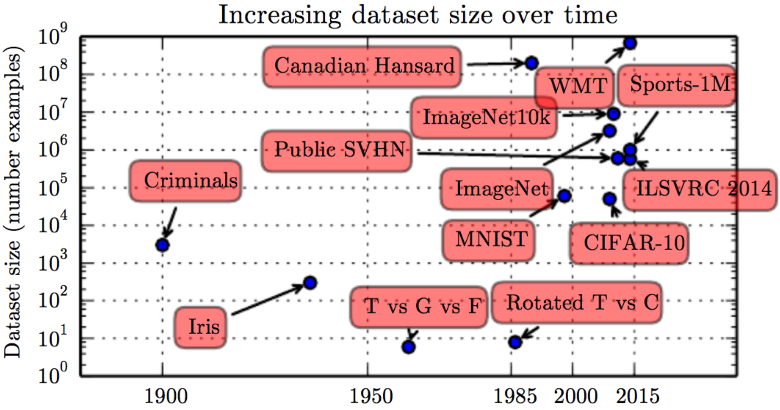
Transfer Learning

Vincent Barra
LIMOS, UMR 6158 CNRS, Université Clermont Auvergne

Lack of unlabeled data

Can we do Deep Learning with few labeled data?

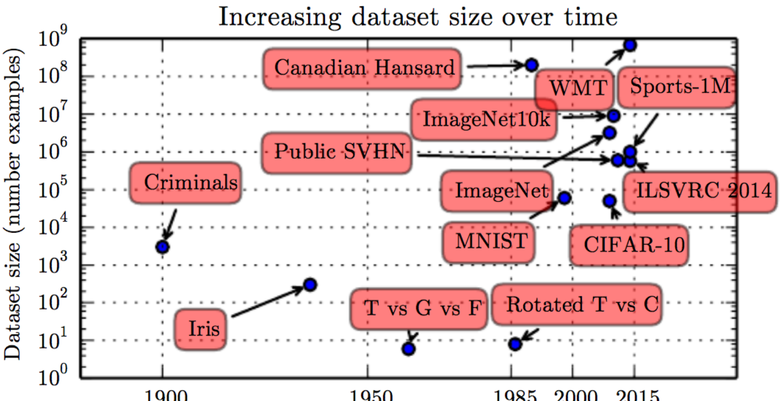
- Learn useful representation from unlabeled data
- Train on a nearby surrogate objective for which it is easier to generate labels



Lack of unlabeled data

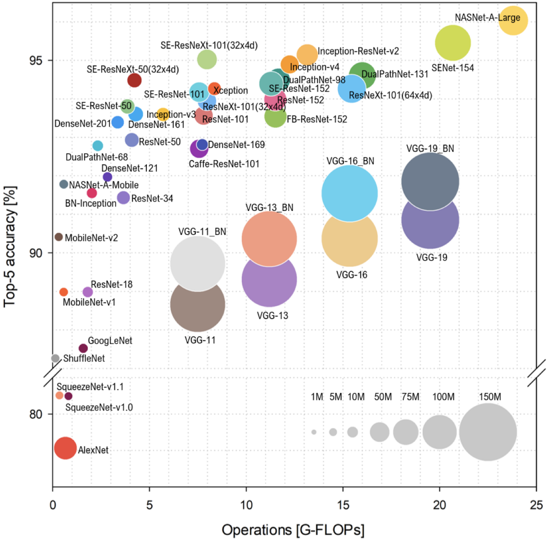
Can we do Deep Learning with few labeled data?

- Learn useful representation from unlabeled data
- Train on a nearby surrogate objective for which it is easier to generate labels
- Transfer learned representation from a related task



Transfer Learning

Training a Deep net from scratch on your dataset can be hard

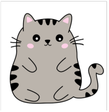


Transfer Learning

Initial task/domain

Same domain

Same task



Transfer Learning

Concept

- Several networks have already been trained on a different domain for a different source task
- Adapt this network to the target class

Many variations

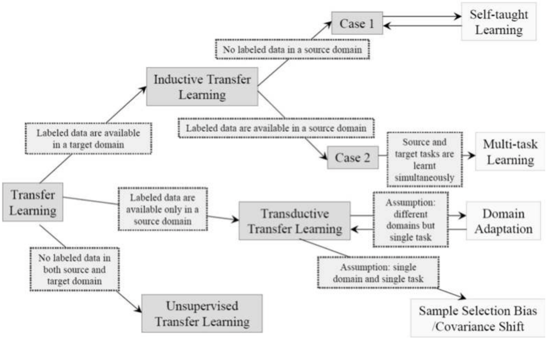
- Close domain, different task
- Different domain, same task
- Partial/full adaptation

See Lecture

"CNN Architectures"

Transfer Learning

A taxonomy of transfer Learning



Source: Pang & Yang, TKDE 2010)

We concentrate on domain adaptation/multi-task learning

Domain adaptation

Domain adaptation

- Domains are modeled as probability distributions over an instance space
- Task associated to a domain (classification, regression..)

