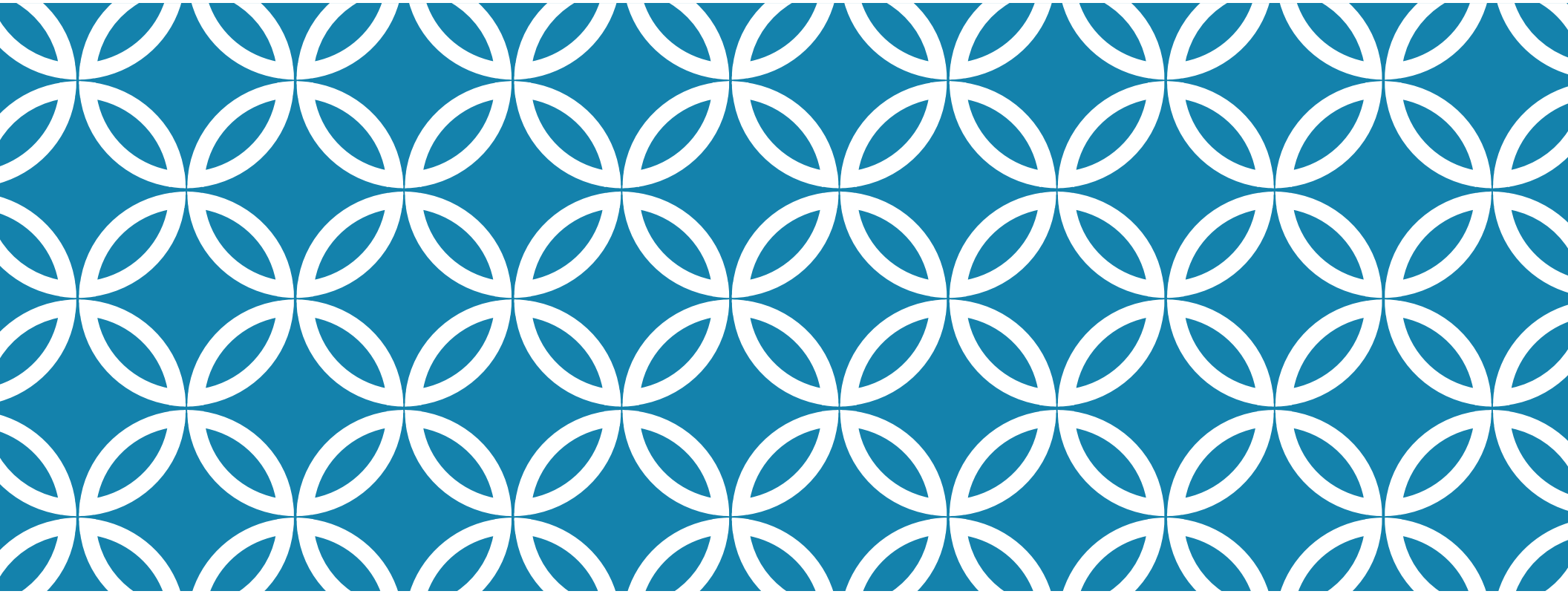


DEEP LEARNING

Transfer learning
&
Multi-task learning

PRACTICALITIES

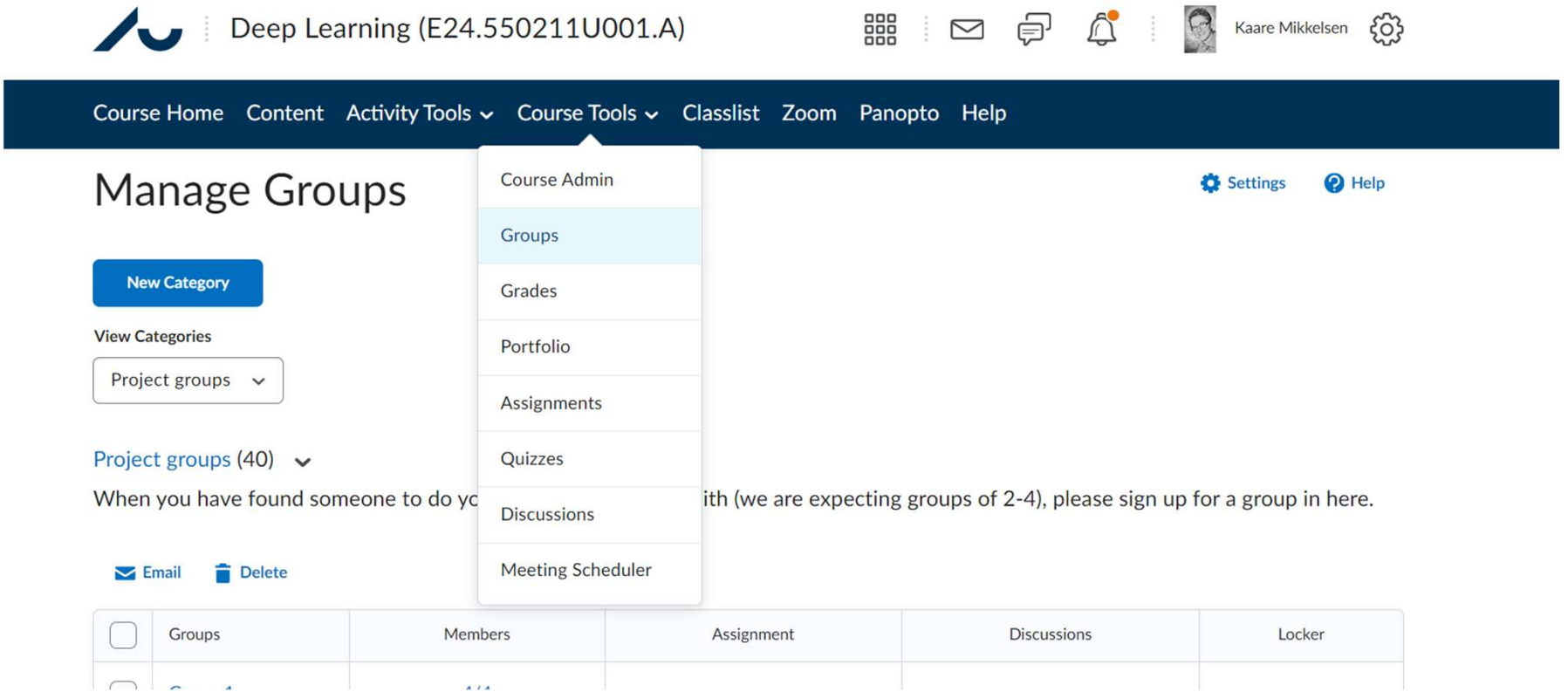
First assignment:

The TA's will get started on correcting them as soon as possible. Feedback will be given as soon as possible.

Project:

Time to form up in groups (2-4) if you already know your partner(s), or look for some if you don't!

PROJECT GROUPS:



GROUP SIZES

”What is your policy on small groups (1 or 2 people)?”

- We’re not huge fans. Mostly for the very practical reason that each new group is a new report we have to read during christmas holiday – and we actually thought we were being nice by letting you do the work in groups. But, we understand there may be very good reasons (such as working with sensitive data). If you have a good reason, let us know, and we’ll see about making an exception. It has happened before.

1 is worse than 2, 3 is very nice and 4 is downright awesome.

PROJECT TOPICS

You are allowed to pick any topic you want. However, bear in mind that you should pick a topic that is both complex enough to warrant a deep learning treatment, and simple enough that you can get somewhere.

Do note that it is still possible to pass the course even if you do not succeed in the task you design for yourself. Obviously, the writing process is simpler if you succeed, but you can still demonstrate understanding of the course without a solved problem.

Also: if your project is a 'kaggle project', we expect you to argue what is different about your approach.

LEARNING OBJECTIVES

These are the learning objectives of the course, according to which we are supposed to rate your exam (i.e. the final project). Note the ones in bold:

Learning Objectives:
After the course, the students are expected to be able to:

- **Identify and describe problems that can be solved with deep learning**
- Describe typical elements of a neural network architecture, including input size, number of parameters, activation functions and cost function
- **Explain and apply different training algorithms for neural networks**
- **Explain common building blocks, such as convolutional layers, pooling layers, fully-connected layers and recurrent (LSTM) layers, activation functions and optimization (cost) functions**
- Explain different network architectures such as feedforward networks, generative adversarial networks (GAN) and Variational autoencoders
- **Design a complex deep-learning model based on an analysis of the problem at hand and the project's goals**
- **Implement deep-learning models**
- **Describe and apply methods for evaluating the performance of deep learning models and relate results to purpose, method and input data**
- **Use selected programming libraries (frameworks) and development tools for deep learning**

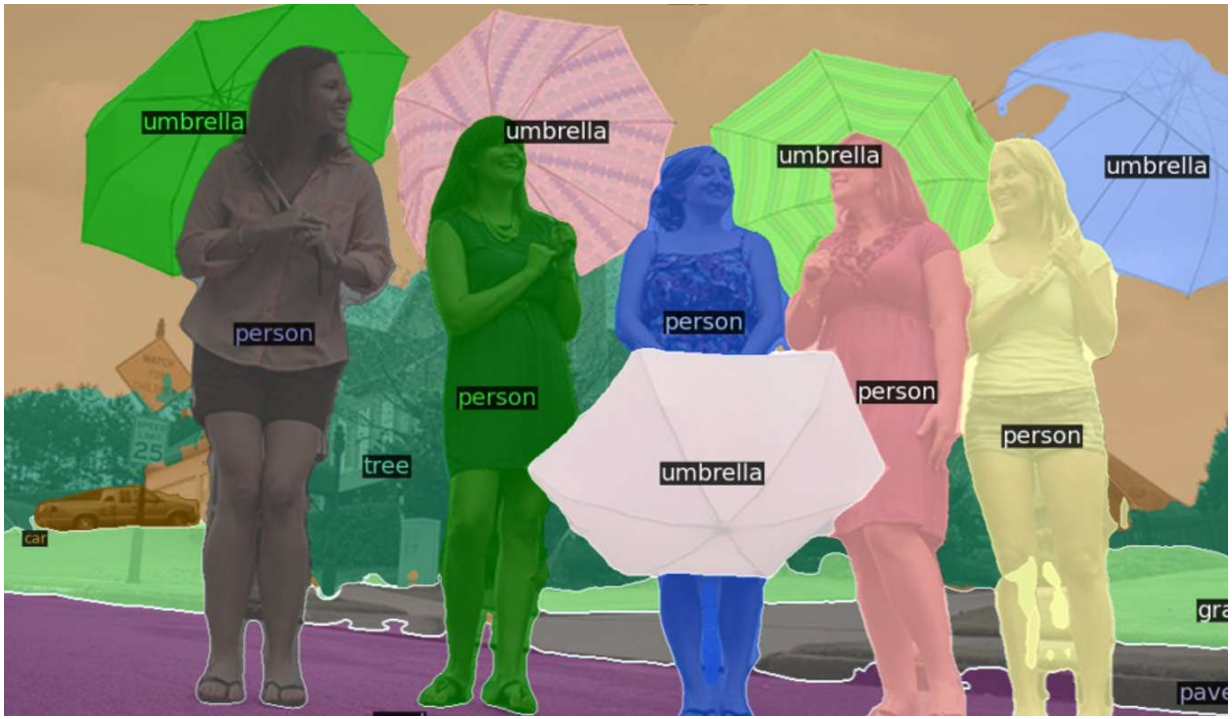
GPU NOTES

Group users have now been created on the gpu cluster.

