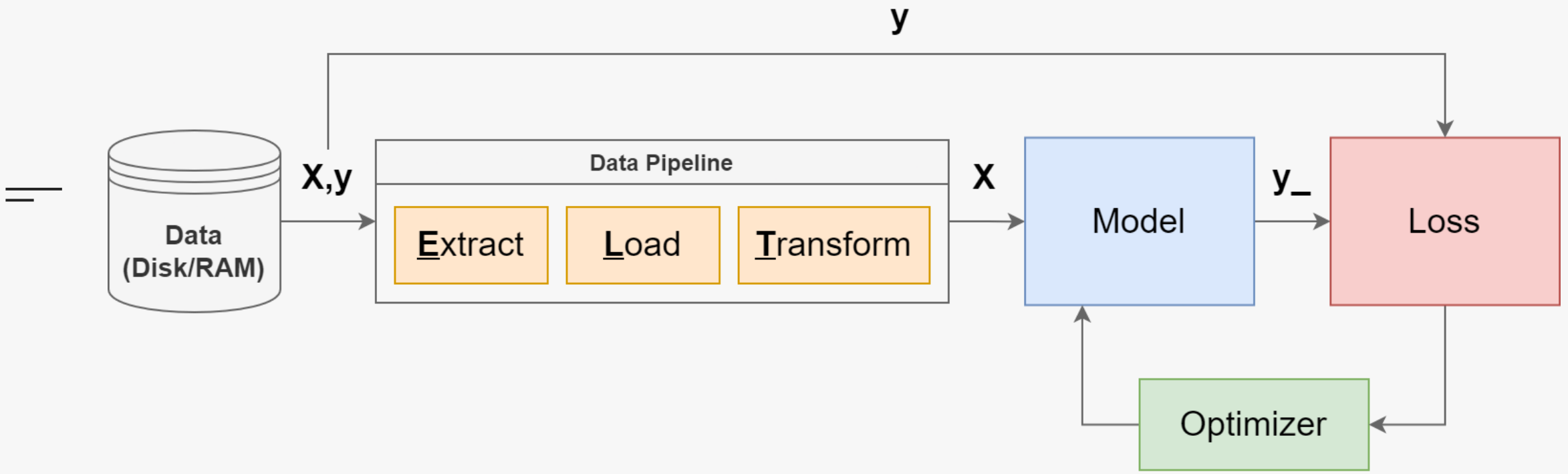
Deep Learning for Computer Vision

Ahmed Hosny Abdel-Gawad

Senior AI/CV Engineer

ConvNets with Small Datasets

Data Pipelines - ETL



Data Pipelines - ETL

Images

Files

CSV

TFRecords

...

Load into:

Numpy arrays

Tensors

...

Data

augmentation

Scaling

...