The question

How can we learn a low-error classifier on a target data distribution, using labeled data from a source distribution ?

Domain adaptation

- $\mathcal{X}_S, \mathcal{X}_T$: source/target domain
- $\mathcal{Y}_S, \mathcal{Y}_T$: source/target label space

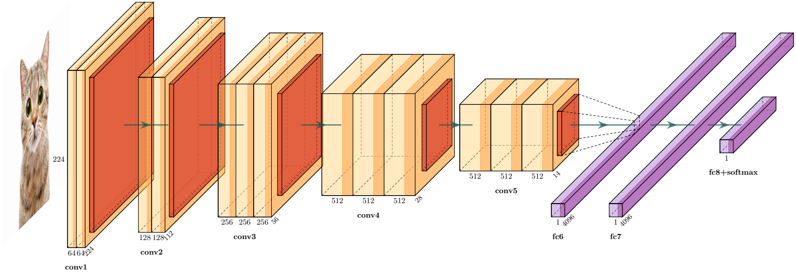
What can happen

- Data distribution change from $\mathcal{X}_S$ to $\mathcal{X}_T$ or $\mathbb{P}_S(x) \neq \mathbb{P}_T(x)$.
- Conditional probabilities may be different: $\mathcal{Y}_S \neq \mathcal{Y}_T$ or $\mathbb{P}_S(y|x) \neq \mathbb{P}_T(y|x)$



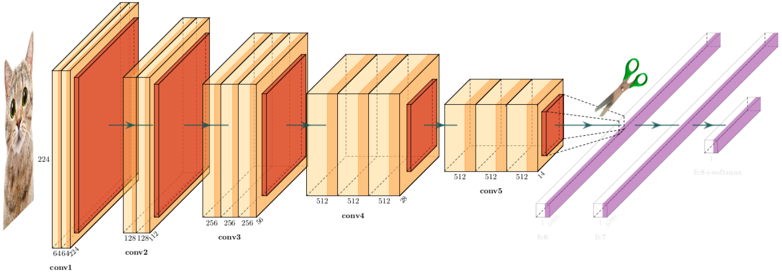

Transfer Learning

Take an already trained network



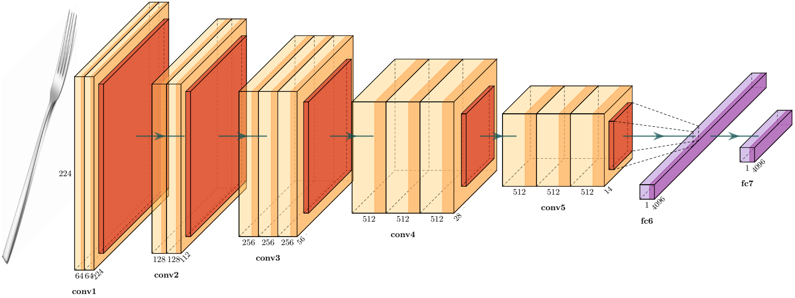
Transfer Learning

Use output of one or more layers as feature detector

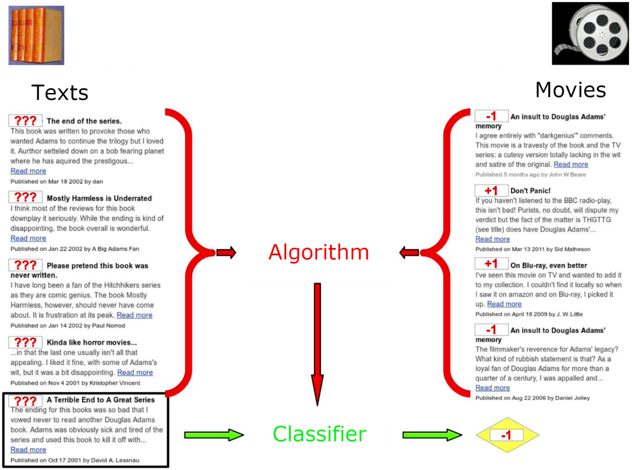


Transfer Learning

Train a new classifier on these features



Not only in Computer Vision



Transfer Learning

It works well...

