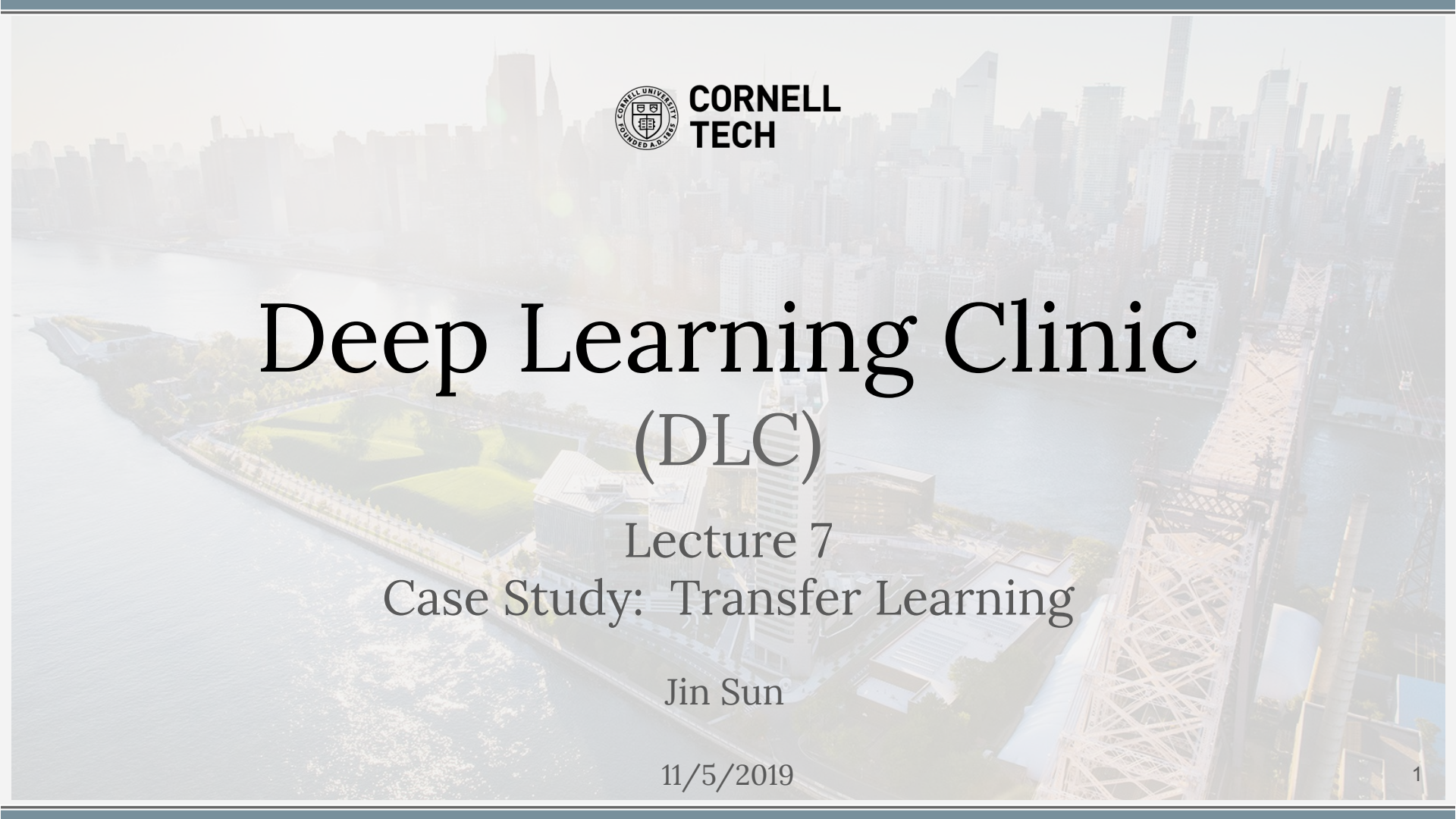


CORNELL TECH

Deep Learning Clinic (DLC)

Lecture 7
Case Study: Transfer Learning

Jin Sun

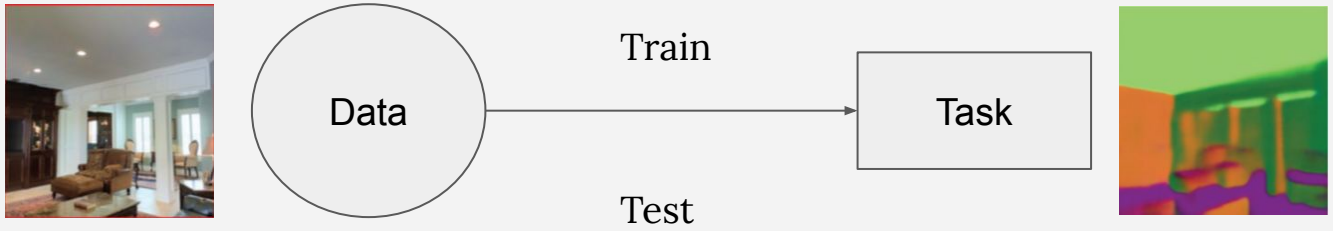
11/5/2019

Today - Transfer Learning

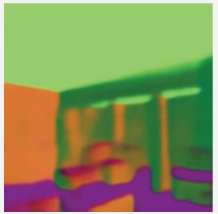
- **Overview**
- Fine-tuning
- Domain Transfer / Adaptation
- Multi-Task Learning
- Lifelong Learning

