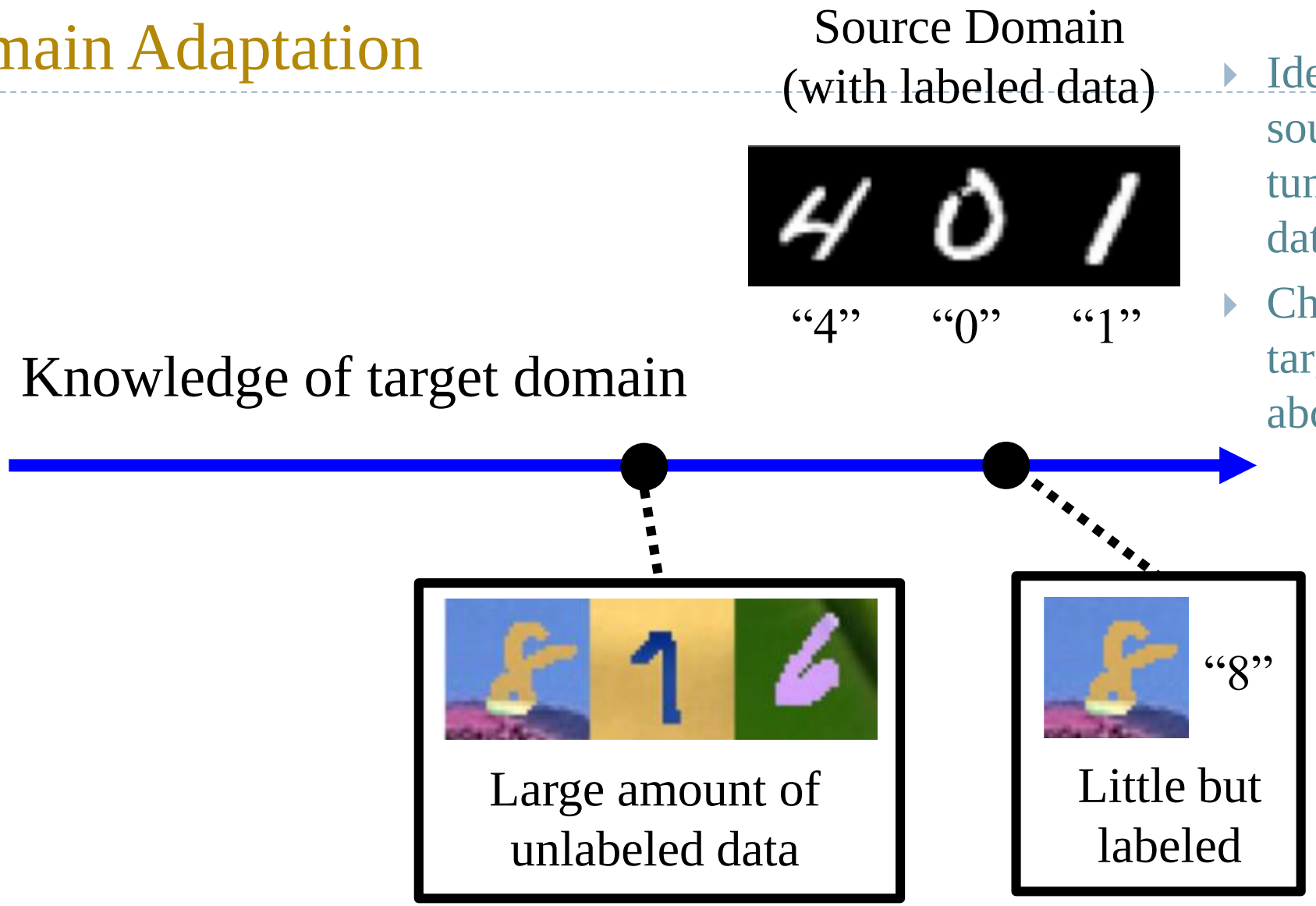
The results are from: http://proceedings.mlr.press/v37/ganin15.pdf

Domain shift: Training and testing data have different distributions.