... But can we do better than off-the-shelf features ?

Transfer Learning

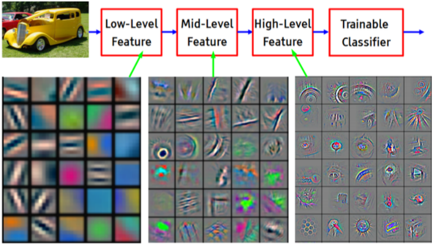
It works well...

... But can we do better than off-the-shelf features ?

Fine tuning

- Change the classification layer to match the problem
- Retrain some/all layers of the whole network

Fine tuning



What layers to choose ?

- In computer vision
  - First layers detect simpler and more general patterns
  - Deeper layers capture more specific patterns related to data

$\Rightarrow$ Allow the last block(s) of convolution/pooling to be retrained.

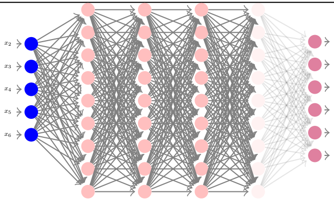
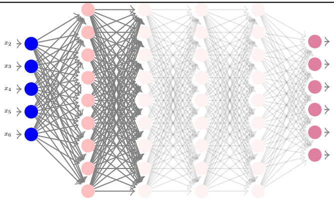
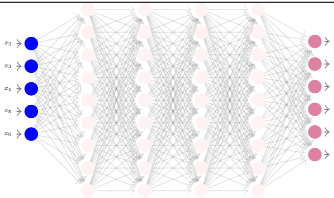
- In sequential data: may keep the last few layers

Fine tuning

Freeze / Finetune

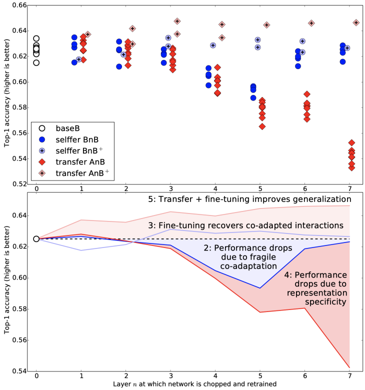
- Frozen layers: not updated during backpropagation
- Finetuned layers: updated during backpropagation
- Depends on the target task:
  - freeze if target task labels are scarce and no overfitting
  - finetune if more target labels.

→ set learning rates to be different for each layer to find a tradeoff

Taille des données	Réseau	
Petite		Geler toutes les couches, entraîner le nouveau classifieur
Moyenne		Geler certaines couches
Grande		Geler toutes les couches. Initialisation avec les poids pré-entraînés.

FINE TUNING

- If sufficient examples are available, fine tuning improves generalization
- Transfer learning and fine tuning can serve as an initialization process
- Very often better performance than training from scratch



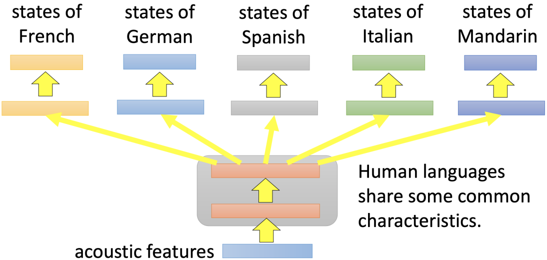
Yosinki et al, NIPS 2014

Source: 500 classes from ImageNet

Target, another 500 classes from ImageNet.

Multitask learning

The multilayer architecture of Deep Neural Network makes them suitable for multitask learning.



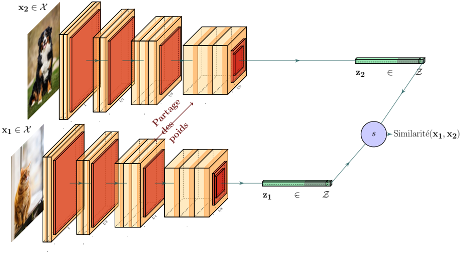
Huang & al, ICASSP 2013.

The training task can also depend on the number of training examples in the target domain.

- Sufficient number of examples : OK
- No examples: unsupervised domain adaptation
- Few number of examples (Few-shot learning)
  - Embedding learning
  - Data augmentation
  - Data generation
  - Semi supervised domain adaptation

Embedding learning

Embedding learning: siamese, triplet network...