The users are tied to the group numbers. A short guide is shown on brightspace for how to use the cluster.

Bear in mind that the cluster has space for at most 16 groups. Last year, some of the feedback received was that we should have stressed a *little bit* more how crowded it can get.

LUMI users have now been created for all those who signed up for the tutorial. If you haven't received an email, please check your spam.



LAST WEEK

Last week, you heard about:

- Modern Convolutional Neural Networks
- Residual blocks
- Semantic segmentation
- Object detection

QUIZ QUIZ QUIZ:
WHAT IS A RESIDUAL CONNECTION?

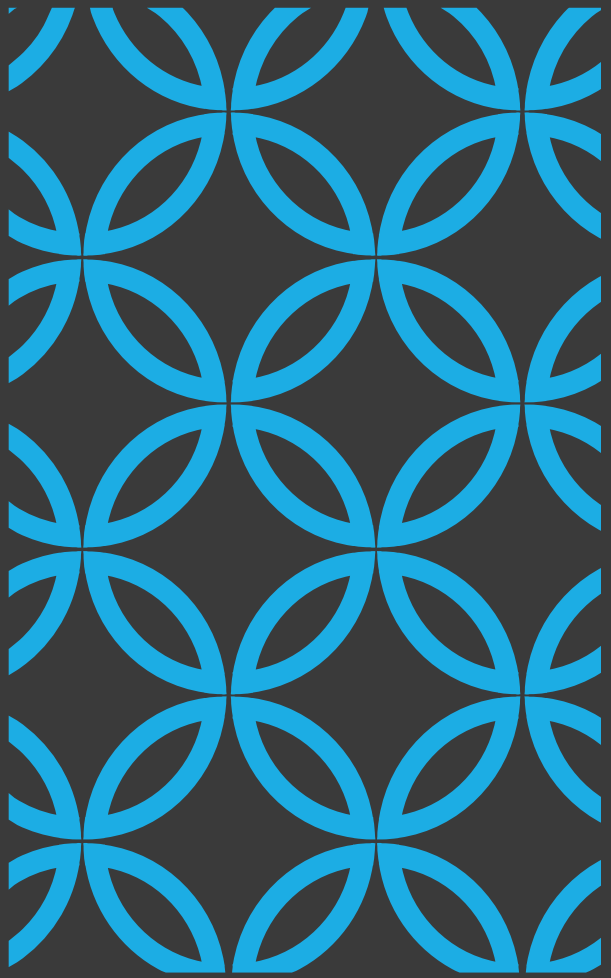
A residual connection is:

A: A ‘left over’ connection which is not being used (usually related to a deactivated neuron)

B: A network is said to ‘have a residual connection’ if it focuses on reducing the residuals in a regression problem.

C: A connection between non-neighbouring layers in a neural network.

D: A connection between layers transmitting changes instead of values.



TRANSFER LEARNING

Moving learning from one domain to another

CONCEPT

Problem: Not enough data or compute to train the model

Possible solution: Could we somehow use data from another problem to help us get started?

PLAN

- Terminology
- Simple / standard approach
- Advanced examples
- Multi-task learning
- Even more examples

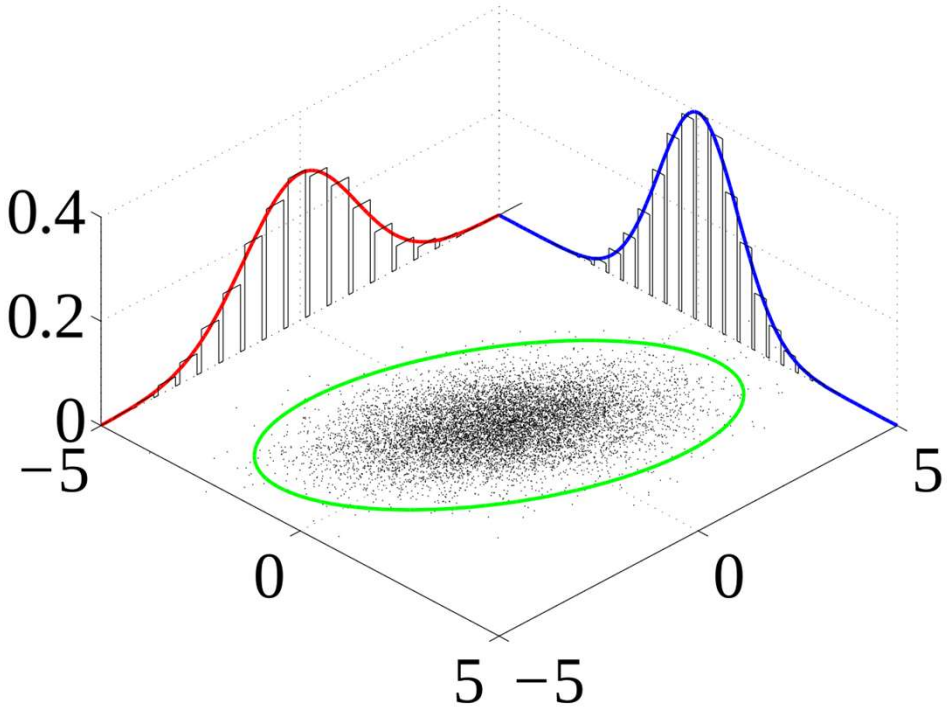
TERMINOLOGY : MARGINAL DISTRIBUTION

Given a 2-dimensional probability distribution, p(x,y), we can collapse the distribution to either axis:

$$p(x) = \sum_{y} p(x, y)$$

This is the marginal distribution over x.

In our case, x could be the possible inputs to our model, and y the target outputs.



TERMINOLOGY : DOMAINS

Domain, D: a combination of a feature space, χ, and a marginal distribution, $P(\chi)$

Task, T: possible labels (y), and the (unknown) function $f: \chi \rightarrow y$ (can also be thought of as $P(y|x)$)

Example: Detection of spam emails
D: $\chi =$ written emails, $\mathrm{P}(\chi) =$ possible sentences in Danish and English
T: $y =$ {spam, not-spam (ham)}, $f =$ predicting spamminess based on content

Example: Digit ID in MNIST
D: $\chi =$ 32x32 grayscale images, $\mathrm{P}(\chi) =$written digits
T: $y =$ {0,1,2,3,4,5,6,7,8,9}, $f =$ recognising digits

Source: 'A Survey on Transfer Learning', Pan et al 2009, DOI: 10.1109/TKDE.2009.191

TERMINOLOGY

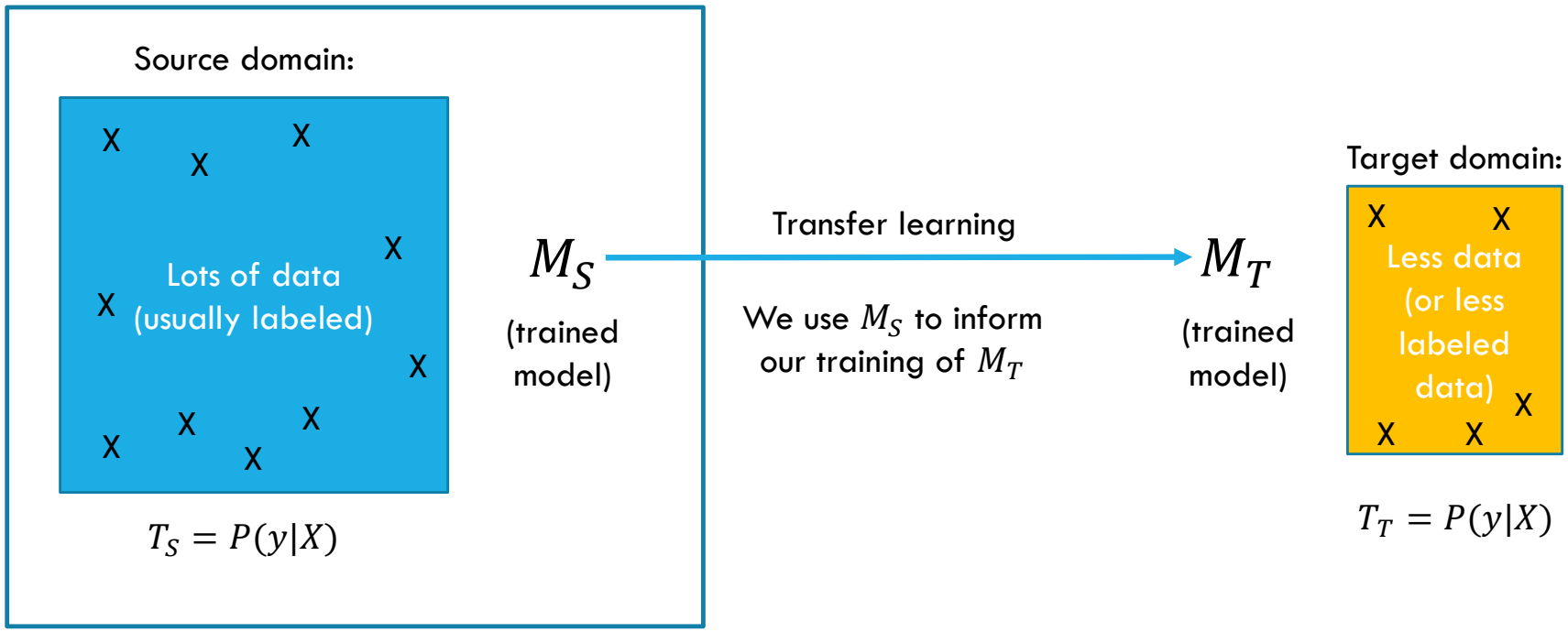
In transfer learning, we use two sets of domains and tasks, which we call *source* and *target*:

$$\mathrm{D_S}: \{\chi_S, P_S(\chi)\} \quad , \quad \mathrm{D_T}: \{\chi_T, P_T(\chi)\}$$
$$T_S: \{y_S, f_S\} \quad , \quad T_T: \{y_T, f_T\}$$

The objective is to use a good algorithm or dataset (the *source*) to improve performance on a related problem (the *target*).