Domain adaptation

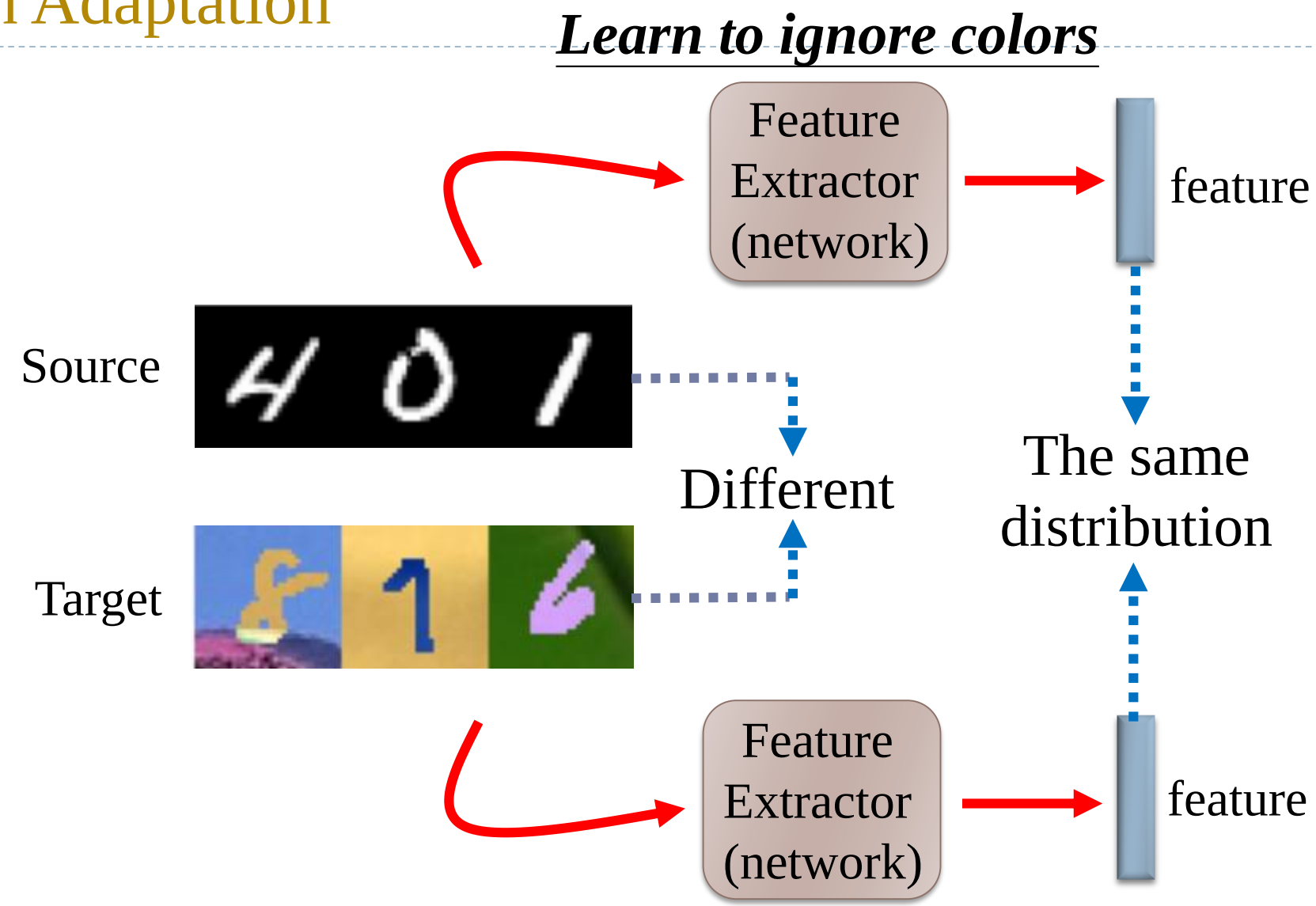
https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/da_v6.pptx