Feed data to the model: batch-by-batch,

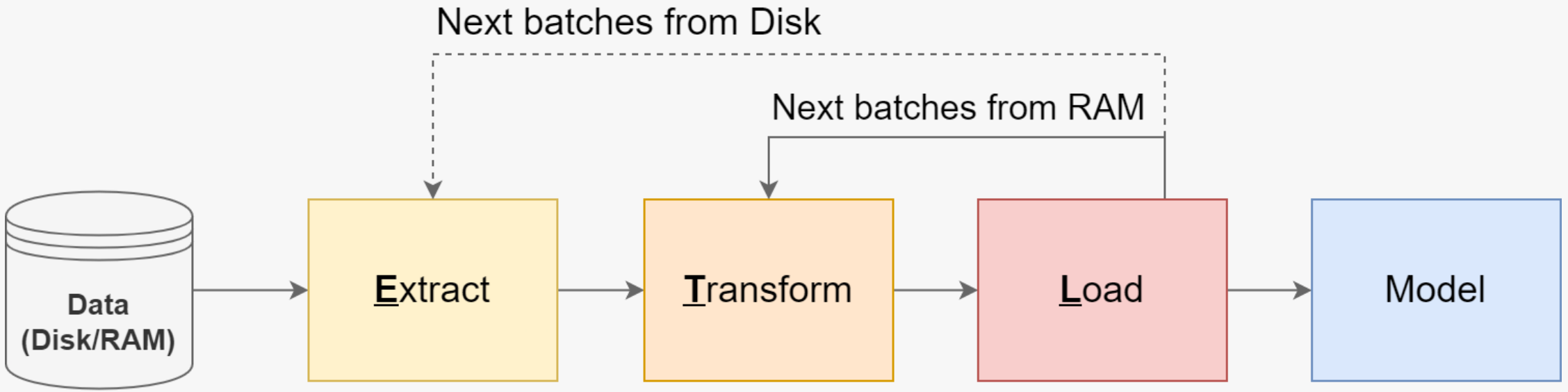
Shuffled,

Epochs,

(ex: fit)

Handle train_step in train loop

Data in RAM or Disk



You could think of this as something that generates batches of data, each of size=batch_size, but without loading the whole data in RAM.

Instead, we just want to load the batch_size in RAM.

Image Augmentation

Image Data Augmentation

Data augmentation is a technique to increase the variation in a dataset by applying transformations to the original data. It is often used when the training data is limited and as a way of preventing **overfitting**.

Image augmentation generates similar but distinct training examples after a series of random changes to the training images, thereby expanding the size of the training set.

Image augmentation can be motivated by the fact that random tweaks of training examples allow models to less rely on certain attributes, thereby improving their **generalization** ability.

For example, we can crop an image in different ways to make the object of interest appear in different positions, thereby reducing the dependence of a model on the position of the object. We can also adjust factors such as brightness and color to reduce a model's sensitivity to color.

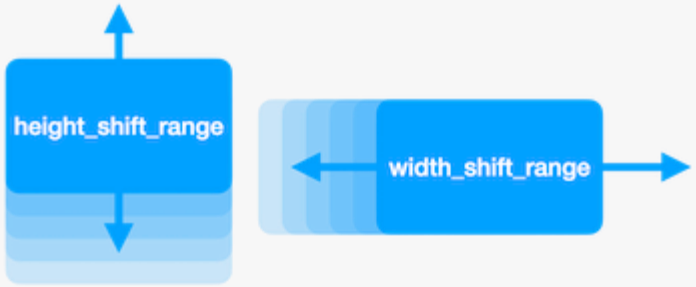
Data augmentation is usually done on the fly when training a model. While it can be done, it is usually not practical to store the augmented data on disk. After all, we want to vary the augmented data every time it is shown to the model!

Image Augmentation Techniques

We can apply various changes to the initial data. For example, for images we can use:

1. **Geometric transformations** – you can randomly flip, crop, rotate or translate images, and that is just the tip of the iceberg
2. **Color space transformations** – change RGB color channels, intensify any color
3. **Kernel filters** – sharpen or blur an image
4. **Random Erasing** – delete a part of the initial image
5. **Mixing images** – basically, mix images with one another. Might be counterintuitive but it works

Geometric Image Augmentations



Width & Height Shift

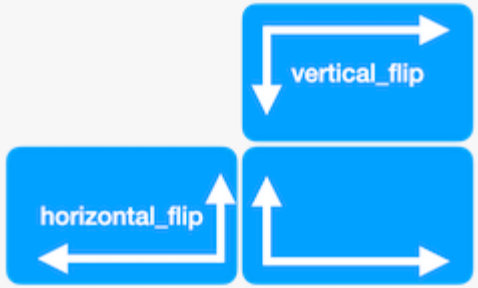
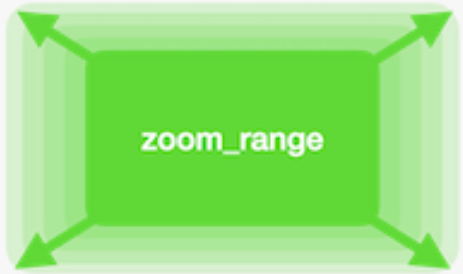
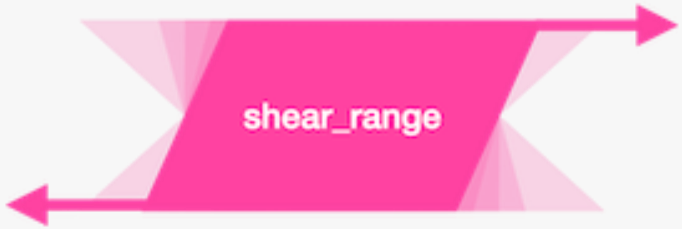


Image Flips

Rotation



Zoom



Shear

Brightness

Augmentation with Lots of Data?