Sometimes, we use the notation $f = P(y|x)$.

TRANSFER LEARNING:



COMBINATIONS OF SOURCES AND TARGETS

In regular supervised learning, $\mathrm{D}_S = D_T, T_S = T_T$. For something to be 'transfer learning', we require that either $\mathrm{D}_S \neq D_T$ or $T_S \neq T_T$ (or both).

Some specific classes of transfer have their own names:

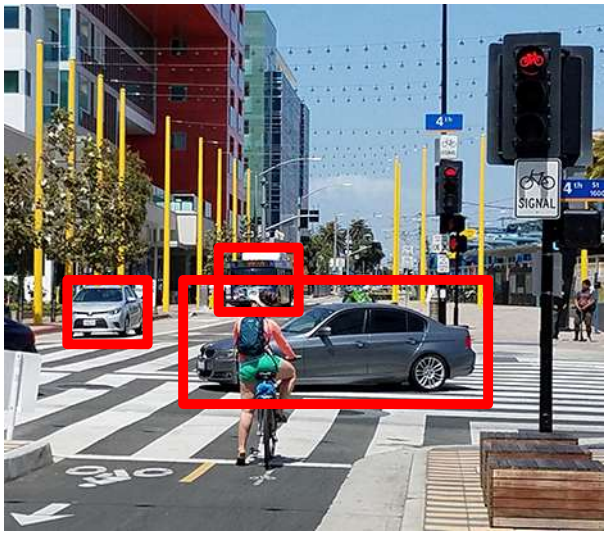
Inductive learning: whenever $T_S \neq T_T$

Example: teach a drone how to win in a race, by training in a simulator

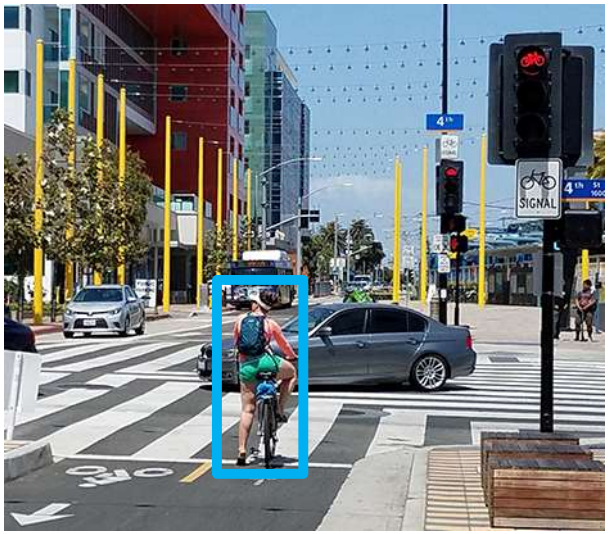
(In this case we may try to get T_S and T_T close to each other, but they won't be identical)

INDUCTIVE LEARNING: $D_S = D_T, T_S \neq T_T$:

Source:
Detect cars



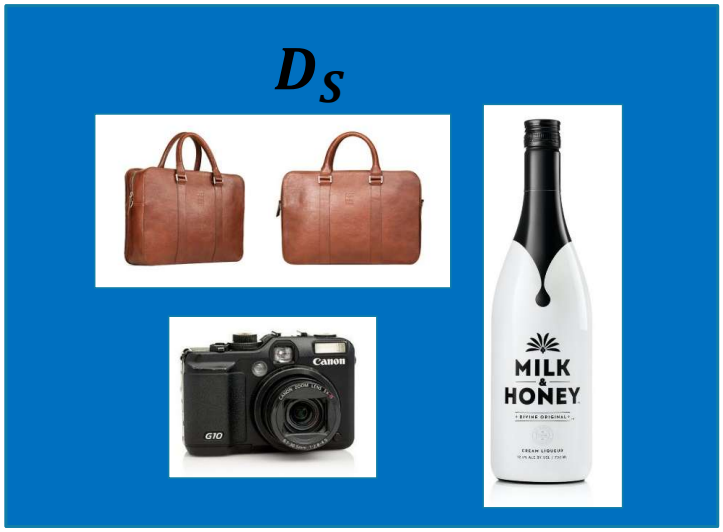
Source:
Detect bicycles

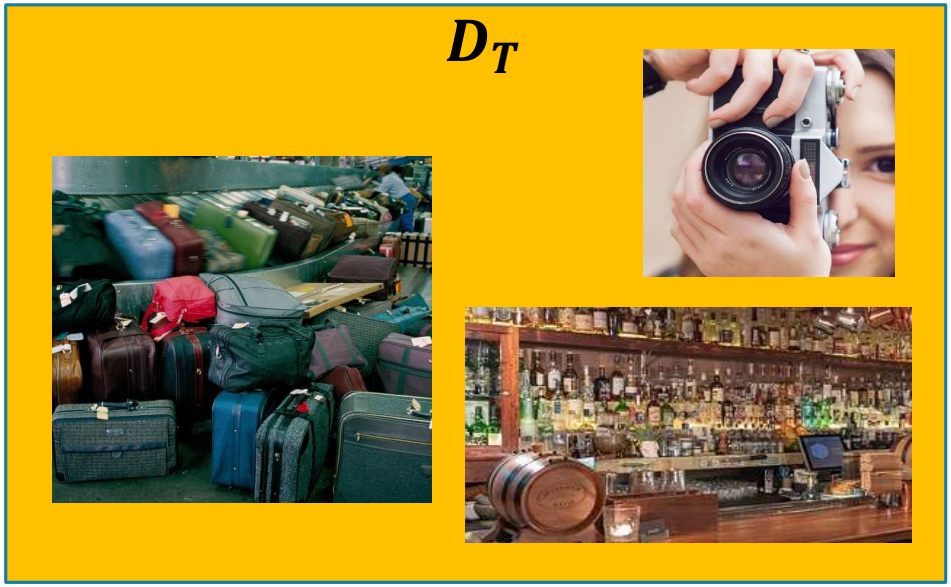


TRANSDUCTIVE LEARNING: $D_S \neq D_T, T_S = T_T$:

The assumption $P_S(y|x) = P_T(y|x)$ is called 'covariate shift' (because only x, the *covariate*, is allowed to change).

Example: Training a machine vision classifier on product images, but using it for detecting objects in natural settings:

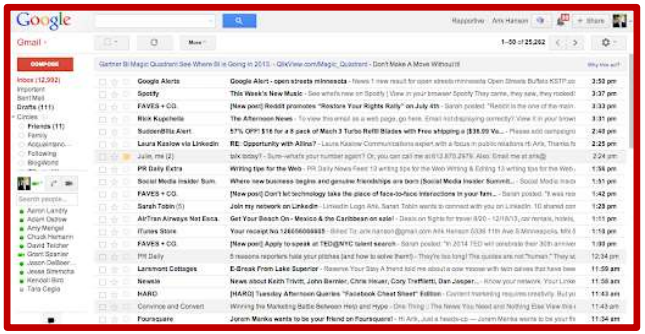


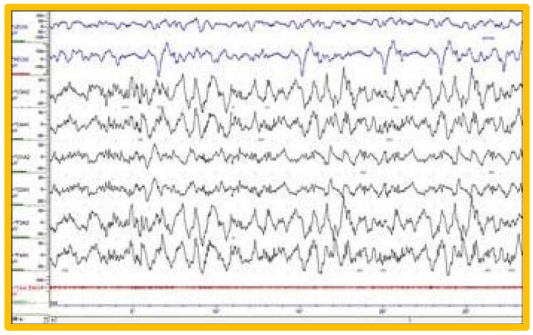


ALSO KNOWN AS ‘DOMAIN ADAPTATION’:

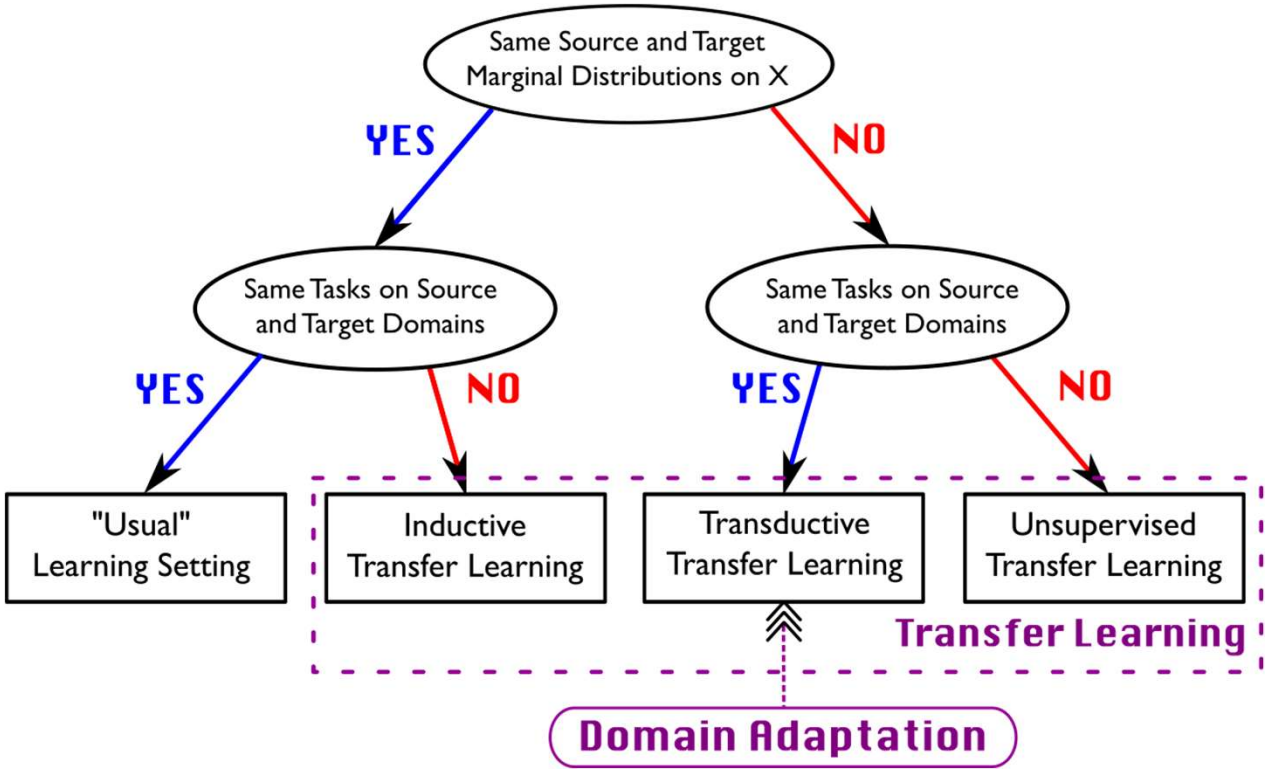
Examples:

- Get a general spam filter to work specifically on my inbox.
- Finetune a population-based sleep scorer to work especially well on an individual
- Learn to estimate sentiment from IMDB reviews, and transfer it to airbnb reviews.





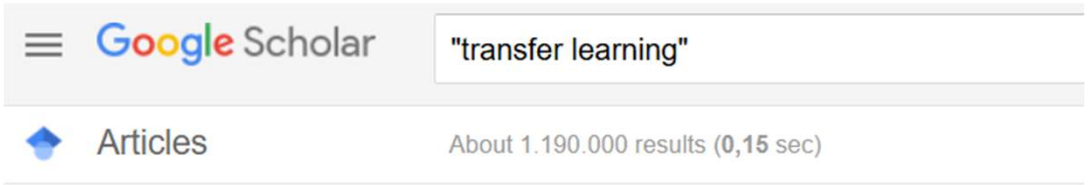
HANDY CHART FROM WIKIPEDIA:



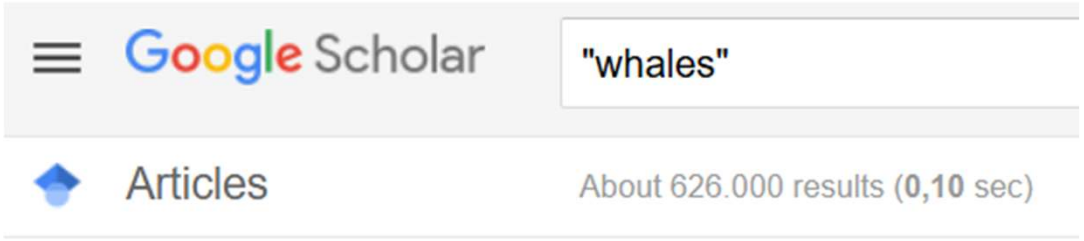
By Emilie Morvant - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=58812416

TRANSFER LEARNING: BIGGER THAN WHALES!

There is a *ridiculous* amount of papers on transfer learning:



To give you a sense of scale:



Also, bear in mind that we have not discussed how these terms relate to **supervised, unsupervised** or **semisupervised** learning.

Instead of trying to do everything, in the following we will cover a few select examples.

DISCUSS:

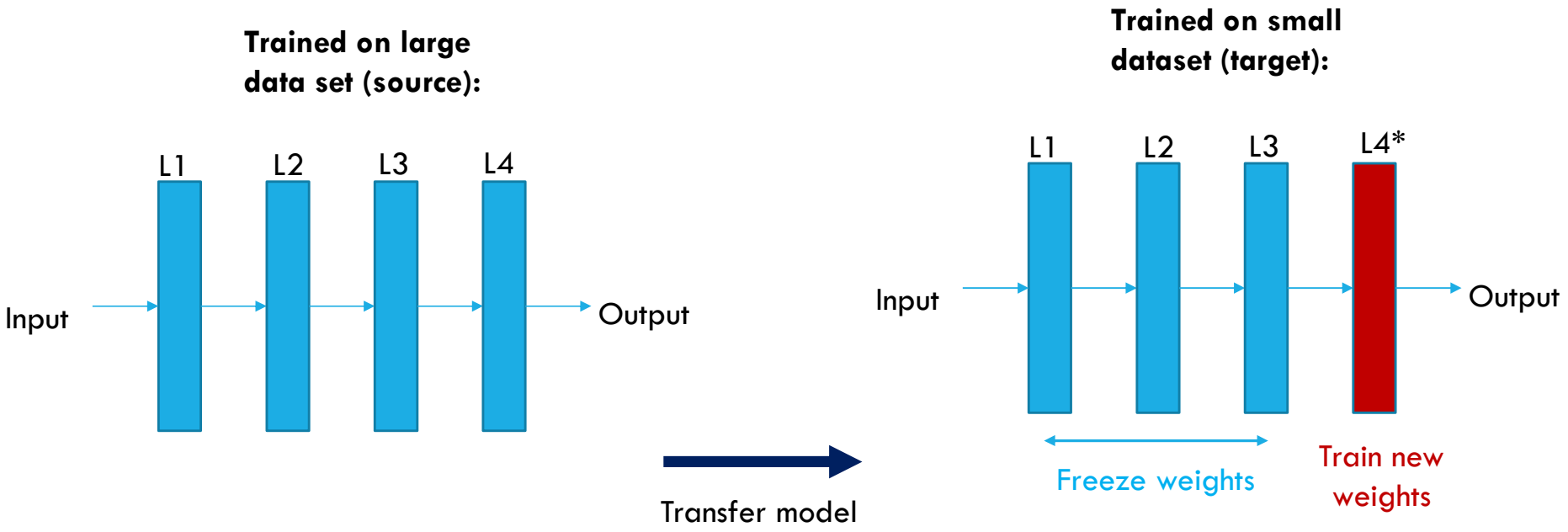
Assume a model for analysing behavior patterns (like, from a smartwatch) in a patient population. It's known that different age groups react slightly differently to the disease. We have a model that predicts disease level based on observed behaviours. We wish to transfer this model from one patient population to another.

What can we say about this transfer? Which are preserved?

- Feature space: χ
- Marginal distribution: $P(\chi)$
- Label space: $\{y\}$
- Target function: $P(y|x)$

MOST COMMON VERSION

The most common approach to transfer learning consists in taking a previously trained model, keeping most of it, and retraining a small part (typically the last layer).

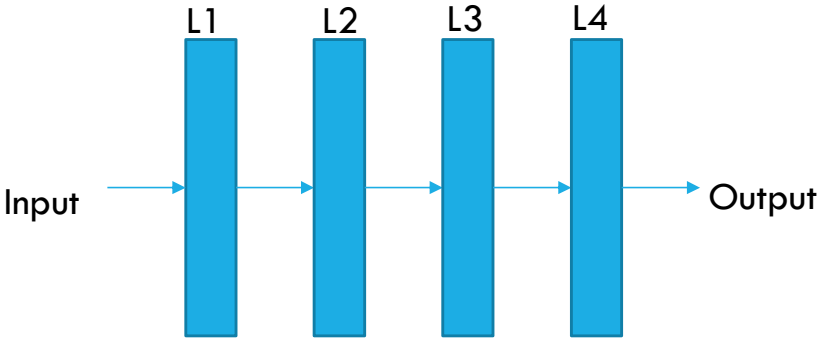


(In some treatments, this type of transfer learning is called 'feature extraction', while 'fine tuning' requires changing all weights. In others, both this apprach and whole-network training is called 'finetuning'.)

EXAMPLE USAGE

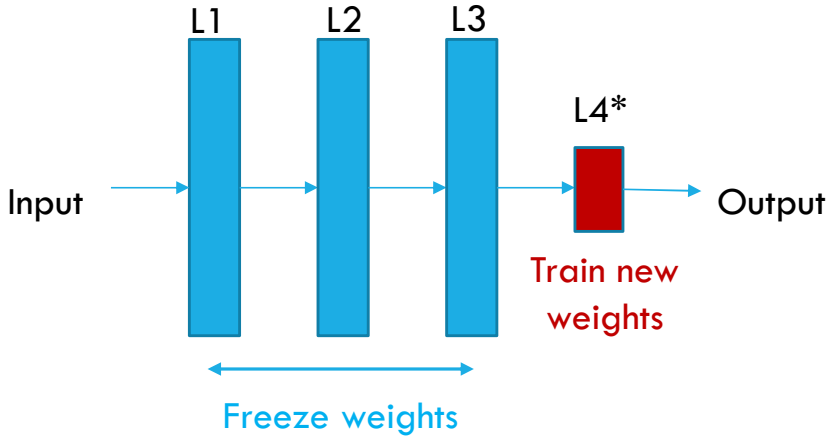
Assume we need a network that can sort photos taken inside cars and photos taken inside buildings.

However, we only have less than a hundred of each type. This is not enough to train a neural network, but it might be enough to finetune one.



So we take a large network trained on the ImageNet ILSVRC data set (1.2 mio. images), which distinguishes between 1000 classes. As we only have two classes, we remove the last layer, and put in a small one with only two outputs.

We freeze the rest of the network, and train the last layer normally.