Domain Adaptation

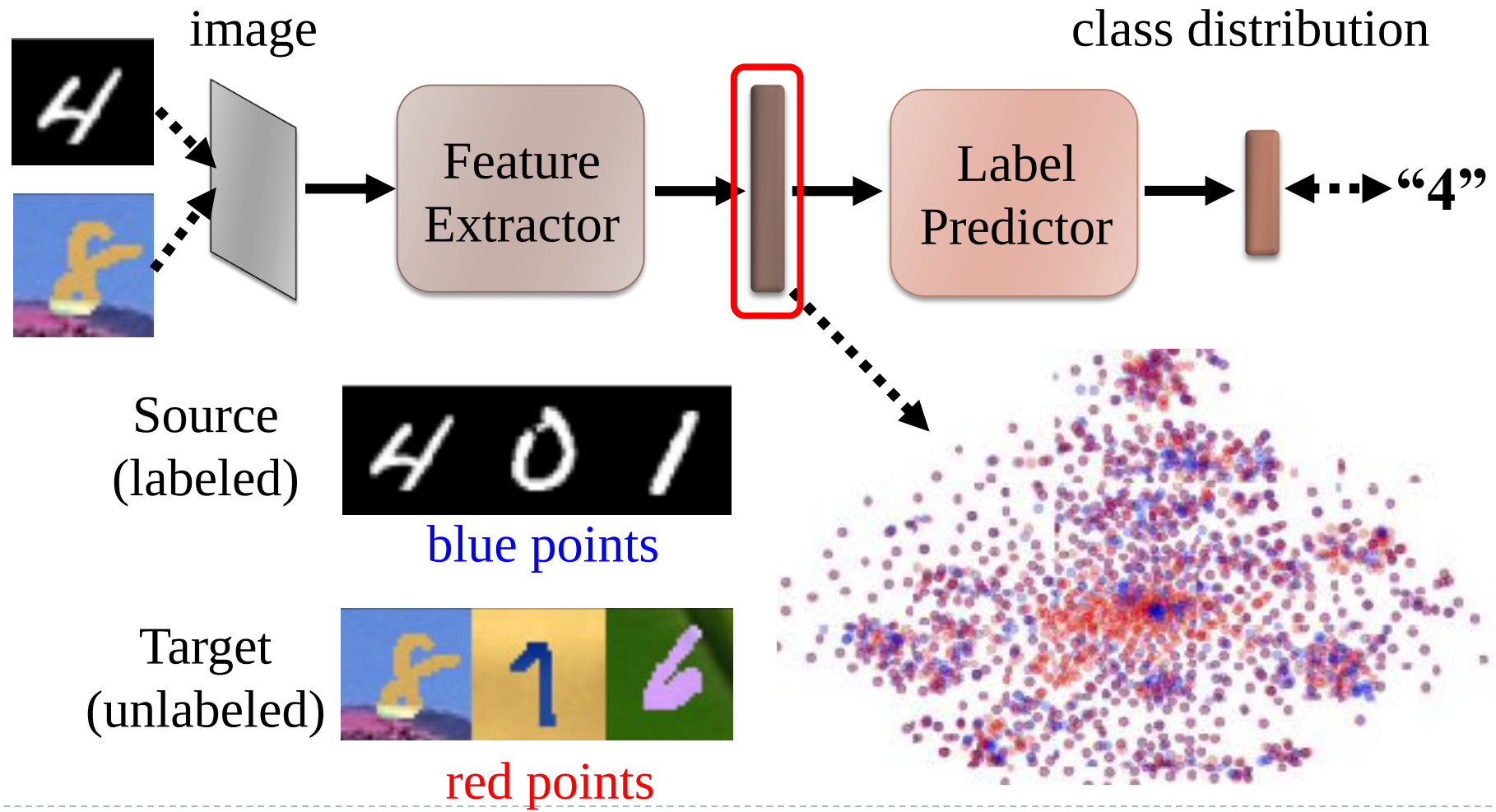


- Idea: training a model by source data, then fine-tune the model by target data
- Challenge: only limited target data, so be careful about overfitting