Brand A (Ford)

Brand B (Chevrolet)

A Ford car (Brand A), but facing right.

Machine Learning algorithms find the most obvious features that distinguishes one class from another.

Here, the feature was that all cars of Brand A were facing left, and all cars of Brand B are facing right.

Augmentation can help to increase the amount of **relevant data** in dataset.

link

Test-time Data Augmentation

Classes = [airplane, automobile, **bird**, cat, deer, dog, frog, horse, ship, truck]

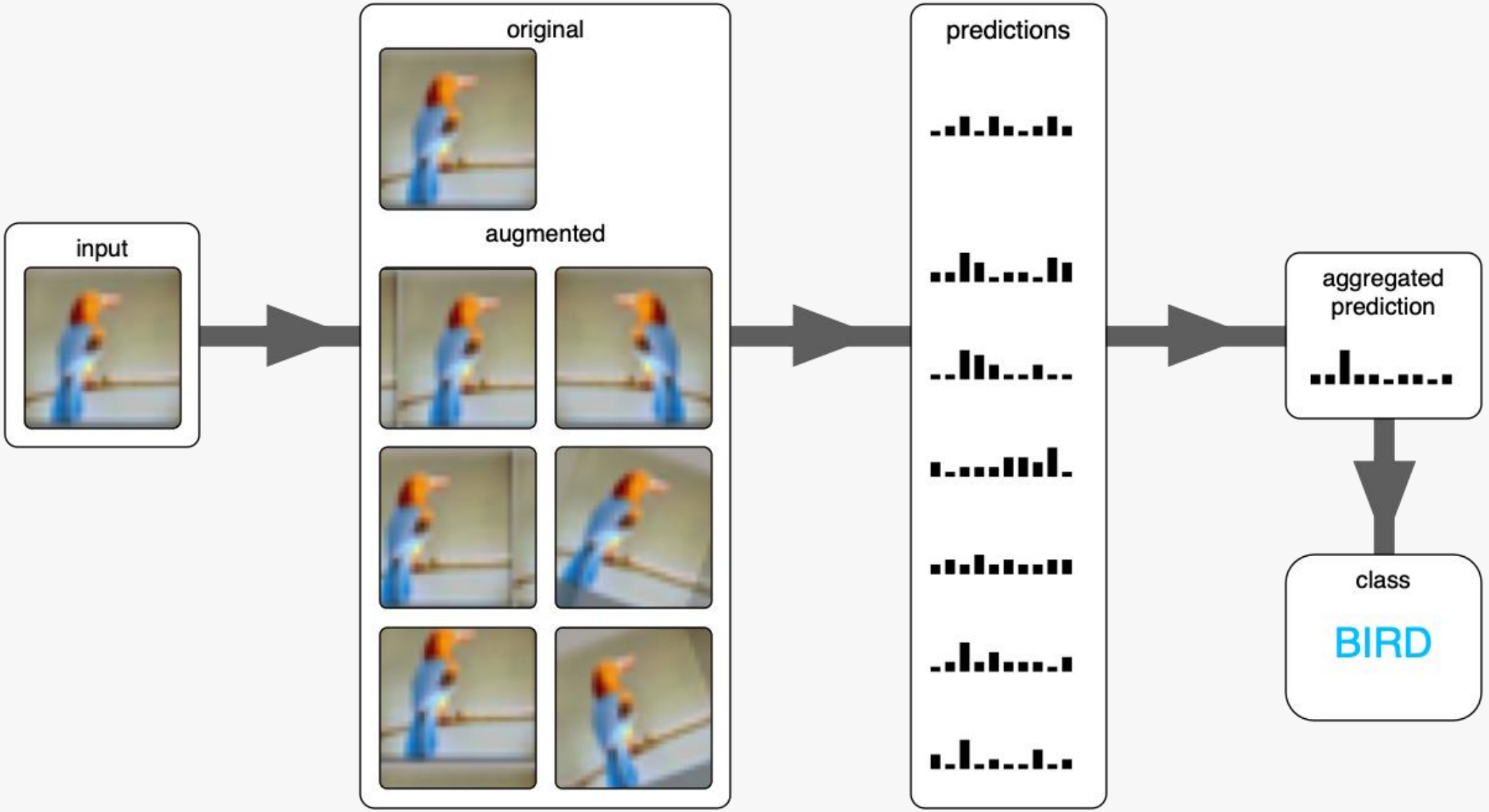
y_pred = [[0.08 0.00 **0.54** 0.01 0.37 0.00 0.00 0.00 0.00 0.00]]