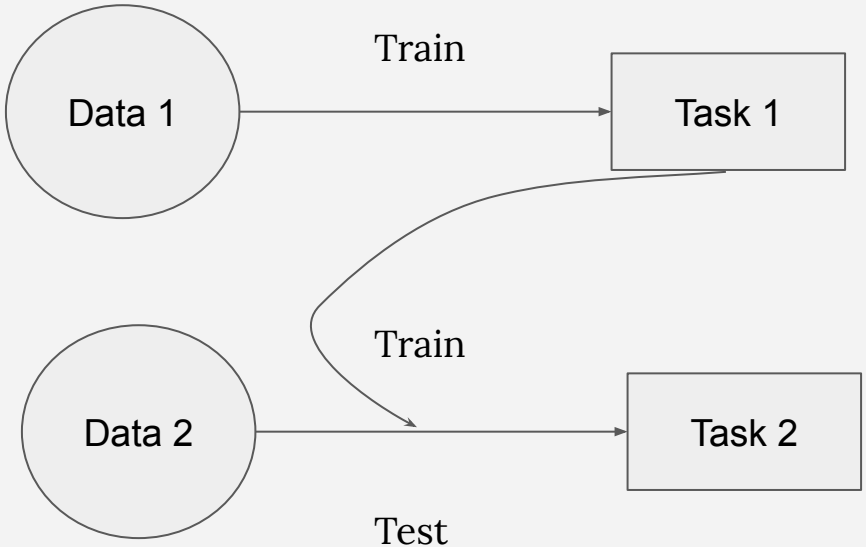
Overview

Standard Machine Learning Assumption



Fine-tuning





Domain Adaptation

Data 1

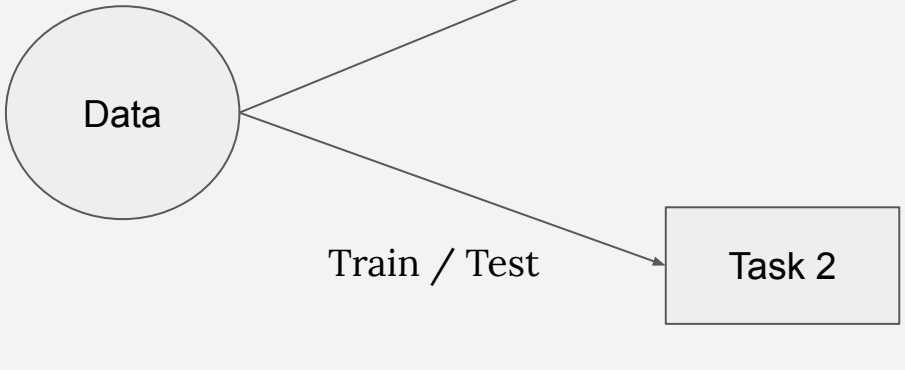
Train

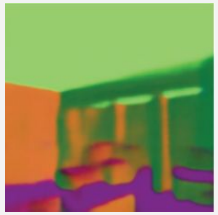
Task

Data 2

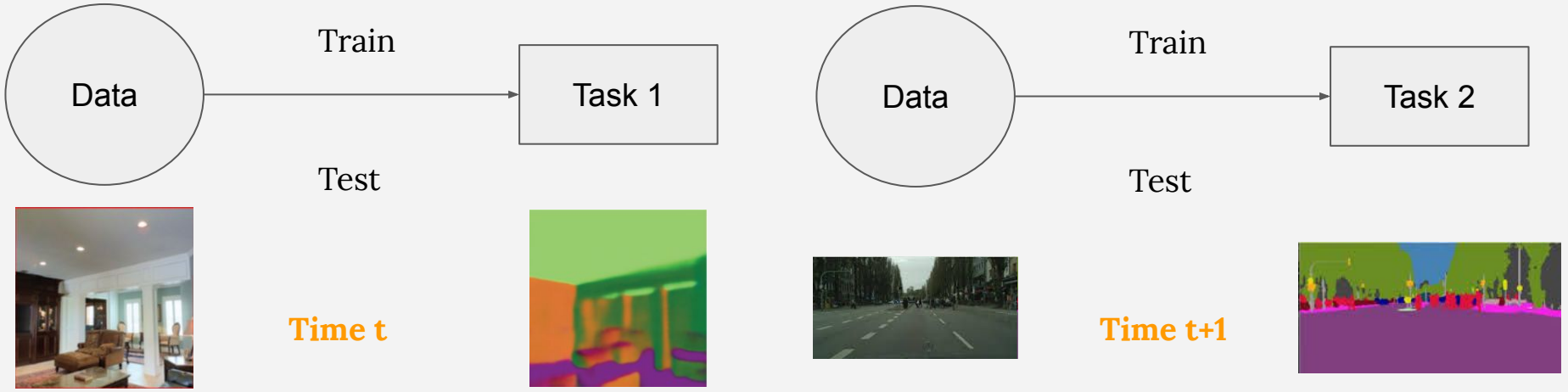
Test

Multi-task Learning

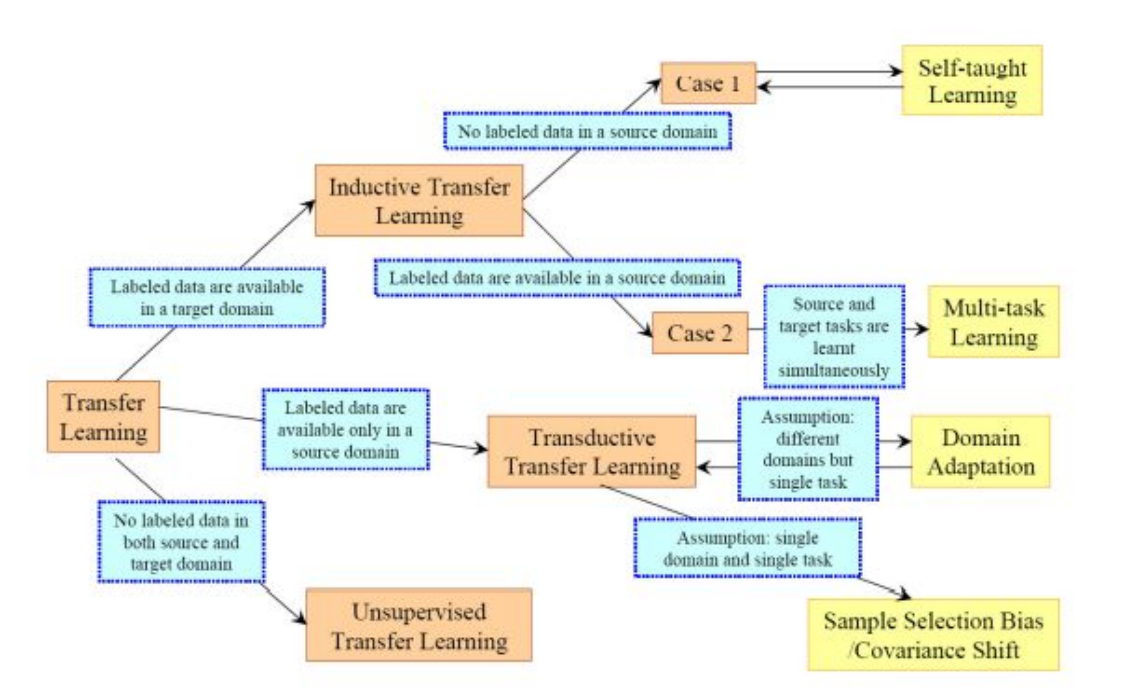




Lifelong Learning



Landscape



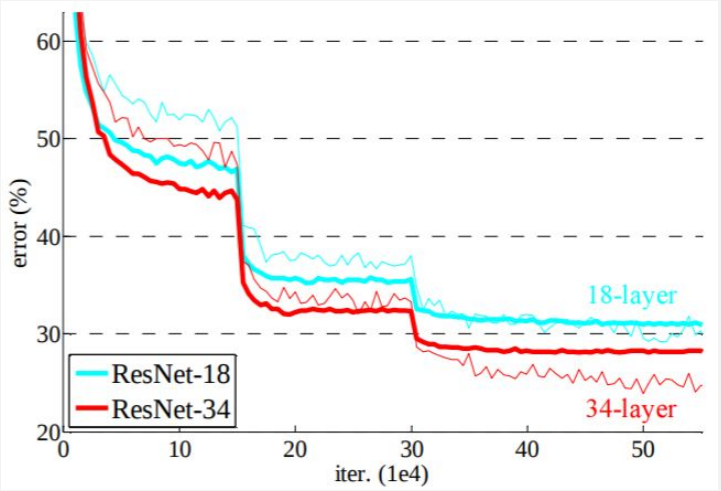
Pan, Sinno Jialin, and Qiang Yang. "A survey on transfer learning." *IEEE Transactions on knowledge and data engineering* 22.10 (2010)

Why Transfer Learning Is Important

Neural networks are generally hard to train

Our implementation for ImageNet follows the practice in [21, 41]. The image is resized with its shorter side randomly sampled in $[256, 480]$ for scale augmentation [41]. A 224×224 crop is randomly sampled from an image or its horizontal flip, with the per-pixel mean subtracted [21]. The standard color augmentation in [21] is used. We adopt batch normalization (BN) [16] right after each convolution and before activation, following [16]. We initialize the weights as in [13] and train all plain/residual nets from scratch. We use SGD with a mini-batch size of 256. The learning rate starts from 0.1 and is divided by 10 when the error plateaus, and the models are trained for up to 60×10^4 iterations. We use a weight decay of 0.0001 and a momentum of 0.9. We do not use dropout [14], following the practice in [16].

In testing, for comparison studies we adopt the standard 10-crop testing [21]. For best results, we adopt the fully-convolutional form as in [41, 13], and average the scores at multiple scales (images are resized such that the shorter side is in $\{224, 256, 384, 480, 640\}$).

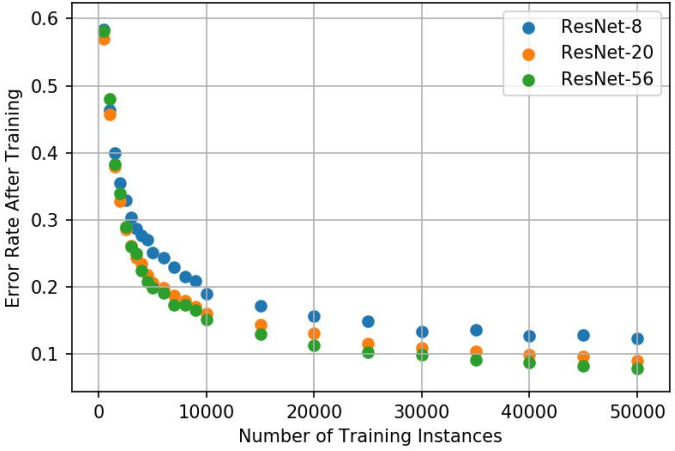


He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

Why Transfer Learning Is Important

Not enough data

ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93



http://seansoleyman.com/effect-of-dataset-size-on-image-classification-accuracy/

Why Transfer Learning Is Important

Multi-tasks serve as implicit regularization:

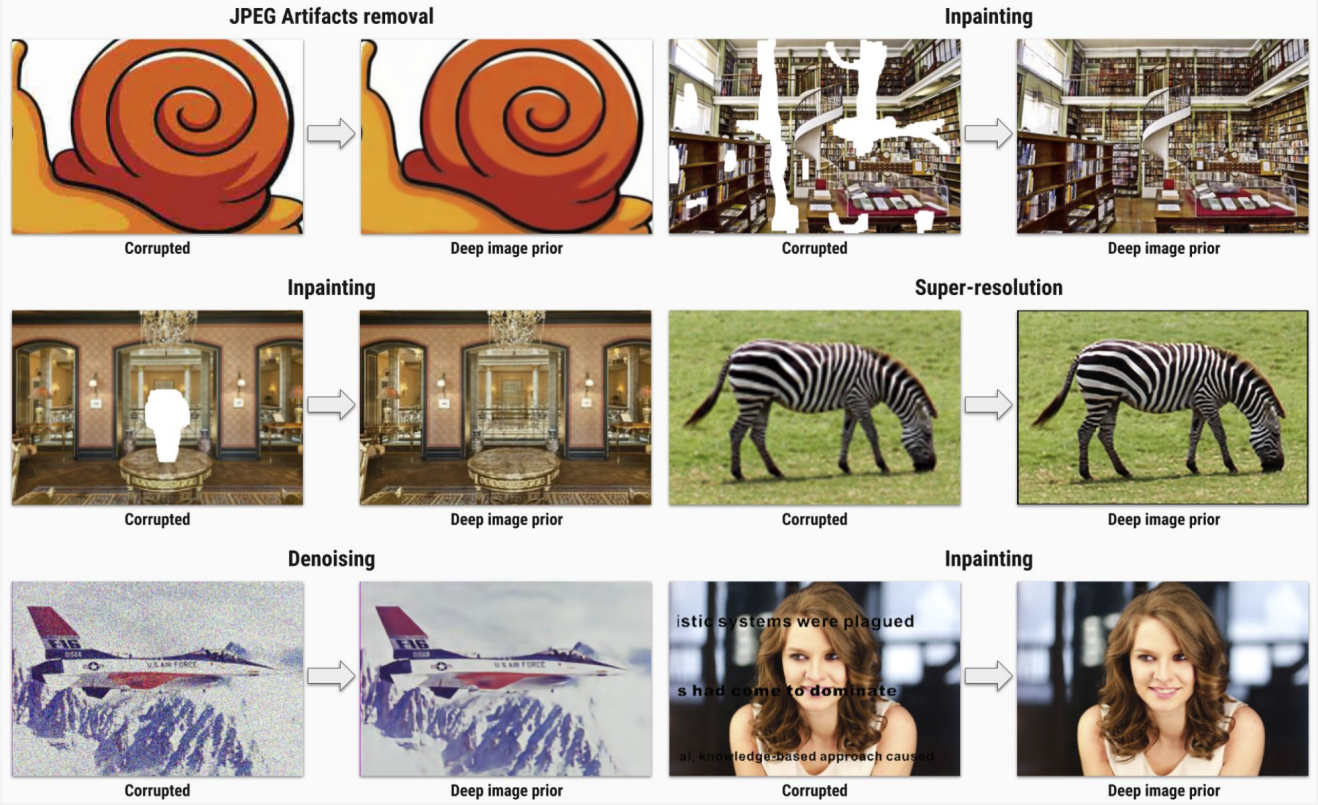
The other task's loss function can serve as a regularize term for the current task.

Model is forced not to overfit too much to one task.

Otherwise it will fail on remaining tasks.

[Assumption] Multi-tasks are sharing some network structures.

Neural Network Structure as Deep Image Prior



Ulyanov, Dmitry, Andrea Vedaldi, and Victor Lempitsky. "Deep image prior." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

Deep Image Prior

In image restoration problems the goal is to recover original image x having a corrupted image x_0. Such problems are often formulated as an optimization task:

$$\min_x E(x; x_0) + R(x)\,, \tag{1}$$

where $E(x; x_0)$ is a *data term* and $R(x)$ is an *image prior*. The data term $E(x; x_0)$ is usually easy to design for a wide range of problems, such as super-resolution, denoising, inpainting, while image prior $R(x)$ is a challenging one. Today's trend is to capture the prior $R(x)$ with a ConvNet by training it using large number of examples.

We first notice, that for a surjective $g : \theta \mapsto x$ the following procedure in theory is equivalent to (1):

$$\min_\theta E(g(\theta); x_0) + R(g(\theta))\,.$$

In practice g dramatically changes how the image space is searched by an optimization method. Furthermore, by selecting a "good" (possibly injective) mapping g, we could get rid of the prior term. We define $g(\theta)$ as $f_\theta(z)$, where f is a deep ConvNet with parameters θ and z is a fixed input, leading to the formulation

$$\min_\theta E(f_\theta(z); x_0)\,.$$

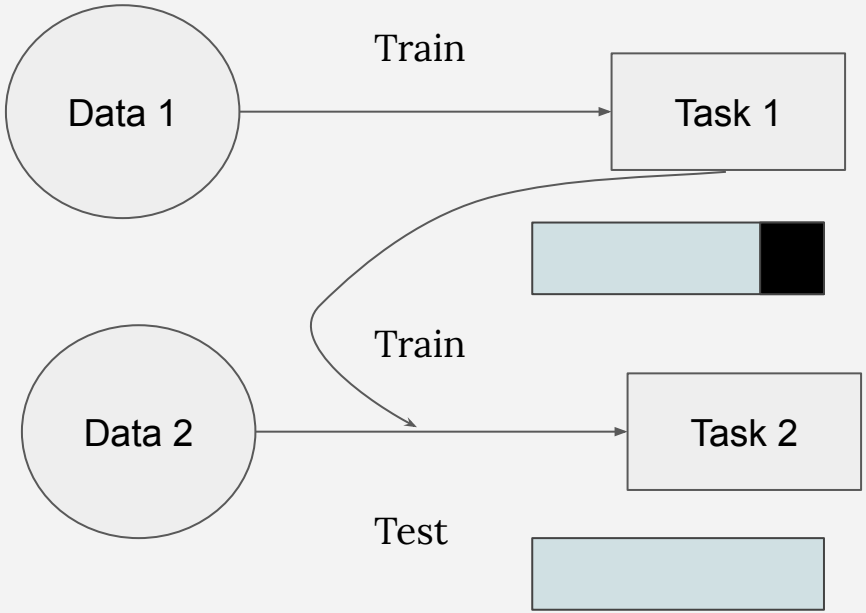
Here, the network f_θ is initialized randomly and input z is filled with noise and fixed.

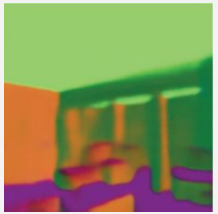
In other words, **instead of searching for the answer in the image space we now search for it in the space of neural network's parameters**. We emphasize that we never use a pretrained network or an image database. Only corrupted image x_0 is used in the restoration process.

Today - Transfer Learning

- Overview
- **Fine-tuning**
- Domain Transfer / Adaptation
- Multi-Task Learning
- Lifelong Learning

Fine-tuning





Fine-tuning Practice

1. Get the most powerful pre-trained recent neural network

   Resnet 18/50/101, Inception, VGG

2. Determine how many layers to keep

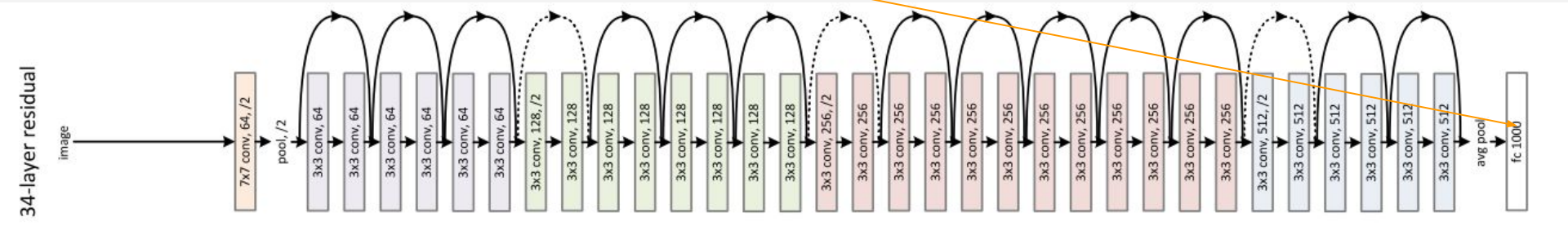
   Most common is the second to last layer, but you might need something different!

3. Add new task-dependent layers
4. Re-train the network using new data and hyperparameters

Example: Fine-tuning in Pytorch

https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

```
from torchvision import models
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 2)
```



Understand Model Structure in PyTorch

Each model is an instance of a class.

Layers can be called by their names:

`model.conv1, model.layer1, model.fc…`

```
In [16]: model
Out[16]:
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
```

Understand Model Structure in PyTorch

It is very flexible, you can partly ‘borrow’ a pre-trained model’s layer and make a new and more complex model.

```
class MyNet(nn.Module):
      def __init__(self, …):
            …
            resnet = models.resnet18(pretrained=True)

            self.firstconv = nn.Conv2d(num_in_channels, 64, kernel_size=(7,7), stride=(2,2),
padding=(3,3), bias=False) # can change the input layer too, if you are not using image as input
            …
            self.encoder1 = resnet.layer1
            self.myencoder = nn.Conv2d(128, 128, kernel_size=(3,3))
            …
      def forward(self, x):
            x = self.firstconv(x)
            …
            x = self.encoder1(x)
            x = self.myencoder(x)
```

Understand Model Structure in PyTorch

Or you can unzip all modules into a list and build another sequential model for simpler access.

```
resnet = models.resnet18(pretrained=True)
model = nn.Sequential(*(list(resnet.children())[:K])) # use K to control how many layers to keep
```

Then you can use the new model as any other sequential model.
For example the model can now be indexed:

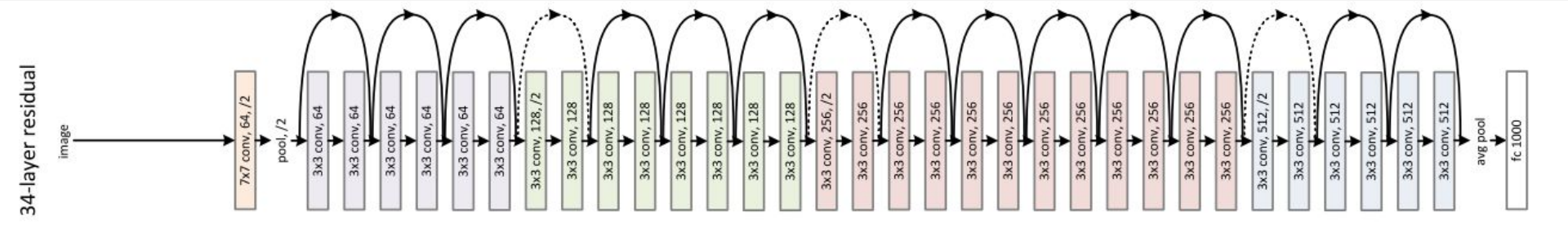
```
model[0], model[1], ...
```

Which Layers To Keep

Depends on what information is important for the new task

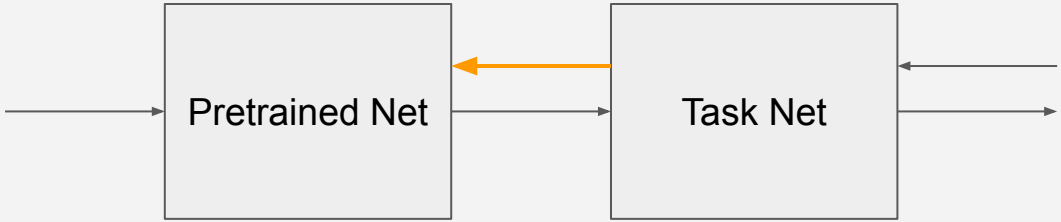
Low level → Mid level → High level

Edges → Object Parts → Objects



Freeze or Not-Freeze

Only when you have large enough new task data

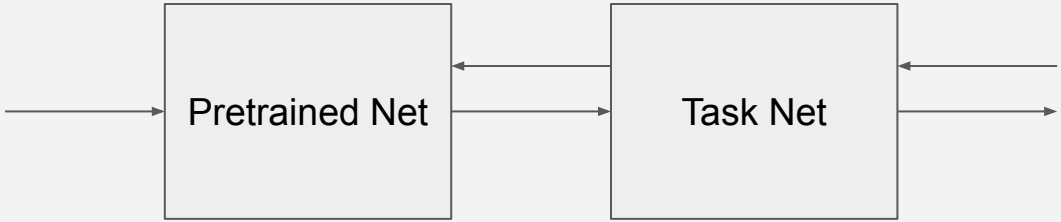


To implement in PyTorch:

```
for param in model.parameters():
    param.requires_grad = False
```

Freeze or Not-Freeze (Softer Version)

Different Learning Rate for Different Layer



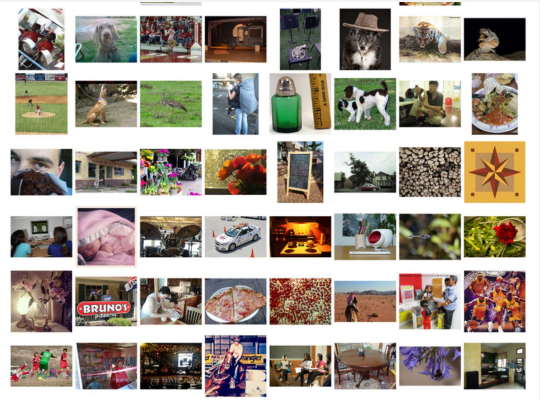
To implement in PyTorch:

```
optim.SGD([
                {'params': model.base.parameters()},
                {'params': model.classifier.parameters(), 'lr': 1e-3}
            ], lr=1e-2, momentum=0.9)
```

Understand The Limitation of Fine-tuning

A pre-trained model captures the statistics of the dataset it was trained on. In vision, that means ImageNet.

Take a look at the ImageNet dataset:



If your new task data distribution is vastly different from it,

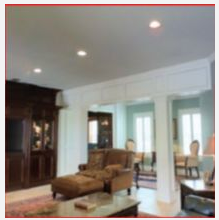
For example, non-visual data, or data from another spectrum.

... a pretrained model is acting like a good (?) initialization, at best.

Today - Transfer Learning

- **Overview**
- Fine-tuning
- **Domain Transfer / Adaptation**
- Multi-Task Learning
- Lifelong Learning

Domain Transfer / Adaptation



Data 1

Train

Task

Distribution Mismatch!

Data 2

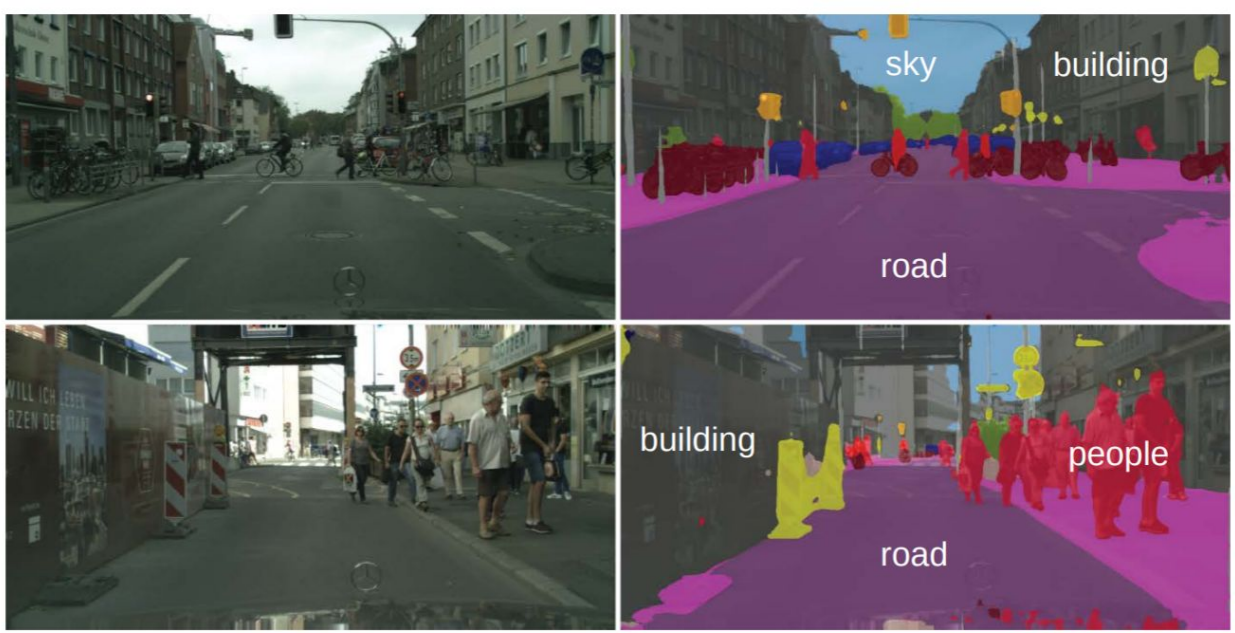
Test

Domain Transfer / Adaptation

Domain Shift Problem

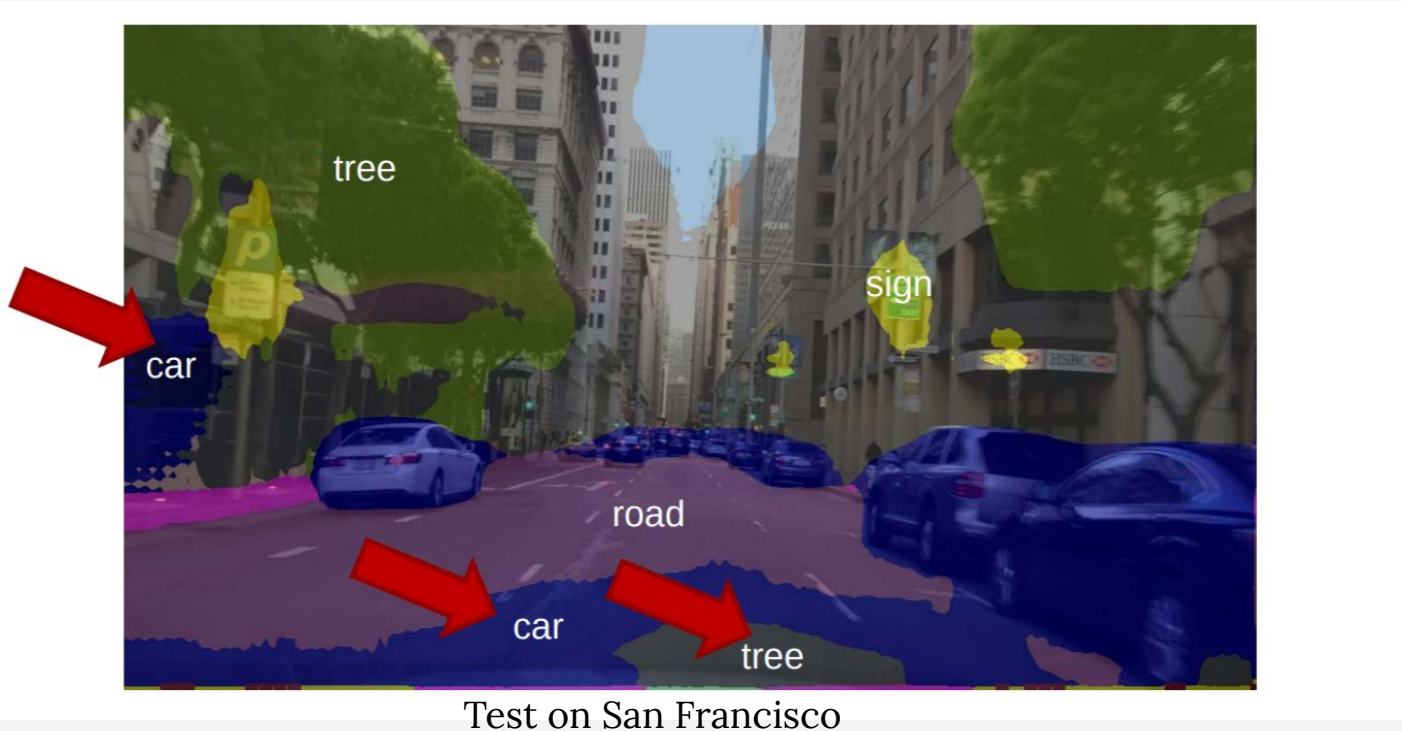
Slides from “What you saw is not what you get” Domain adaptation for deep learning, Kate Saenko