EXAMPLE USAGE

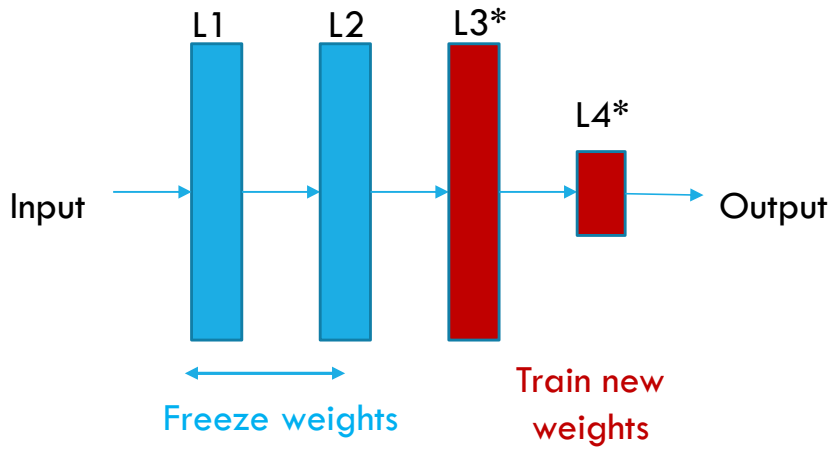
Note that the more data we have, the greater the part of the network we can retrain.

If we don't have to change the size of the layer, it usually makes sense to use the values in the source network as a starting point for the training.

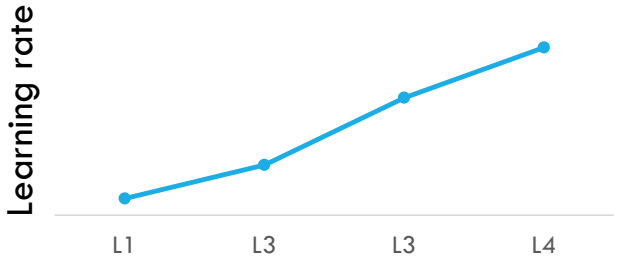
We can also imagine, instead of completely freezing the early layers, to have a gradually diminishing learning rate (so, separate learning rates for each layer). However, bear in mind that this increases the computation time of the backpropagation step.

Please note that whether or not freezing is necessary is hard to predict in advance – some times there may be too little data for training from scratch, but enough that freezing isn't needed.

If we had more target data:



Alternative:



HOW DOES IT WORK IN PYTORCH

As usual, our deep learning project has certain steps:

1. Import data
2. Define model
3. Train model

In this introduction, we shall skip the first step, and assume that we have already imported a suitable target dataset.

PACKAGE IMPORT

These slides are vastly more useful as notes if I also include the boring bits:

```
from __future__ import print_function
from __future__ import division
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
```

CHANGING & FREEZING LAYERS

The first thing we do is load a pretrained version of resnet18, and freeze all params.

After that, we overwrite the last, fully connected layer, to have the right number of classes.

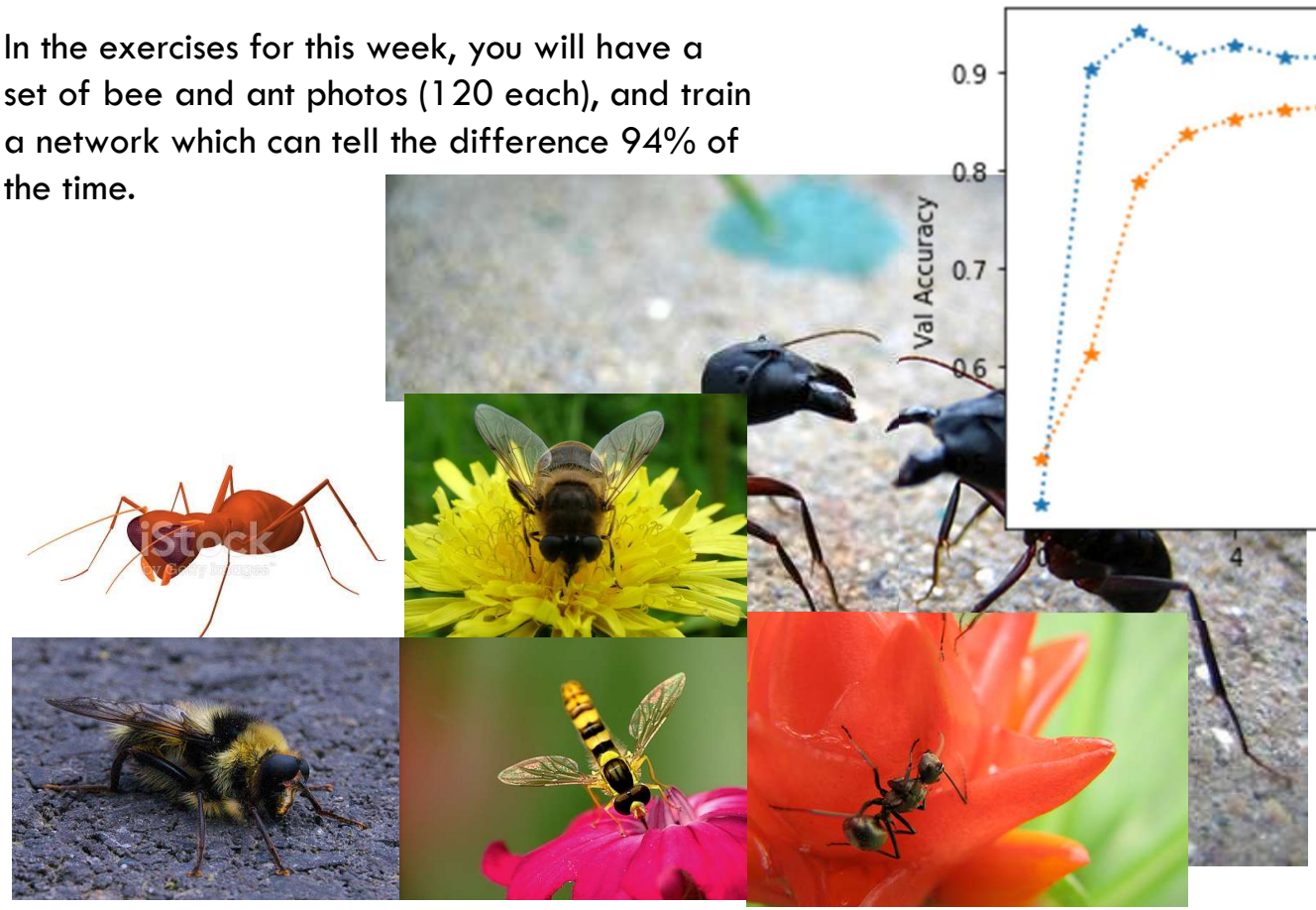
```
#load model and set up for fine tuning:
model_ft = models.resnet18(pretrained=True)
#freezing parameters:
for param in model_ft.parameters():
            param.requires_grad = False

num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, num_classes)
```

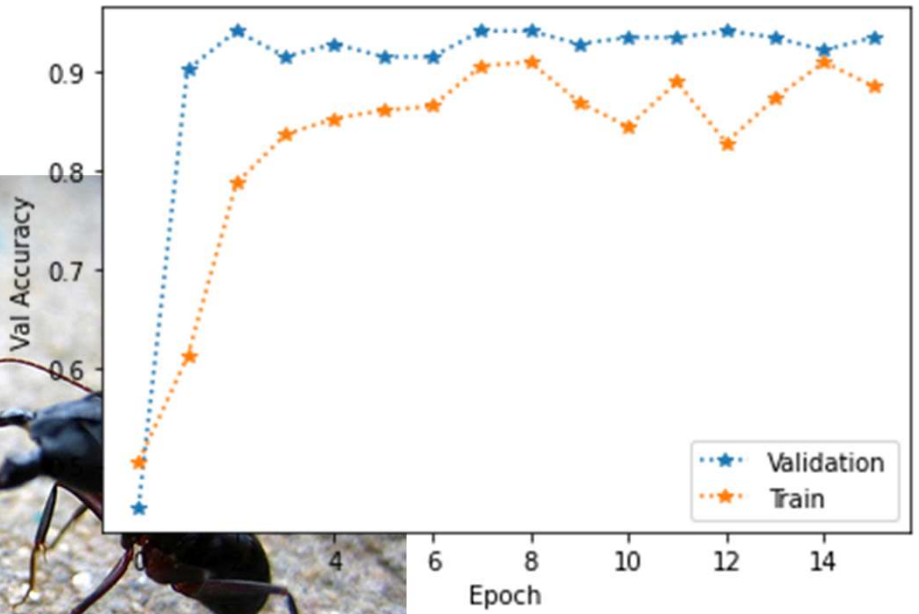
```
# Detect if we have a GPU available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Send the model to GPU
model_ft = model_ft.to(device)

# find parameters to update
params_to_update = []
for name,param in model_ft.named_parameters():
    if param.requires_grad == True:
        params_to_update.append(param)


# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(params_to_update, lr=0.001, momentum=0.9)
```

After this, we move the model to the gpu, and tell the optimizer to only focus on the last layer.

TRAINING & EVALUATING

In the exercises for this week, you will have a set of bee and ant photos (120 each), and train a network which can tell the difference 94% of the time.





MORE ADVANCED EXAMPLE: 'ANCHORING'

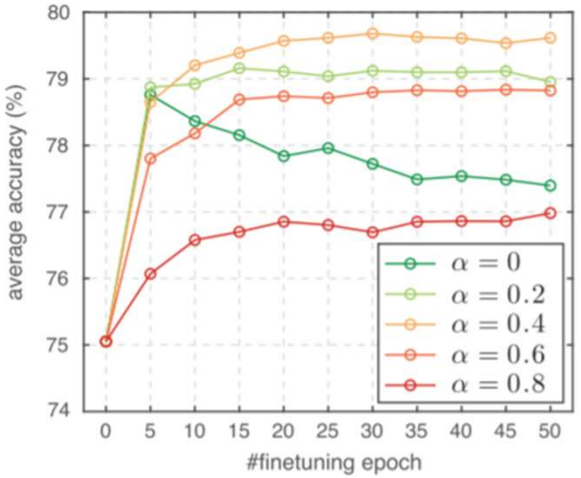
In Phan et al 2020*, it was investigated whether a sleep scoring algorithm could be personalized using only a single night's recording. Previous work had shown that at least 10 recordings were necessary for proper fine tuning.