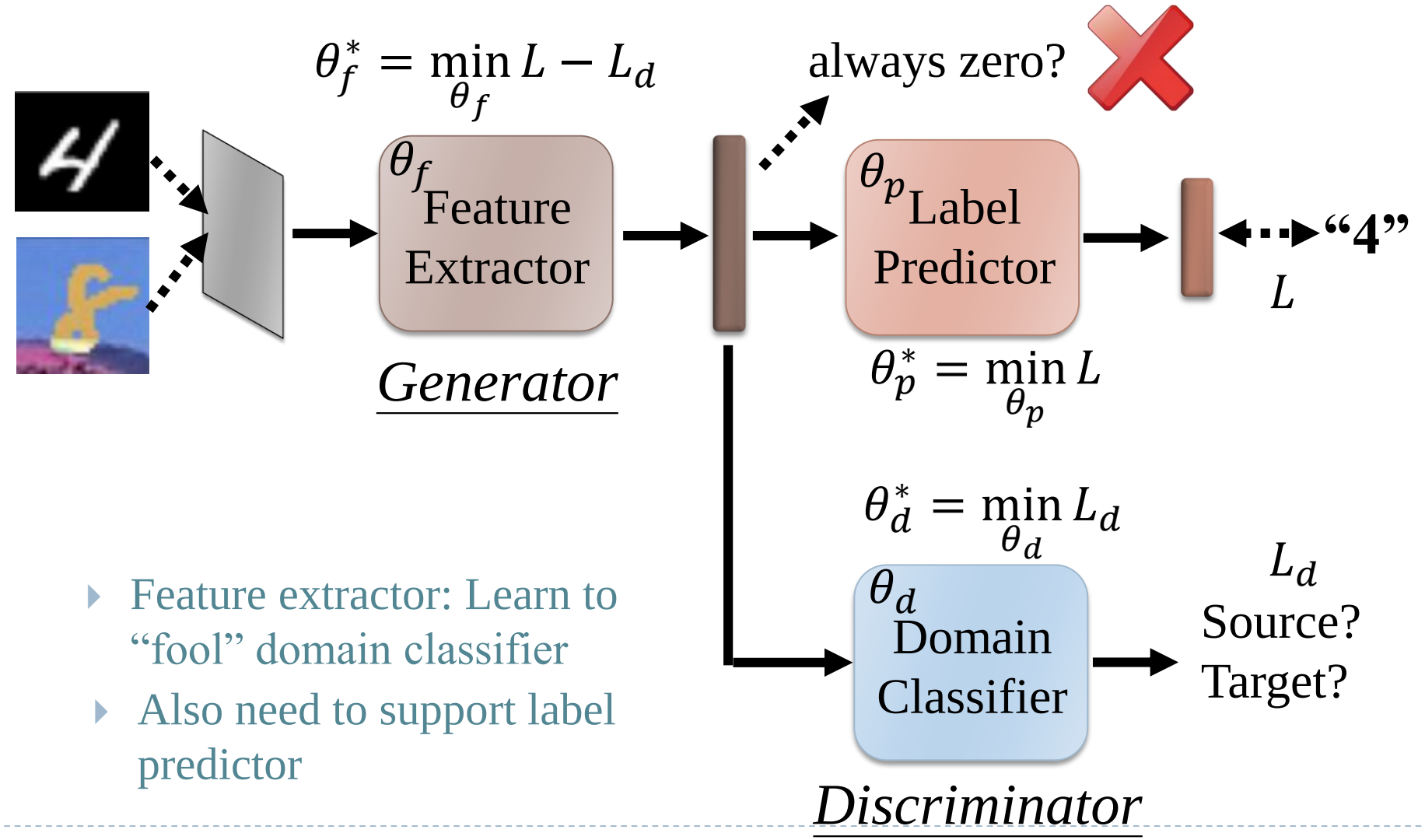
Domain Adaptation



Domain Adversarial Training



Domain Adversarial Training



- Feature extractor: Learn to "fool" domain classifier
- Also need to support label predictor

Domain Adversarial Training

METHOD	SOURCE / TARGET	MNIST / MNIST-M	SYN NUMBERS / SVHN	SVHN / MNIST	SYN SIGNS / GTSRB
SOURCE ONLY		.5749	.8665	.5919	.7400
PROPOSED APPROACH		**.8149** (57.9%)	**.9048** (66.1%)	**.7107** (29.3%)	**.8866** (56.7%)
TRAIN ON TARGET		.9891	.9244	.9951	.9987

Conclusion

- Many ML models, especially neural networks, often have many more parameters than we have labeled training examples
  - Of course these parameters are highly correlated, so they are not independent “degrees of freedom”. Nevertheless, such big models are slow to train and, more importantly, they may easily overfit. This is particularly a problem when you do not have a large labeled training set
- Pretraining using supervised, unsupervised or self-supervised way can greatly benefit the downstream tasks by transferring knowledge from one task to another
- In some cases, domains might be even different in training and testing phases
  - Our goal is to fit the model on the source domain, and then modify its parameters so it works on the target domain. This is called (unsupervised) domain adaptation