Domain Transfer / Adaptation



Slides from “What you saw is not what you get” Domain adaptation for deep learning, Kate Saenko

Domain Transfer / Adaptation



Test on San Francisco

Slides from “What you saw is not what you get” Domain adaptation for deep learning, Kate Saenko

Dataset Bias

task	Train on: \ Test on:	SUN09	LabelMe	PASCAL	ImageNet	Caltech101	MSRC	Self	Mean others	Percent drop
"car" classification	SUN09	**28.2**	29.5	16.3	14.6	16.9	21.9	28.2	19.8	**30%**
	LabelMe	14.7	**34.0**	16.7	22.9	43.6	24.5	34.0	24.5	**28%**
	PASCAL	10.1	25.5	**35.2**	43.9	44.2	39.4	35.2	32.6	**7%**
	ImageNet	11.4	29.6	36.0	**57.4**	52.3	42.7	57.4	34.4	**40%**
	Caltech101	7.5	31.1	19.5	33.1	**96.9**	42.1	96.9	26.7	**73%**
	MSRC	9.3	27.0	24.9	32.6	40.3	**68.4**	68.4	26.8	**61%**
	Mean others	10.6	28.5	22.7	29.4	39.4	34.1	53.4	27.5	48%
"car" detection	SUN09	**69.8**	50.7	42.2	42.6	54.7	69.4	69.8	51.9	**26%**
	LabelMe	61.8	**67.6**	40.8	38.5	53.4	67.0	67.6	52.3	**23%**
	PASCAL	55.8	55.2	**62.1**	56.8	54.2	74.8	62.1	59.4	**4%**
	ImageNet	43.9	31.8	46.9	**60.7**	59.3	67.8	60.7	49.9	**18%**
	Caltech101	20.2	18.8	11.0	31.4	**100**	29.3	100	22.2	**78%**
	MSRC	28.6	17.1	32.3	21.5	67.7	**74.3**	74.3	33.4	**55%**
	Mean others	42.0	34.7	34.6	38.2	57.9	61.7	72.4	44.8	**48%**

with the dataset? (answer key below)

Torralba, Antonio, and Alexei A. Efros. "Unbiased look at dataset bias." (2011): 1521-1528.

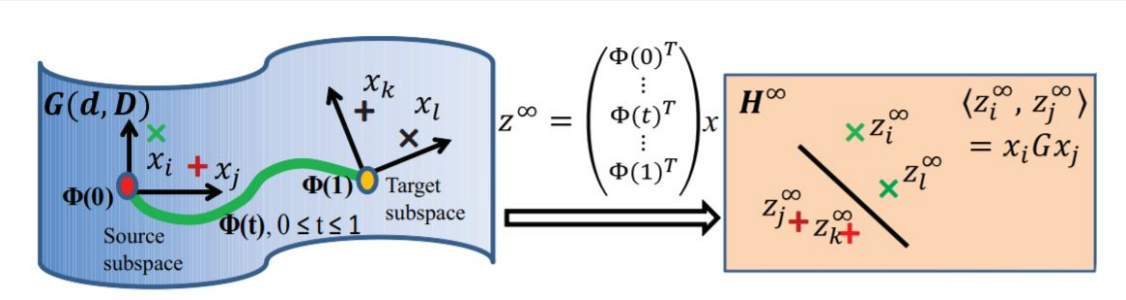
Dataset Bias

You can even name a dataset with high accuracy by looking at its samples.

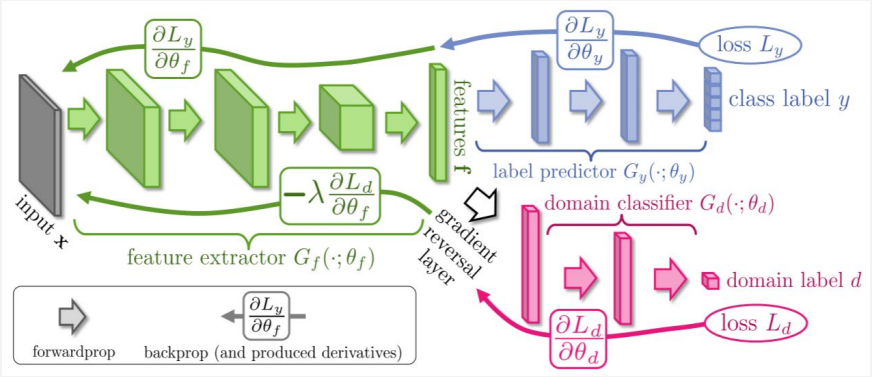
Figure 1. Name That Dataset: Given three images from twelve popular object recognition datasets, can you match the images with the dataset? (answer key below)

Torralba, Antonio, and Alexei A. Efros. "Unbiased look at dataset bias." (2011): 1521-1528.

Unsupervised Domain Alignment

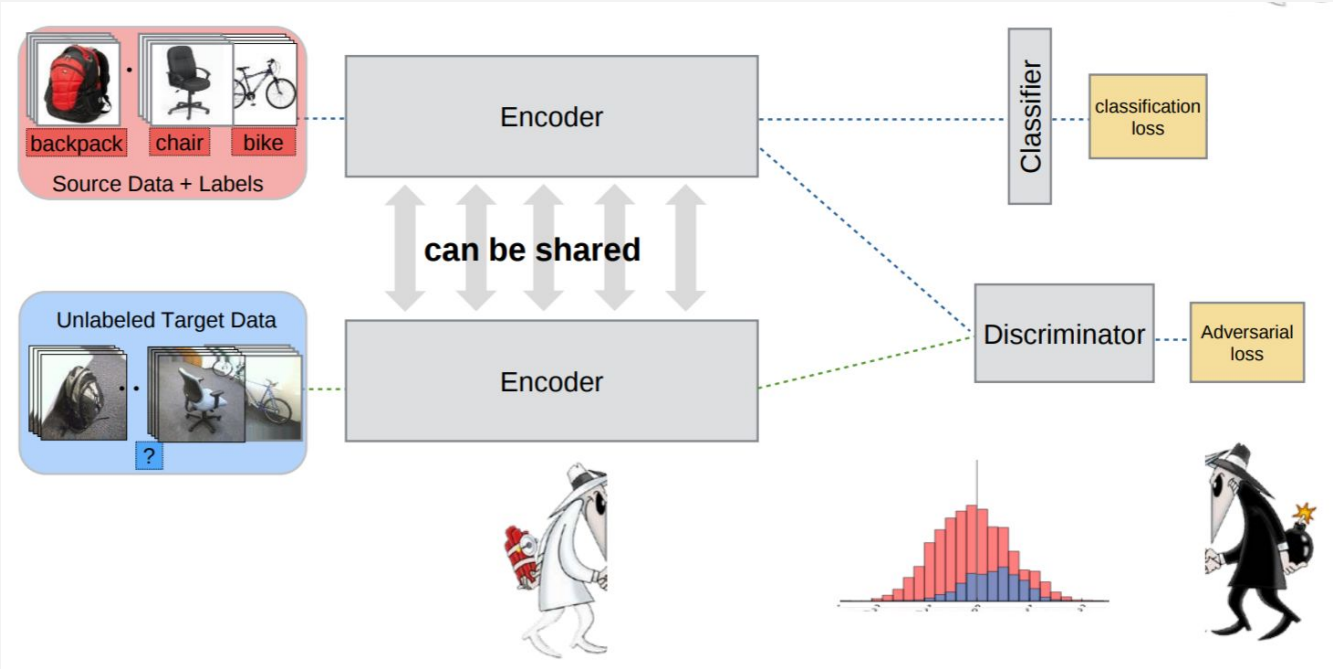


B. Gong, Y. Shi, F. Sha, and K. Grauman. Geodesic flow kernel for unsupervised domain adaptation. In CVPR, 2012



Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by back propagation. In ICML 2015

Adversarial Domain Alignment



Tzeng, Eric, et al. "Adversarial discriminative domain adaptation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

Slides from “What you saw is not what you get” Domain adaptation for deep learning, Kate Saenko

Another reason why we should care about domain transfer is:

Sometimes we can get a large amount of data with small cost from one domain than another.