The authors tried adding an additional regularization (or anchoring) term to the loss:

$$E'(\Theta^p) = -(1-\alpha)\frac{1}{L}\sum_{n=1}^{N}\sum_{l=1}^{L}\sum_{c\in\mathcal{C}}\mathbb{I}(y_{nl}=c)\log P_{\Theta^p}(\hat{y}_{nl}=c) + \frac{\lambda}{2}\|\Theta^p\|_2^2$$
$$-\alpha\frac{1}{L}\sum_{n=1}^{N}\sum_{l=1}^{L}\sum_{c\in\mathcal{C}}P_{\Theta}(\hat{y}_{nl}=c)\log P_{\Theta^p}(\hat{y}_{nl}=c).$$

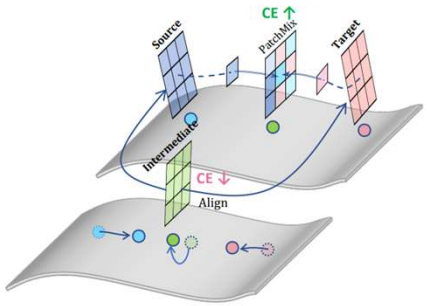
*"Personalized automatic sleep staging with single-night data: a pilot study with Kullback–Leibler divergence regularization"

This turned out to increase accuracy and avoid overfitting:

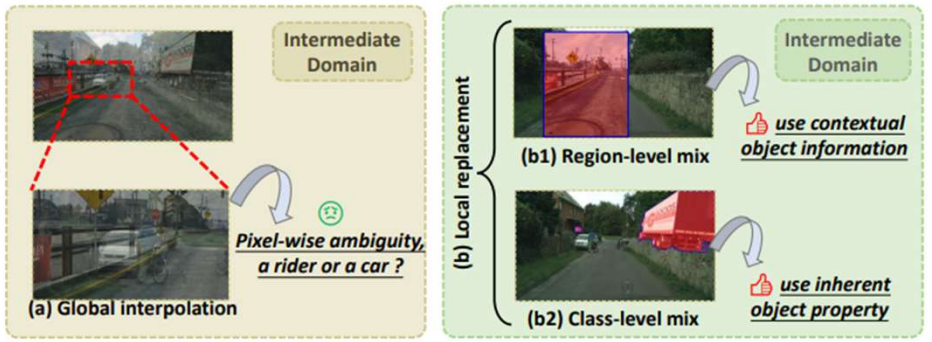


DOMAIN-BRIDGING APPROACHES

In general, many emerging methods can be loosely described as ‘domain bridging’, where different methods are used to design a ‘bridge’ between source and target domains, such that a model can be trained first on the easy problem of the ‘bridge’, before training on the target domain.



Patch-mix-transformer

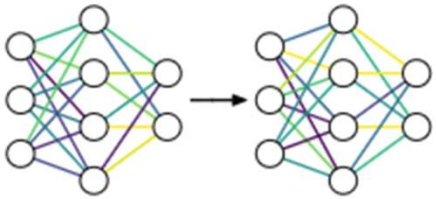


‘Deliberated Domain Bridging’

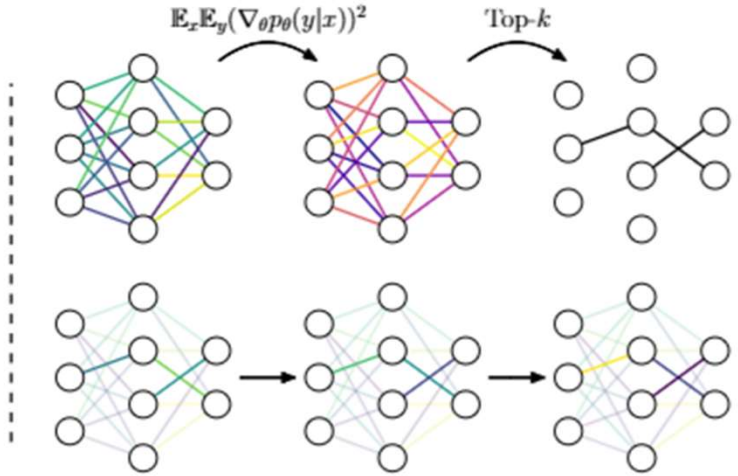
FISH:

(Fisher-Induced Sparse uncHanging)

Regular SGD:



FISH:

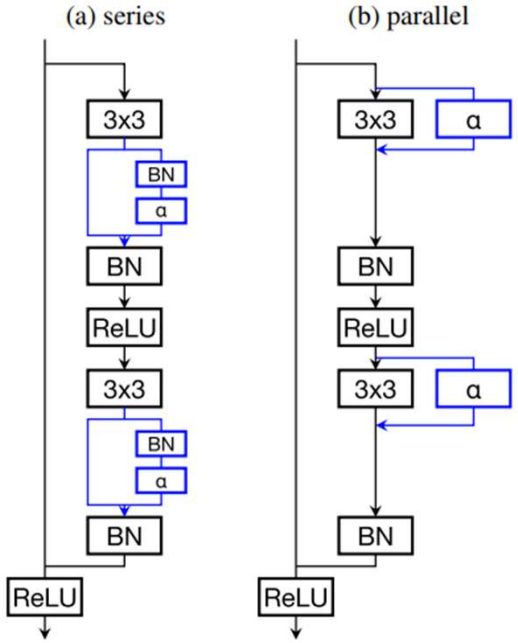


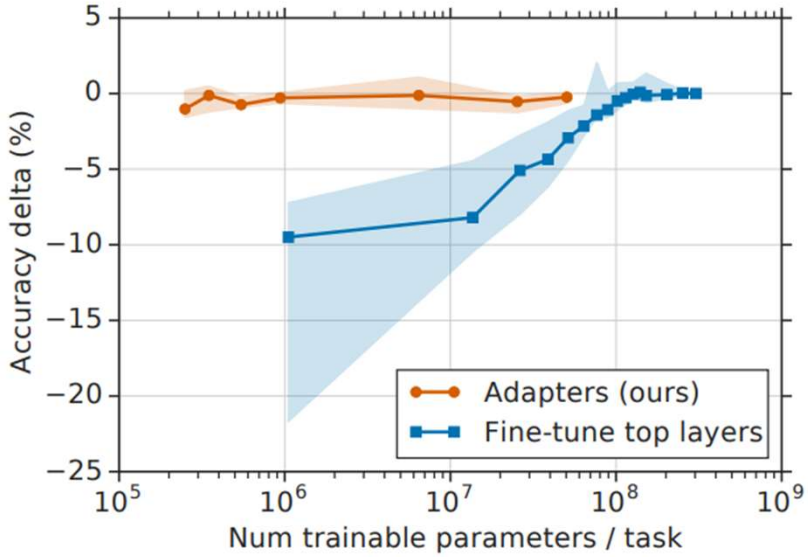
By inspecting the 'Fisher information matrix' of the model parameters, it is possible to select the parameters that seem most important for the model, and only update those.

[2111.09839] Training Neural Networks with Fixed Sparse Masks

PARAMETER EFFICIENT TRANSFER (PEFT) OR: ADAPTERS

In a special approach to fine tuning, 'adapters' are dedicated segments placed inside (in series or parallel) a pretrained network. During finetuning, only the adapter weights are updated. When it works, it seems to lead to models that can be fine tuned very efficiently.





Efficient Parametrization of Multi-Domain Deep Neural Networks (thecvf.com)

Parameter-Efficient Transfer Learning for NLP (mlr.press)

IMPORTANT TYPE: LOW-RANK ADAPTATION (LORA)

Originally invented for Large Language Models, LoRA specifically focuses on how to reduce the size of matrix updates (there are lot of matrix multiplications in Transformers).

$$M \rightarrow M + AB$$

Where the sizes of the matrices are:

$$M: N \times L \qquad A: N \times r \qquad B: r \times L$$

If M is frozen, and only A and B are updated, the set of free parameters is reduced to r(N+L), which is potentially a lot less than N*L.

(Another, related version is called QLoRA.)

[2106.09685] LoRA: Low-Rank Adaptation of Large Language Models

BREAK

MULTITASK LEARNING (MTL)

As we have discussed, many different tasks on similar data can benefit from extracting the same features. A different way to take advantage of this, is to have the same network perform multiple tasks simultaneously.

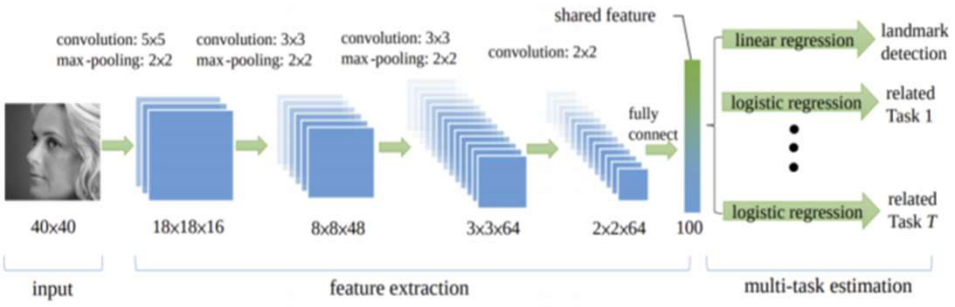
Examples:
- For sentences, predict the next sentence, as well as the mood of the writer.
- For photos of people, identify gender, estimate age, and ethnicity.

It has been found that combining multiple tasks can have a regularizing effect, which increases accuracy on the test data. However, this effect can be hard to achieve.