References

[1] Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition Chapter 11,14

[2] Deep learning with Python, 2nd Edition Chapter 8

[3] https://speech.ee.ntu.edu.tw/~hylee/ml/2022-spring.php Lecture 7 and Lecture 11

[4] A Survey on Contrastive Self-supervised Learning

Appendix

Resources

- Repositories
  - https://www.tensorflow.org/hub
- Tutorial on transfer learning
  - https://www.tensorflow.org/tutorials/images/transfer_learning
- Tutorials on other related topics
  - https://d2l.ai/chapter_natural-language-processing-pretraining/index.html (NLP Pretrained)
  - https://github.com/koshian2/Pseudo-Label-Keras (Pseudo labeling)
  - Multi-Task Learning
  - Zero-Shot Learning
  - A Survey on Contrastive Self-supervised Learning

Active learning

- In active learning, the goal is to identify the true predictive mapping $y = f(x)$ by querying as few (x, y) points as possible
  - There are three main variants. In query synthesis, the algorithm gets to choose any input x, and can ask for its corresponding output $y = f(x)$. In pool-based active learning, there is a large, but fixed, set of unlabeled data points, and the algorithm gets to ask for a label for one or more of these points. Finally, in stream-based active learning, the incoming data is arriving continuously, and the algorithm must choose whether it wants to request a label for the current input or not
  - There are various closely related problems. In *Bayesian optimization* the goal is to estimate the location of the global optimum $\hat{x} = argminx\ f(x)$ in as few queries as possible; typically we fit a surrogate (response surface) model to the intermediate (x,y) queries, to decide which question to ask next
  - In *experiment design*, the goal is to infer a parameter vector of some model, using carefully chosen data samples $D = \{x_1, \ldots, x_N\}$, i.e. we want to estimate $p(\theta|D)$ using as little data as possible

Weakly unsupervised learning

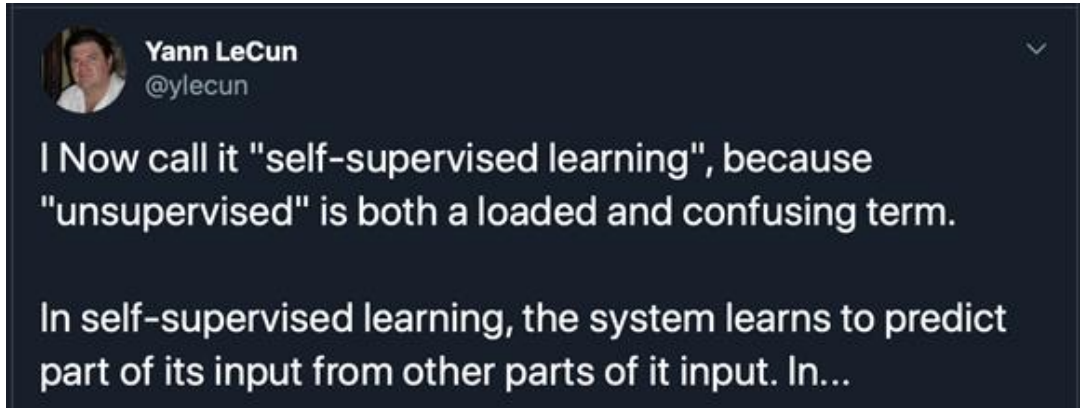
- The term weakly supervised learning refers to scenarios where we do not have an exact label associated with every feature vector in the training set
  - One scenario is when we have a distribution over labels for each case, rather than a single label. Fortunately, we can still do maximum likelihood training: we just have to minimize the cross entropy,

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_n \sum_y p(y|\boldsymbol{x}_n) \log q_{\boldsymbol{\theta}}(y|\boldsymbol{x}_n)$$

  - Where $p(y|x_n)$ is the label distribution for case n, and $q_\theta(y|x_n)$ is the predicted distribution. Indeed, it is often useful to artificially replace exact labels with a "soft" version, in which we replace the delta function with a distribution that puts, say, 90% of its mass on the observed label, and spreads the remaining mass uniformly over the other choices. This is called *label smoothing*, and is a useful form of regularization

What is Self-Supervised Learning?