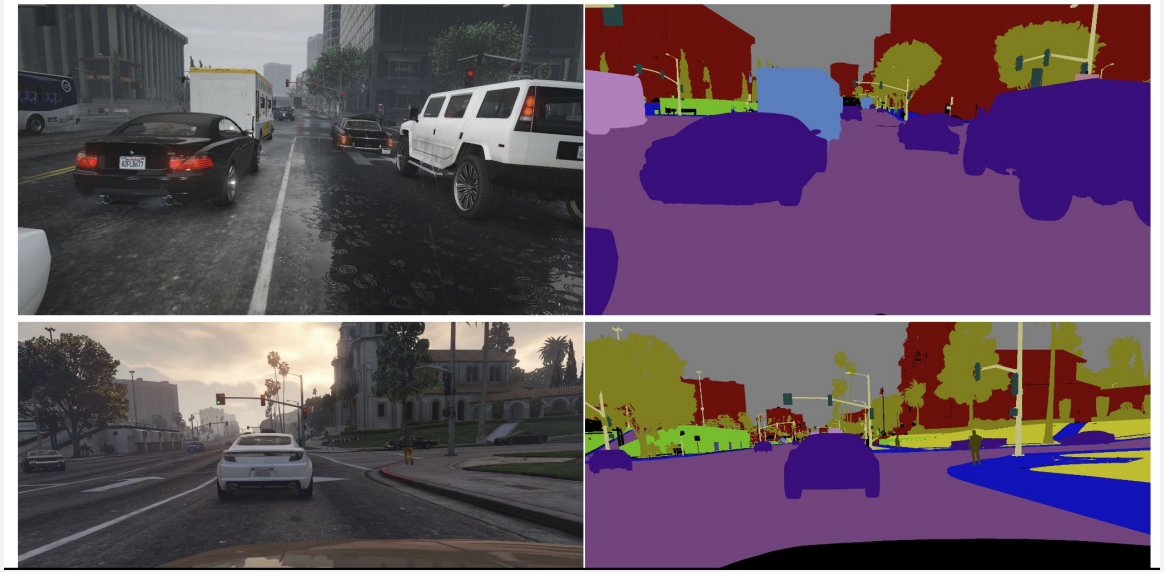
Synthetic Data that Helps Real World Application

Playing for Data: Ground Truth from Computer Games

Stephan Richter*[1] Vibhav Vineet*[2] Stefan Roth[1] Vladlen Koltun[2]

[1]TU Darmstadt [2]Intel Labs

* authors contributed equally



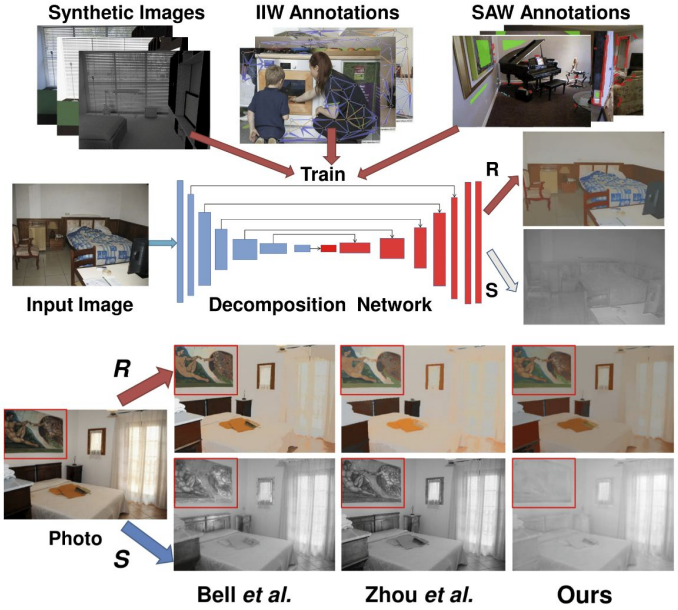
Synthetic Data that Helps Real World Application

CGIntrinsics: Better Intrinsic Image Decomposition through Physically-Based Rendering

Zhengqi Li Noah Snavely

Cornell University/Cornell Tech

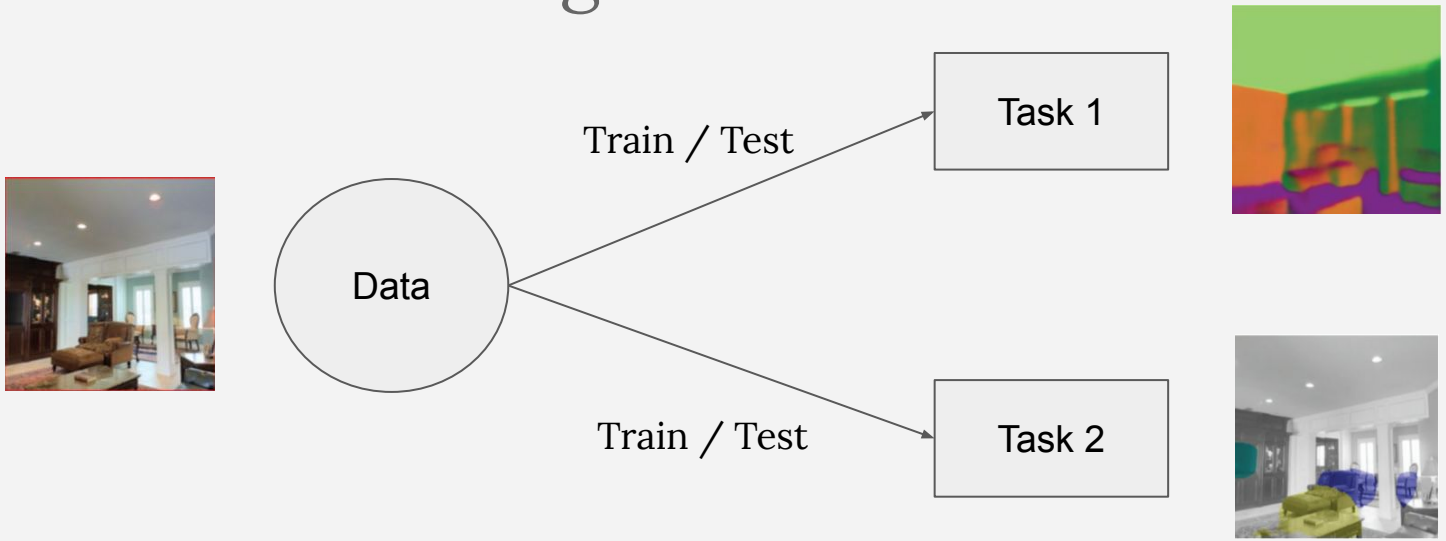
In ECCV, 2018



Today - Transfer Learning

- Overview
- Fine-tuning
- Domain Transfer / Adaptation
- **Multi-Task Learning**
- Lifelong Learning

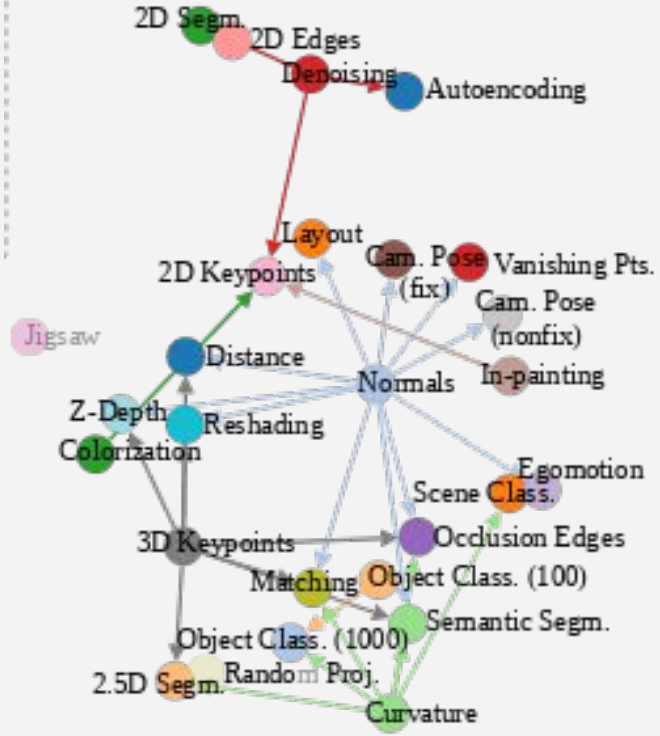
Multi-Task Learning



Many tasks are related to each other. For example: object detection and segmentation; depth estimation and surface normal estimation.

It makes sense to try to reason for those tasks at the same time.

Taskonomy



http://taskonomy.stanford.edu/

CVPR 2018 Best Paper

Taskonomy - Motivation

Question:

There are many seemly independent tasks in computer vision, how does one know whether they are indeed related or not?