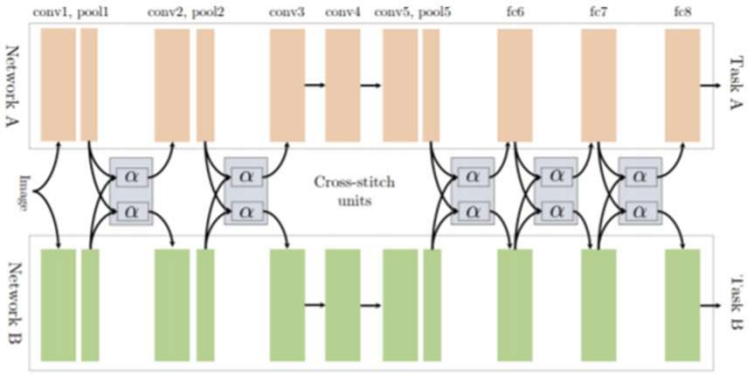
Sources: https://arxiv.org/abs/2009.09796 , https://ieeexplore.ieee.org/document/8848395, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6568068/

MTL ARCHITECTURES

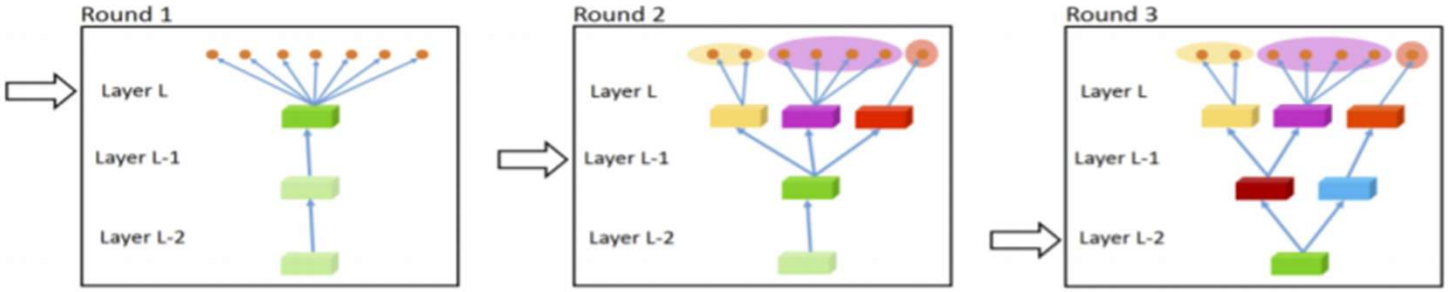
The most common structure of a multiclass network is the ‘shared trunk’, where the tasks only branch at the end.



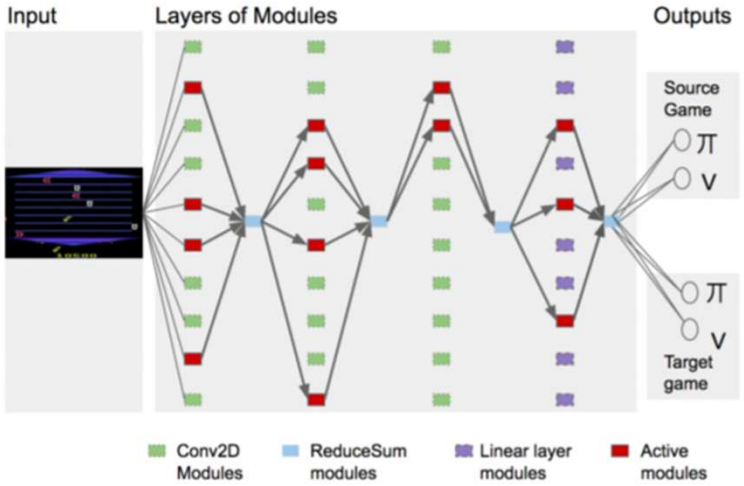
However, there are also examples of more elaborate architectures, such as two coupled networks:



Indeed, there are quite elaborate schemes, such as this, where the computational graph is updated during training, to disentangle the different tasks:



Or this, where different tasks use different 'modules' in a large shared network. The idea here is to combine modules using a genetic algorithm.



Source: https://arxiv.org/abs/2009.09796

CUSTOM LOSS FUNCTIONS IN PYTORCH

Depending on how you train it, it makes sense to either have separate loss functions for each 'head', or to combine multiple loss functions into one.

Luckily it is quite easy to build new loss functions from the exisiting functions, and have them evaluate on different parts of the input:

```
loss1=nn.SmoothL1Loss(reduction='sum',beta=0.05)
loss2=torch.nn.HingeEmbeddingLoss()
split=100
loss_fn = lambda pred,target : 0.5*loss1(pred[:split],target[:split].float())
        +0.5*loss2(pred[split+1:],target[split+1:])
```

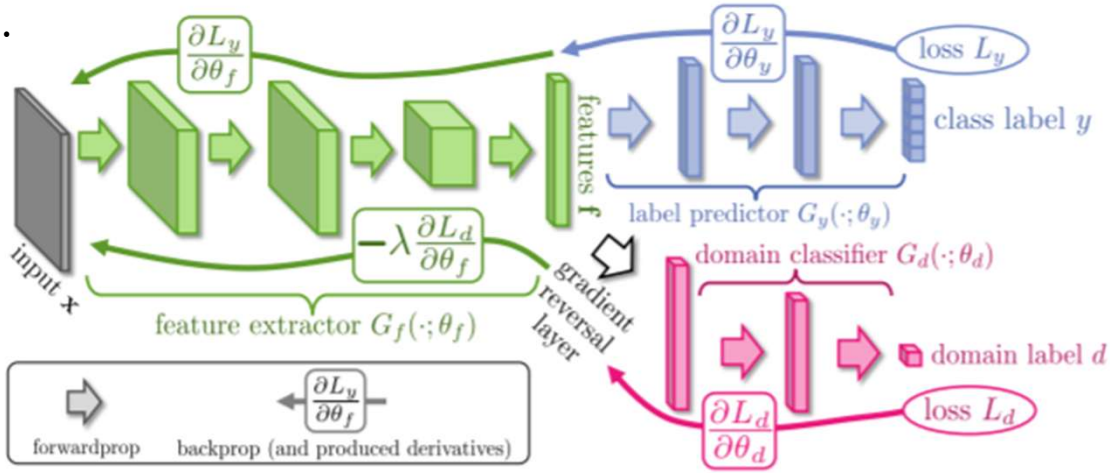
UNSUPERVISED DOMAIN ADAPTATION BY BACKPROPAGATION: $D_S \neq D_T, T_S = T_T$

Note: I have seen this network called a 'Domain Adversarial Neural Network' (DANN), but not by the authors.

In their paper, 'Unsupervised Domain Adaptation by Backpropagation', Ganin & Lempitsky suggest an intriguing method for transfer learning:

They envisage a network made up of three sections: feature extraction, label prediction and domain classification.
The network is alternately presented with datapoints from either source or target domain. For source data points, the label prediction and feature extraction is trained.

For target data points, the domain classifier is trained. However, when the gradient is backpropagated, its sign is changed between domain classifier and feature extractor.
This means that the feature extractor is adjusted to *not* distinguish between the two domains.



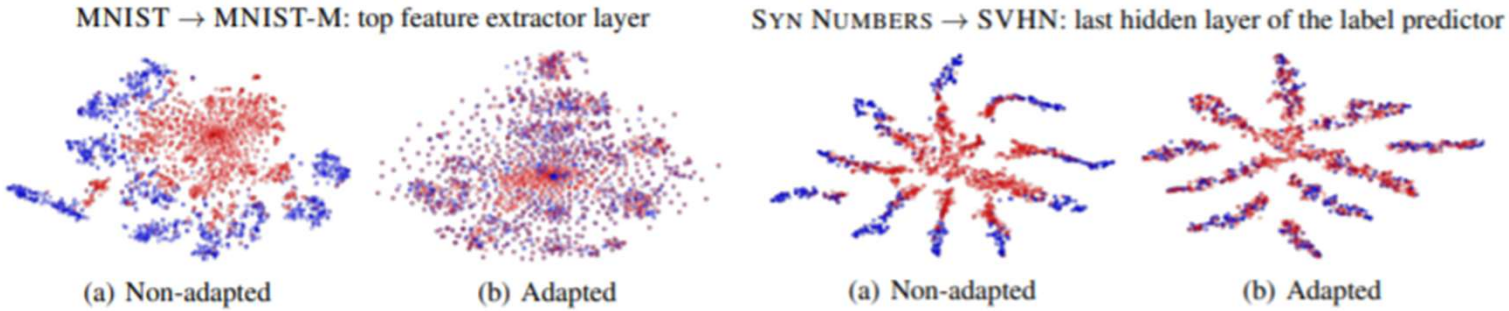
Source: https://arxiv.org/pdf/1409.7495.pdf

The authors tested their method on four different domain transfer problems:
MNIST -> MNIST-M
SYN numbers -> SVHN
SVHN -> MNIST
SYN signs -> GTSRB

Below is shown "t-SNE" plots before and after adaptation.

Figure 2. Examples of domain pairs used in the experiments. See Section 4.1 for details.

METHOD	SOURCE	MNIST	SYN NUMBERS	SVHN	SYN SIGNS
	TARGET	MNIST-M	SVHN	MNIST	GTSRB
SOURCE ONLY		.5749	.8665	.5919	.7400
SA (FERNANDO ET AL., 2013)		.6078 (7.9%)	.8672 (1.3%)	.6157 (5.9%)	.7635 (9.1%)
PROPOSED APPROACH		**.8149** (57.9%)	**.9048** (66.1%)	**.7107** (29.3%)	**.8866** (56.7%)
TRAIN ON TARGET		.9891	.9244	.9951	.9987



(t-SNE plots)

DISCUSSION

How does the previous method relate to the ‘Covariance shift’ assumption?
($P_S(y|x) = P_T(y|x)$)

DISCUSSION

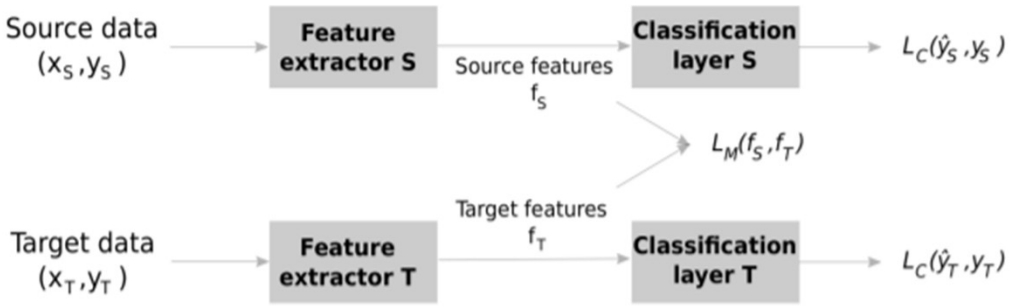
How does the previous method relate to the 'Covariance shift' assumption?
($P_S(y|x) = P_T(y|x)$)

Answer:

We assume covariance shift. If x could occur in both domains, then the network does not need to change x to 'hide' the domain. And, if the same x should lead to different decisions in the two domains, this is a bad strategy. Hence, we must assume the identity holds.