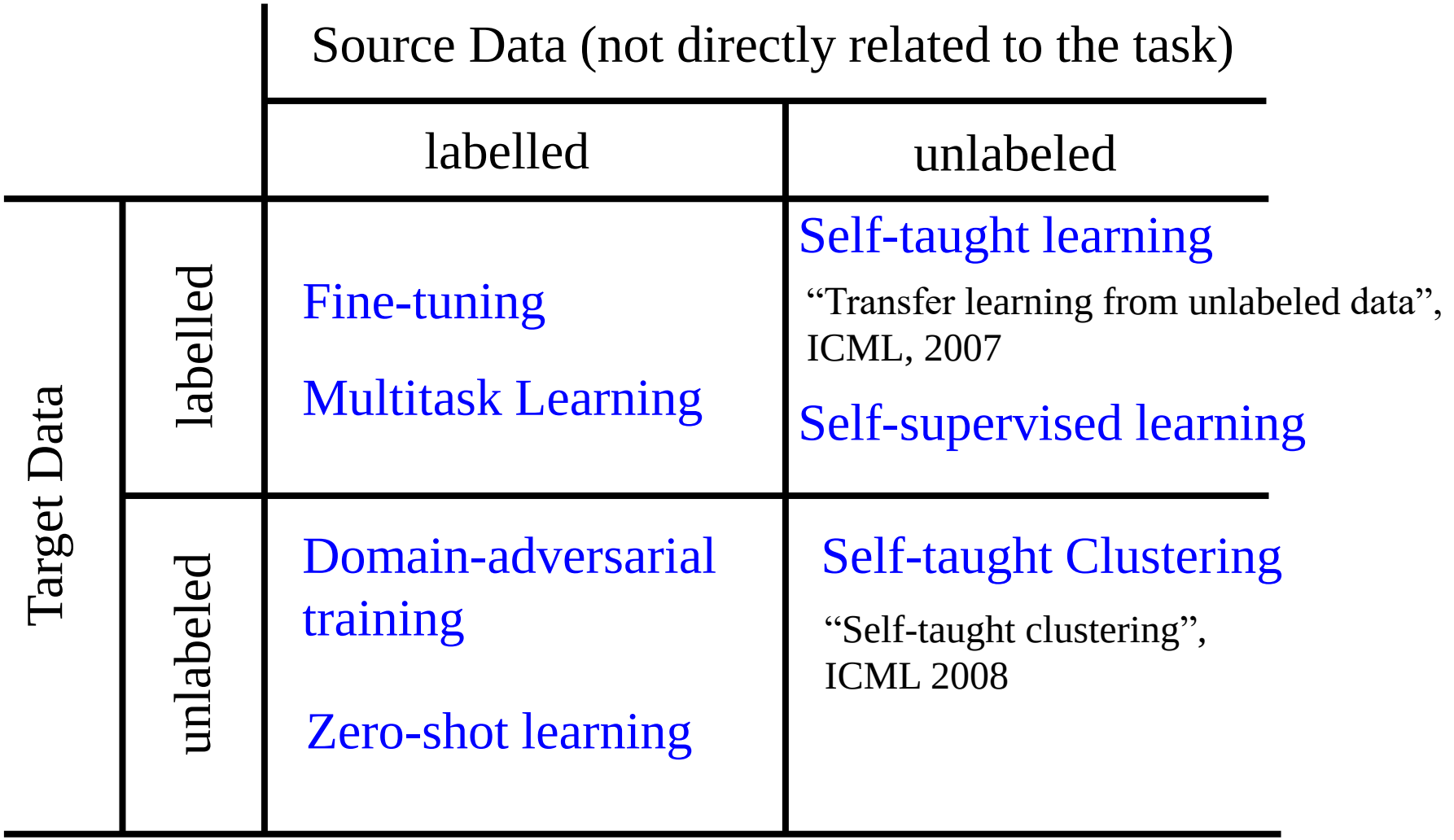
- A version of unsupervised learning where data provides the supervision.



- In general, withhold some part of the data and the task a neural network to predict it from the remaining parts.

- Goal: Learning to represent the world before learning tasks.

Taxonomy of Transfer learning



https://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2017/Lecture/transfer.pptx