y_pred = [[0.08 0.00 **0.54** 0.01 0.37 0.00 0.00 0.00 0.00 0.00]
[**0.43** 0.00 0.11 0.00 0.39 0.01 0.00 0.01 0.04 0.00]]

avg_y = [[0.25 0.00 0.33 0.01 **0.38** 0.01 0.00 0.01 0.02 0.00]]

Classes = [airplane, automobile, bird, cat, **deer**, dog, frog, horse, ship, truck]

Test-time Data Augmentation



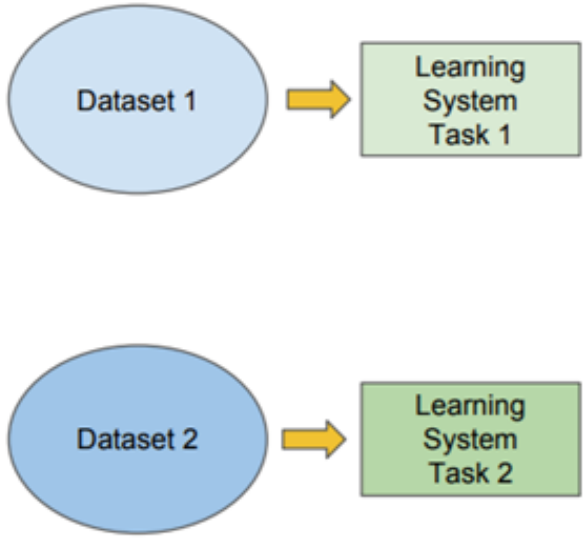
Transfer Learning

What is Transfer Learning?

Deep learning experts introduced transfer learning to overcome the limitations of traditional machine learning models.

Traditional ML

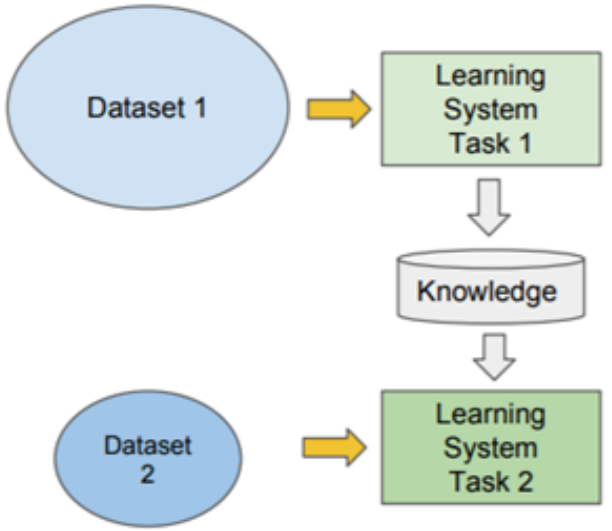
- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



vs

Transfer Learning

- Learning of a new tasks relies on the previous learned tasks:
  - Learning process can be faster, more accurate and/or need less training data



Classical Transfer Learning Strategies

Different transfer learning strategies and techniques are applied based on **the domain of the application**, the **task at hand**, and the **availability of data**.

Before deciding on the strategy of transfer learning, it is crucial to have an answer of the following questions:

- **Which** part of the knowledge can be transferred from the source to the target to improve the performance of the target task?
- **When** to transfer and when not to, so that one improves the target task performance/results and does not degrade them?
- **How** to transfer the knowledge gained from the source model based on our current domain/task?