See lecture

Matching networks

Data augmentation

Given one example, generate n new ones using transformations (rotation, scaling, noise adding, nonlinear transformations, color processing...)

Data generation

Model the data distribution into the target space in order to be able to generate new and unseen samples.

- Generative Adversarial Networks (GANs)
- Variational Autoencoders
- ...

See lecture

Generative models

Un- or Semi-supervised domain adaptation

- Matching source distributions
- Combination of fine tuning & unsupervised adaptation

Out of the scope of this lecture.

TRAINABLE/NON TRAINABLE

$$y = \boldsymbol{W}^T x + b$$

```
dense = tf.keras.layers.Dense(7)
dense.build((None,5))
dense.trainable=False
print("weights:", len(dense.weights))
print("trainable_weights:", len(dense.trainable_weights))
print("non_trainable_weights:", len(dense.non_trainable_weights))
print(dense.weights)
```

```
weights: 2
trainable_weights: 0
non_trainable_weights: 2
[<tf.Variable 'kernel:0' shape=(5, 7) dtype=float32, numpy=
array([[ 0.19174272,  0.1961686 , -0.5426951 ,  0.27109373,  0.23231155,
         0.18705916, -0.15816867],
       [ 0.6983326 ,  0.702092  ,  0.67835873, -0.2961814 , -0.49993056,
        -0.6466697 ,  0.3440327 ],
       [-0.1576004 ,  0.1708141 ,  0.01579124, -0.5328252 , -0.27179474,
        -0.07055551, -0.7060735 ],
       [ 0.03460306,  0.03050655,  0.46395332, -0.29719362,  0.4411903 ,
        -0.17896783, -0.59996223],
       [-0.48393145,  0.38511735, -0.43845502, -0.485897  ,  0.6990538 ,
        -0.6045918 ,  0.4238811 ]], dtype=float32)>, <tf.Variable 'bias:0' shape=(7,) dtype=float32, numpy=array([0., 0., 0., 0., 0., 0., 0.], dtype=float32)>]
```

Trainable/non trainable

$$y = \boldsymbol{W}^T x + b$$

```
dense = tf.keras.layers.Dense(7)
dense.build((None,5))
dense.trainable=False
print("weights:", len(dense.weights))
print("trainable_weights:", len(dense.trainable_weights))
print("non_trainable_weights:", len(dense.non_trainable_weights))
print(dense.weights)
```

```
weights: 2
trainable_weights: 0
non_trainable_weights: 2
[<tf.Variable 'kernel:0' shape=(5, 7) dtype=float32, numpy=
array([[ 0.19174272,  0.1961686 , -0.5426951 ,  0.27109373,  0.23231155,
         0.18705916, -0.15816867],
       [ 0.6983326 ,  0.702092  ,  0.67835873, -0.2961814 , -0.49993056,
        -0.6466697 ,  0.3440327 ],
       [-0.1576004 ,  0.1708141 ,  0.01579124, -0.5328252 , -0.27179474,
        -0.07055551, -0.7060735 ],
       [ 0.03460306,  0.03050655,  0.46395332, -0.29719362,  0.4411903 ,
        -0.17896783, -0.59996223],
       [-0.48393145,  0.38511735, -0.43845502, -0.485897  ,  0.6990538 ,
        -0.6045918 ,  0.4238811 ]], dtype=float32)>, <tf.Variable 'bias:0' shape=(7,) dtype=float32, numpy=array([0., 0., 0., 0., 0., 0., 0.], dtype=float32)>]
```

In Keras

```
# Load a pretrained model
pretrained_model = tf.keras.applications.MobileNetV2(input_shape=[*image_size, 3],
                                                      include_top=False)

# Suppress the classification layer
pretrained_model.layers.pop()

# Freeze all or a subset of weights
for layer in pretrained_model.layers[:-stop_freeze]:
    layer.trainable = False

# Put our own classification layers
model = tf.keras.Sequential([pretrained_model,tf.keras.layers.Flatten(),
                             tf.keras.layers.Dense(5, activation='softmax')])
```

In Keras

Transfer Learning + Fine tuning

```
# Transfer Learning
pretrained_model = keras.applications.MobileNetV2(input_shape=[*image_size,3],
                                                   include_top=False)  # no classifier at the top.
pretrained_model.trainable = False

model = tf.keras.Sequential([pretrained_model, tf.keras.Flatten(),tf.keras.layers.Dense(5,activation='softmax')])
model.compile...
model.fit...

# Fine tuning
pretrained_model.trainable = True
model.compile... # Important to call it again if Trainable is changed
model.fit...
```