TRANSFER LEARNING THROUGH FEATURE MAPPING

In their paper "*Feature matching as improved transfer learning technique for wearable EEG*", de Vos et al suggest matching the latent feature spaces of two parallel pipelines, where one pipeline is dedicated to the source domain, and the other the target domain.



The method requires matching data points in the two domains, such that the two presentations can be simultanously run through the two pipelines.

‘FRUSTRATINGLY EASY DOMAIN ADAPTATION’

In his 2009 paper, Hal Daumé III suggests a very simple approach to domain adaptation - extend the feature space:

$$A = \{\forall x \in \chi_s | x \rightarrow (x, x, 0)\}$$

$$B = \{\forall x \in \chi_T | x \rightarrow (x, 0, x)\}$$

$$\chi_{comb} = A \cup B$$

The approach assumes a labelled target space, but excels in its simplicity.

(Note that this is not strictly speaking a deep learning method, but it has served as inspiration to the field of transfer learning in general).

Source: https://arxiv.org/abs/0907.1815

BOW: BAG OF WORDS

In the paper, the method was tested on text analysis problems, where a text is represented as a 'bag of words':

Document	the	cat	sat	in	hat	with
the cat sat	1	1	1	0	0	0
the cat sat in the hat	2	1	1	1	1	0
the cat with the hat	2	1	0	0	1	1

The logic then is that some words may have shared or different meanings between domains. 'Good' presumably means the same thing in most contexts, but 'monitor' means different things based on whether the domain is the Wall Street Journal or a technology website.

‘RETURN OF FRUSTRATINGLY EASY DOMAIN ADAPTATION’

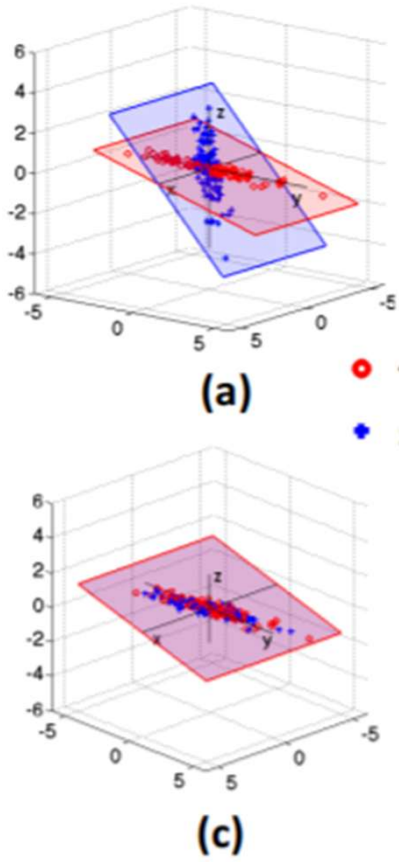
In 2015, a group of completely different authors decided to reuse Daumé’s paper title when presenting their own work: ‘CORelation Alignment’ (CORAL)

This is an unsupervised approach, where source data is transformed to have the same autocorrelation as the target data:

$$\forall x \in X_S : x \rightarrow A^T x A, A = (U_S \Sigma_S^{\frac{1}{2}} U_S^T) U_{T[1:r]} \Sigma_S^{\frac{1}{2}} \left(U_{T[1:r]}\right)^T$$

Where Σ is the autocorrelation matrix in the given domain, and U is the SVD-matrix of X.

The authors tested on both sentiment estimation and object recognition. They found that their approach worked better on ‘deep’ features, relevant for neural networks.



Source: https://arxiv.org/abs/1511.05547

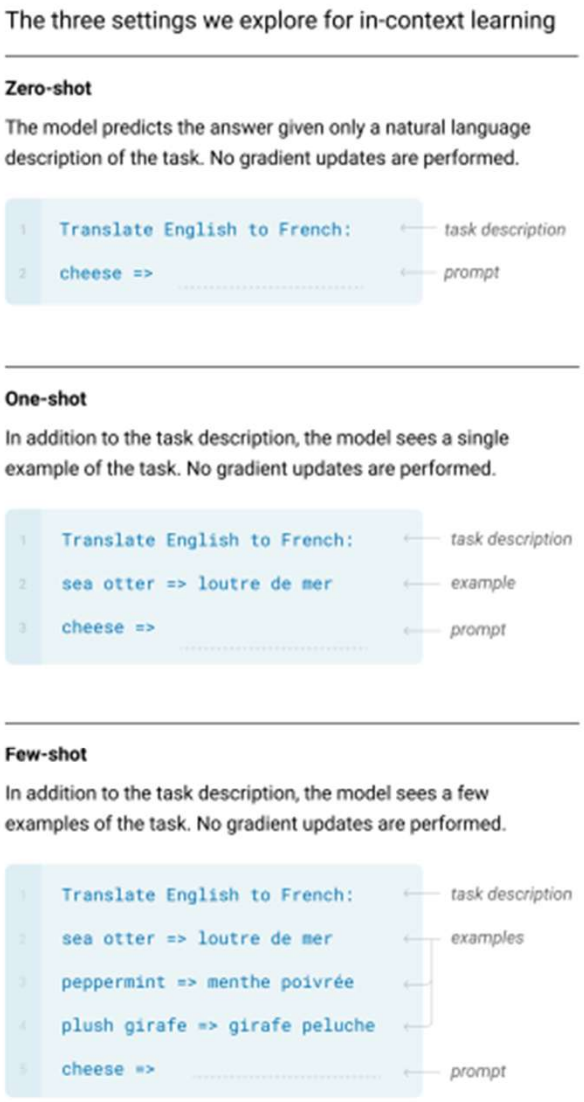
SOMETHING A LITTLE DIFFERENT

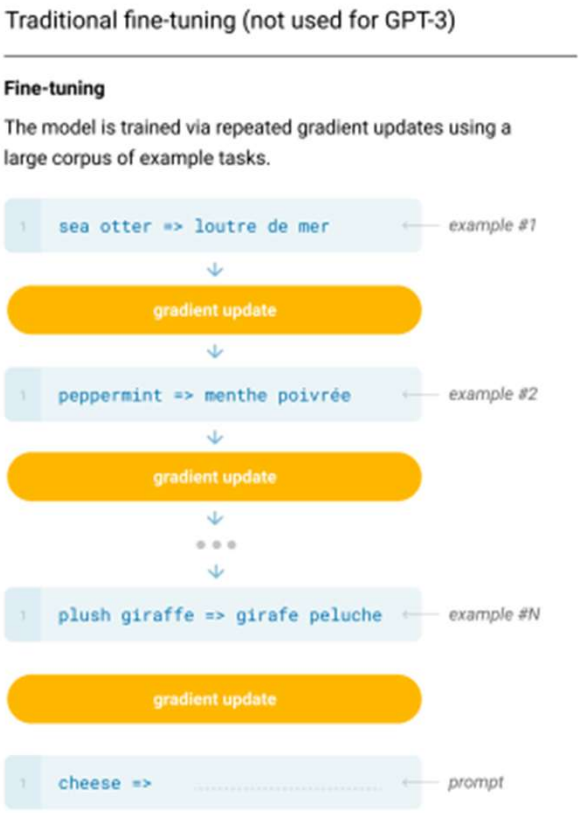
Microsoft co-pilot's take on 'something a little different' →

SOMETHING A LITTLE DIFFERENT: FEW SHOT LEARNING

In recent years, very large generative language models (covered in week 9) have been shown to be able to recognize a pattern from a few examples, and extrapolate from there. **This is without any changes in weights, so it is up for debate whether learning even takes place.**

This is also known as ‘meta learning’ or ‘in-context learning’.

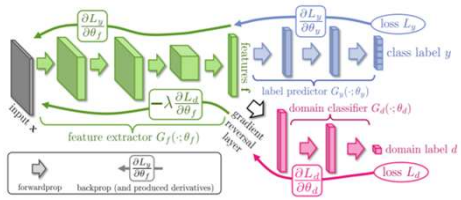




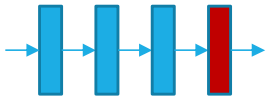
‘Language models are few-shot learners’: https://arxiv.org/abs/2005.14165

SUMMING UP:

We may group transfer learning into two categories:

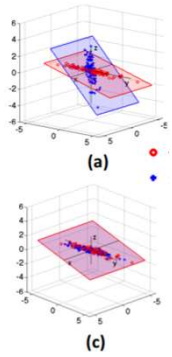


Fine Tuning



This is a standard method

Everything else



This can be as advanced as you like

READING MATERIAL

Required reading:
This is a good introduction to fine tuning:
https://d2l.ai/chapter_computer-vision/fine-tuning.html

Suggested Reading:
This week, the suggested reading also contains a couple of videos:

https://www.youtube.com/watch?v=MIsSuWsZtKE
Generally easy to follow, 1hr long.

https://www.youtube.com/watch?v=F2OJ0fAK46Q
Full, somewhat math heavy lecture by the author of the
'Frustratingly easy' paper. Very good, if you have a couple of hours.

EXERCISES

1: Learn to distinguish bees and ants.

https://colab.research.google.com/drive/1X9H0Oesy8G4be5Pb7A-aEwz1chef5QTk?usp=sharing

2: Perform multitask learning on a data set of my doing. You have to teach a model to count the points on a star, and determine the size of it at the same time.

https://colab.research.google.com/drive/1H60HZnl0ftTYP3UGdFgOtugI8wQrwqcR?usp=sharing

3: Perform multitask learning, but using quantile loss to estimate distributions of values in a regression problem:

https://colab.research.google.com/drive/11E50z0qdHAPrSbyWeN9nkxBWdUj9mrpv?usp=sharing

LUMI PRESENTATION