Transfer Learning Scenarios

The three major Transfer Learning scenarios look as follows:

1. ConvNet as fixed feature extractor: Take a ConvNet pretrained on ImageNet, remove the last fully-connected layer (this layer's outputs are the 1000 class scores for a different task like ImageNet), then treat the rest of the ConvNet as a fixed feature extractor for the new dataset.

2. Fine-tuning the ConvNet: Not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation.

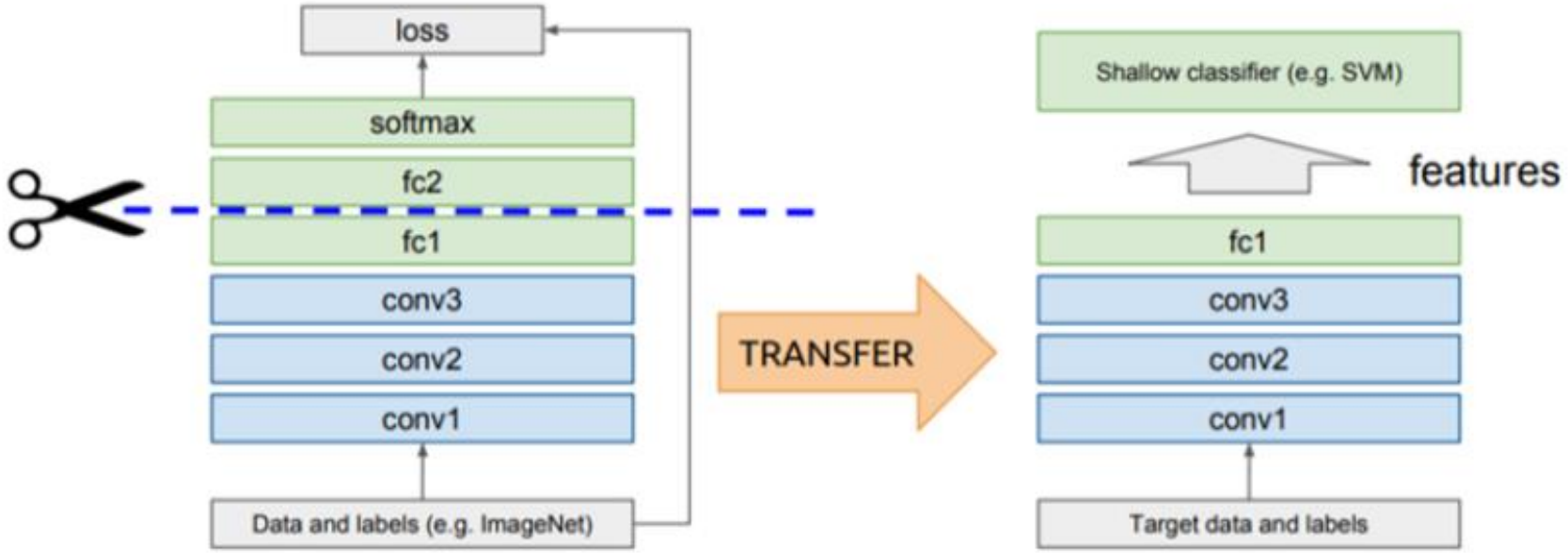
3. Pretrained models: Since modern ConvNets take 2-3 weeks to train across multiple GPUs on ImageNet, it is common to see people release their final ConvNet checkpoints for the benefit of others who can use the networks for fine-tuning.

More..

ConvNets as Feature Extractors

Idea: use outputs of one or more layers of a network trained on a different task as generic feature detectors. Train a new shallow model on these features.