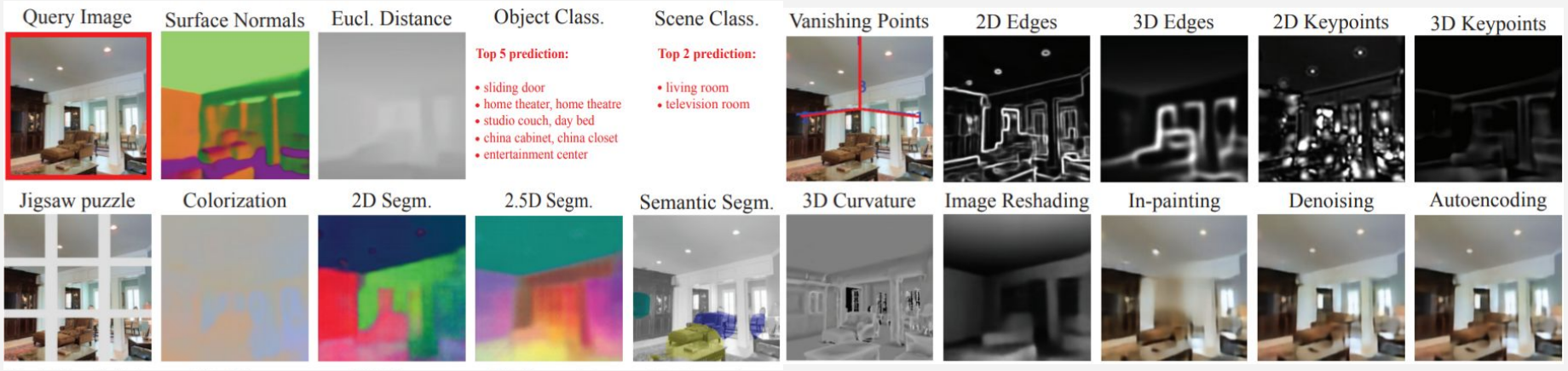
Goal:

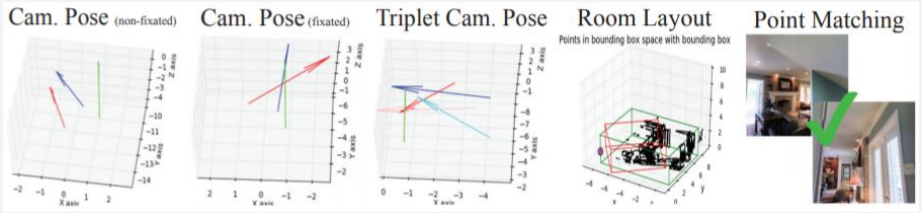
Learn the dependency relationships between various vision tasks.

Result:

Found transfer learning dependencies across a dictionary of twenty-six 2D, 2.5D, 3D, and semantic tasks.

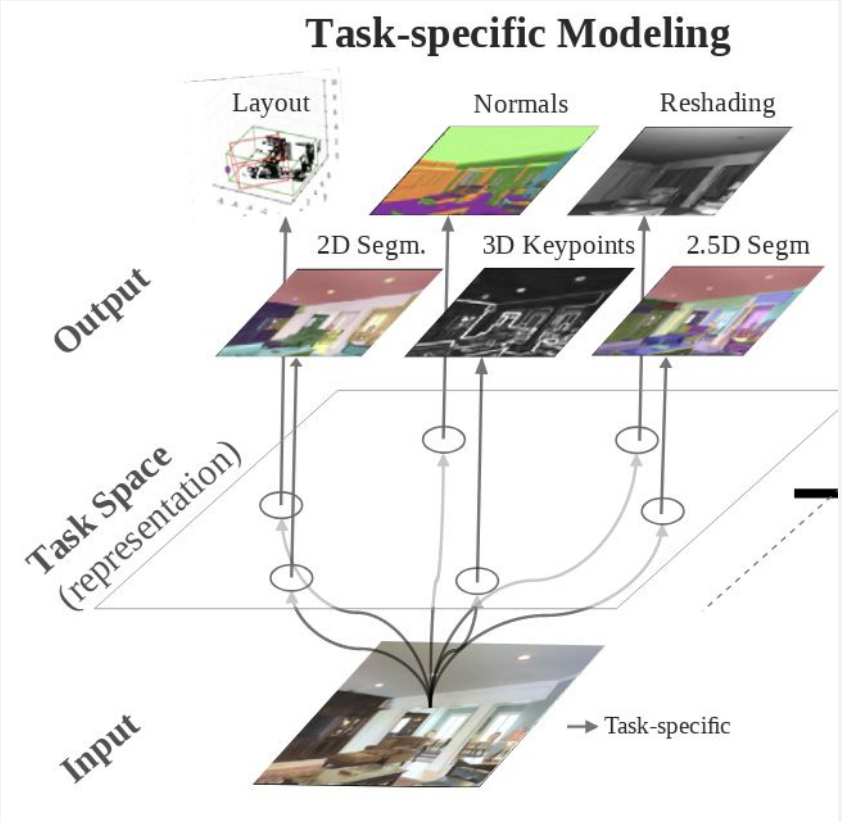
Taskonomy - Tasks





Taskonomy - Task Dependent Models

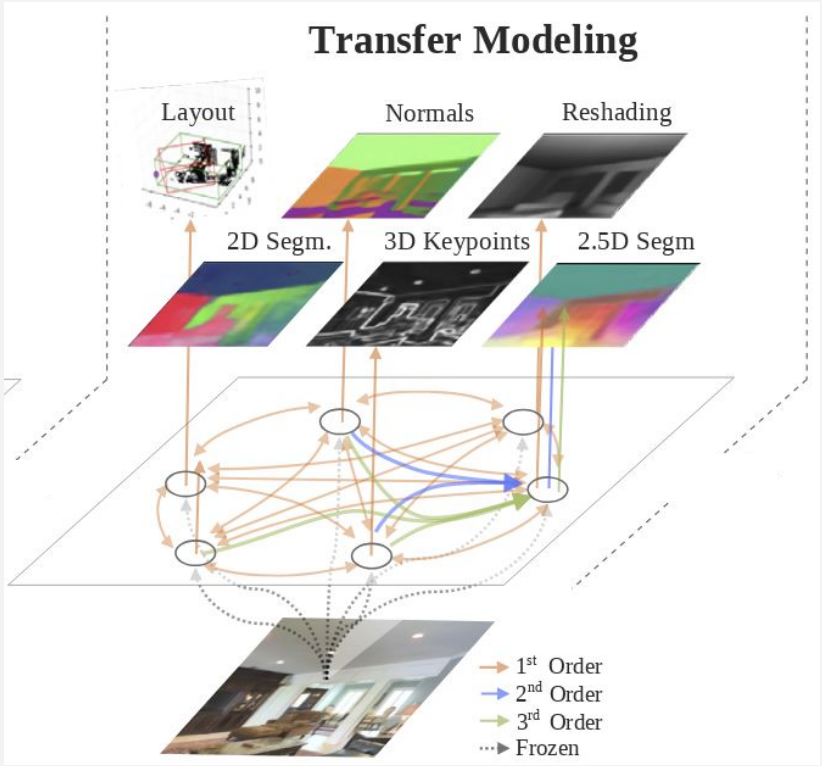
Each task is trained using an encoder-decoder structure.



Taskonomy - Task Transfer Modeling

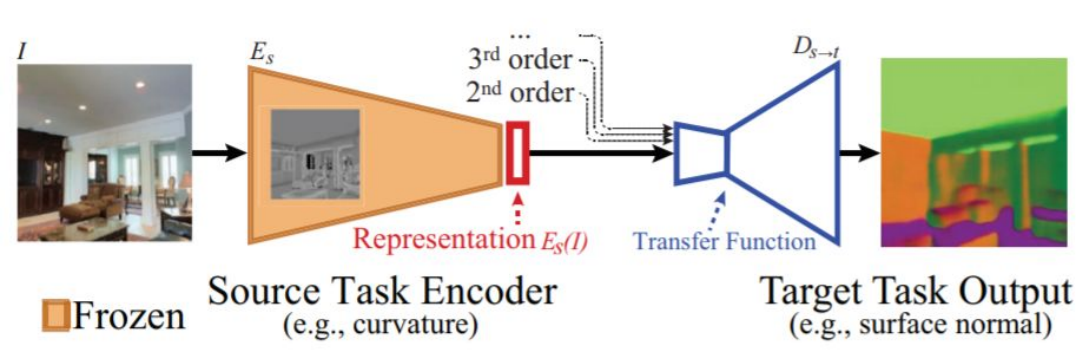
Features extracted from encoders of other tasks are used for each task.

Nth order means N other tasks are used.



Taskonomy - Task Transfer Modeling

A closer look at the transfer module.



No more than 5th order transfer is used.