Assumes that $D_S = D_T$



Transfer Learning with Pre-trained Deep Learning Models as Feature Extractors

Fine-tunning the ConvNets

Fine-tuning: supervised domain adaptation

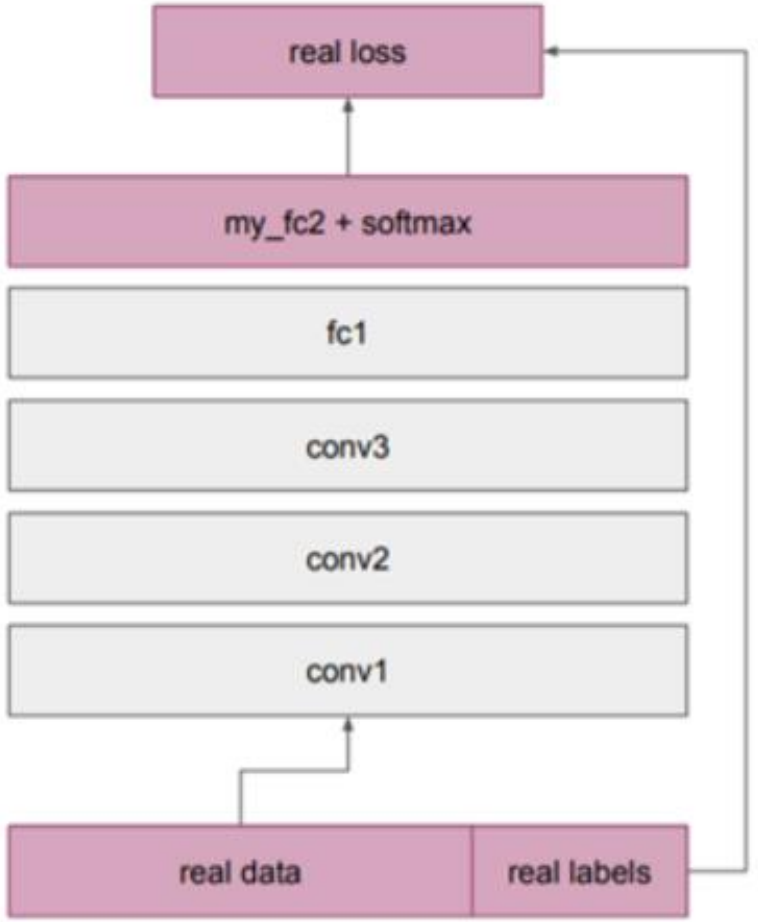
Train deep net on "nearby" task for which it is easy to get labels using standard backprop

- E.g. ImageNet classification
- Pseudo classes from augmented data
- Slow feature learning, ego-motion

Cut off top layer(s) of network and replace with supervised objective for target domain

Fine-tune network using backprop with labels for target domain until validation loss starts to increase

Aligns D_S with D_T

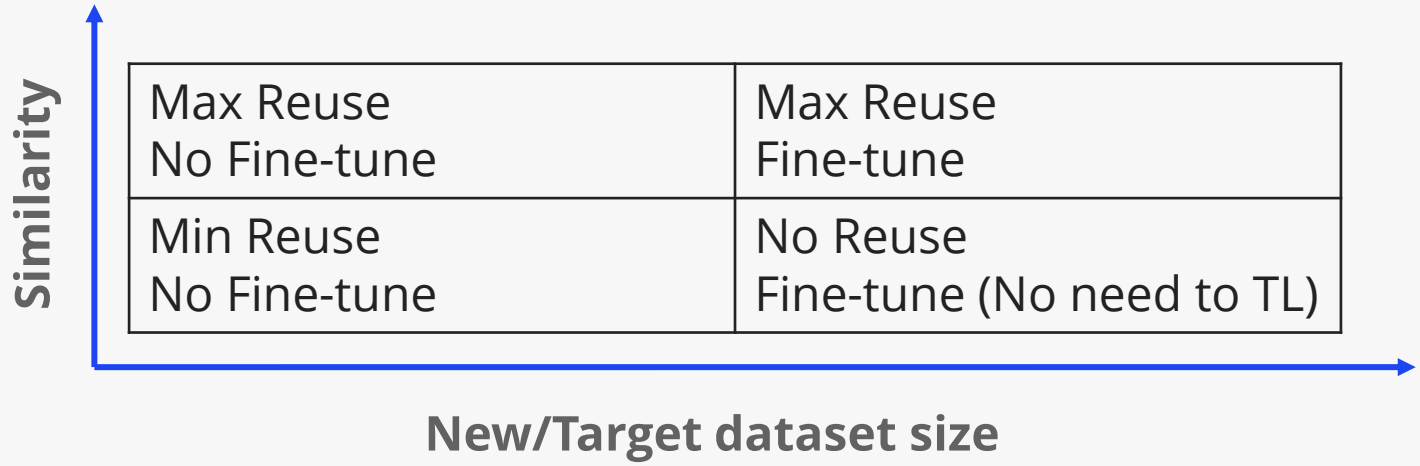


When Transfer Learning?