Taskonomy - Affinity Matrix

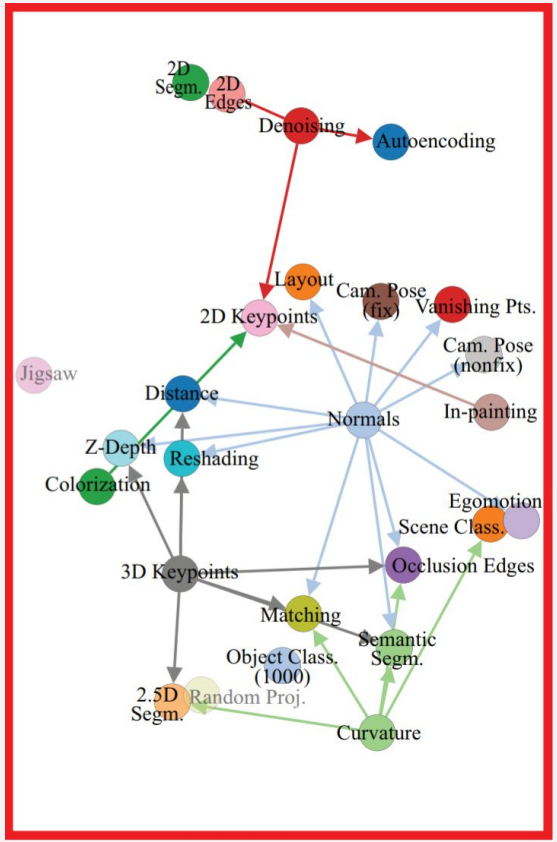
Taskonomy Generation

Given the affinity matrix, build a graph where tasks are nodes and transfers are edges.

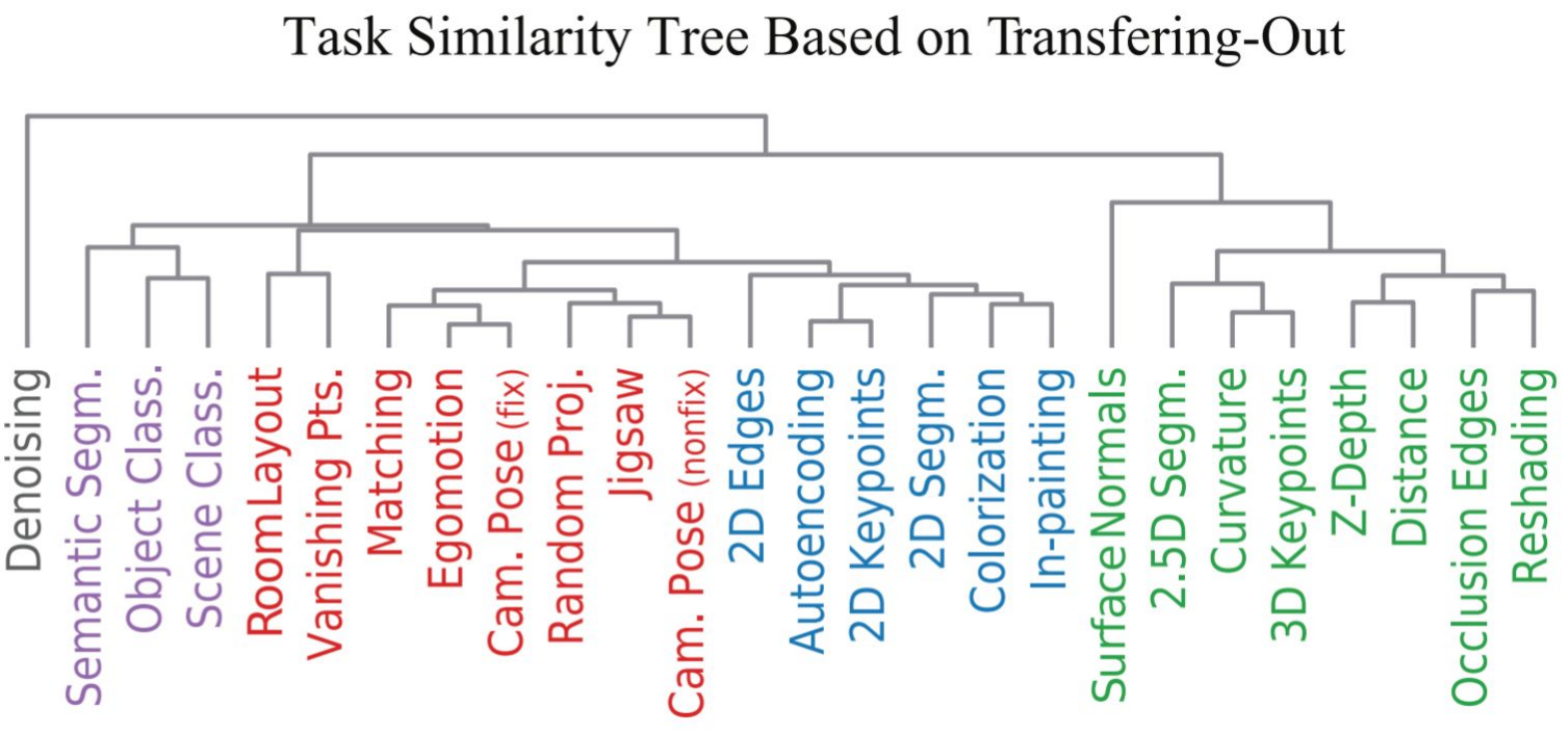
Find a graph that has maximum performance with minimum costs (number of supervision used) -> Boolean Integer Programming

Taskonomy - Main Result

[Live version]



Taskonomy - Main Result



Taskonomy - Summary

Many vision tasks are correlated to each other than originally believed.

Number of labeled data samples needed to solve for N tasks can be greatly reduced if we transfer features from one task to another.

Open Questions:

1. The results are model and data specific.
2. How about non visual tasks (for example robot manipulation).
3. Lifelong learning scenario where models are constantly updating.

Taskonomy - Website and API

http://taskonomy.stanford.edu/api/

https://github.com/StanfordVL/taskonomy/tree/master/taskbank

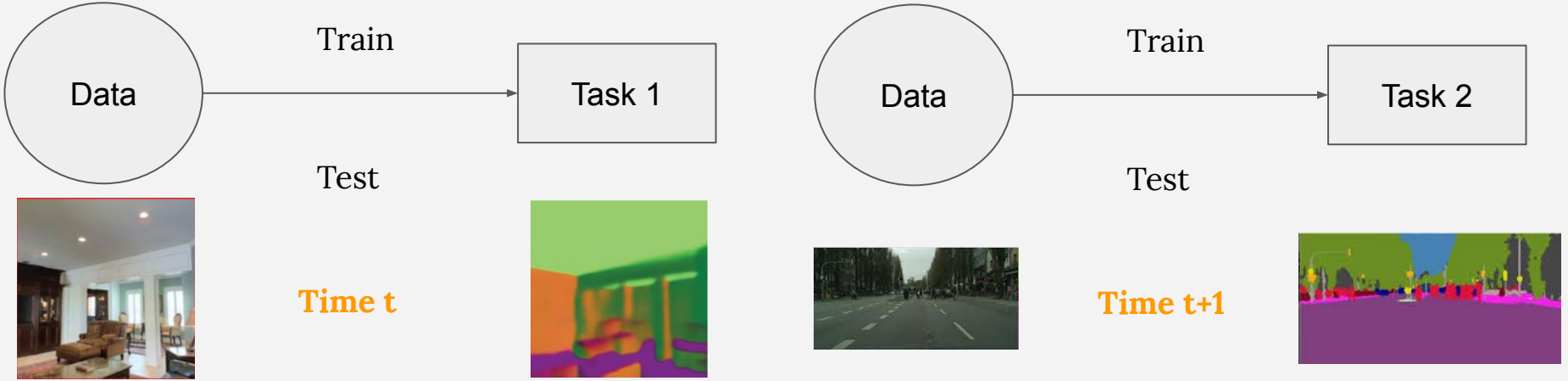
Data is huge: 4.5 million images, 11 TB!

https://github.com/StanfordVL/taskonomy/tree/master/data

Today - Transfer Learning

- Overview
- Fine-tuning
- Domain Transfer / Adaptation
- Multi-Task Learning
- **Lifelong Learning**

Lifelong Learning



Both data and model are changing over time.

How to learn new tasks without forgetting about old tasks?

iCaRL: Incremental Classifier and Representation Learning

Goal:

Learn new data on a single model that has limited access to old train data.

Key Ideas:

Exemplar set of previously seen classes.

Combine loss from training on both new and old data.

Rebuffi, Sylvestre-Alvise, et al. "icarl: Incremental classifier and representation learning." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

iCaRL: Incremental Classifier and Representation Learning

Algorithm 2 iCaRL INCREMENTALTRAIN

input $X^s, \ldots, X^t$ // training examples in per-class sets
input K // memory size
require Θ // current model parameters
require $\mathcal{P} = (P_1, \ldots, P_{s-1})$ // current exemplar sets
  $\Theta \leftarrow$ UPDATEREPRESENTATION$(X^s, \ldots, X^t; \mathcal{P}, \Theta)$
  $m \leftarrow K/t$ // number of exemplars per class
  for $y = 1, \ldots, s-1$ **do**
    $P_y \leftarrow$ REDUCEEXEMPLARSET(P_y, m)
  end for
  for $y = s, \ldots, t$ **do**
    $P_y \leftarrow$ CONSTRUCTEXEMPLARSET(X_y, m, Θ)
  end for
  $\mathcal{P} \leftarrow (P_1, \ldots, P_t)$ // new exemplar sets

Summary - Transfer Learning

- Overview
- Fine-tuning
- Domain Transfer / Adaptation
- Multi-Task Learning
- Lifelong Learning