How do you decide what type of transfer learning you should perform on a new dataset? This is a function of several factors, two most important are:

1. The size of the new (target) dataset

2. Similarity to the original (source) dataset



Freeze vs. Fine-tune

Freeze or fine-tune?

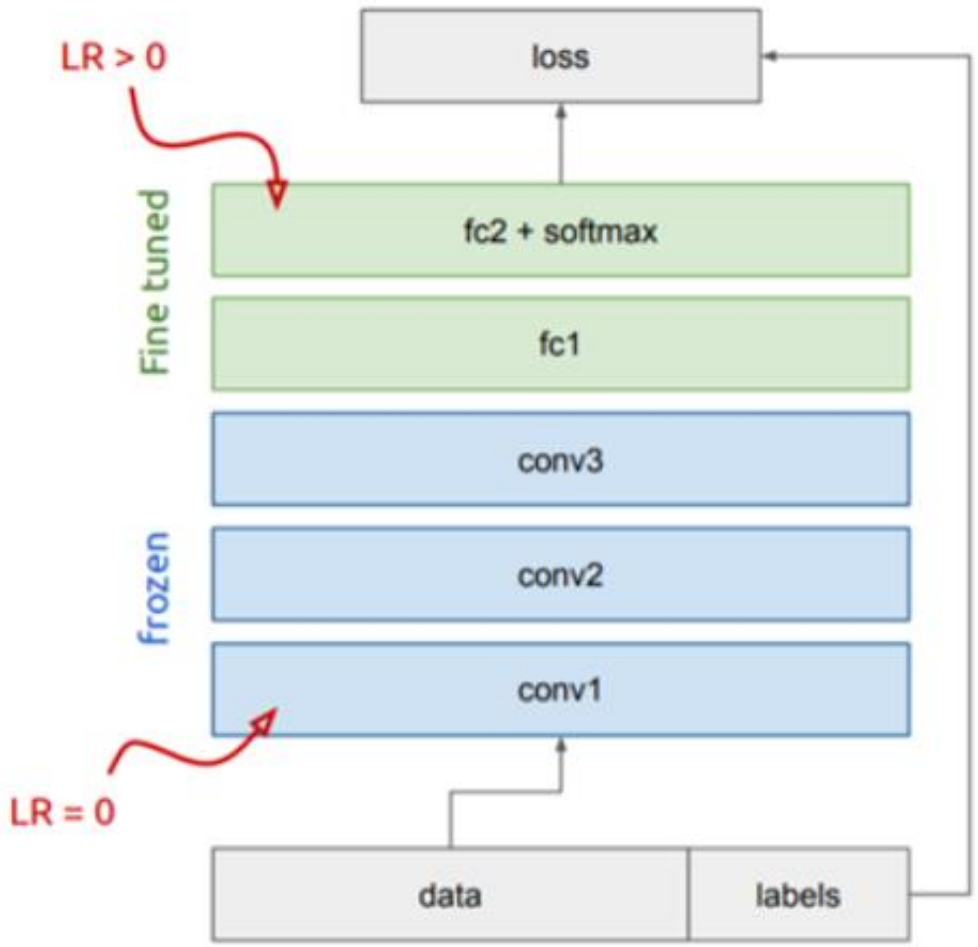
Bottom *n* layers can be frozen or fine tuned.

- **Frozen**: not updated during backprop
- **Fine-tuned**: updated during backprop

Which to do depends on target task:

- **Freeze**: target task labels are scarce, and we want to avoid overfitting
- **Fine-tune**: target task labels are more plentiful

In general, we can set learning rates to be different for each layer to find a tradeoff between freezing and fine tuning



Thank You!