Image Processing I:
Image classification and transfer learning

Research Methods in Data Science – 7PAM2015

Dr Vanessa Graber (based on slides by Dr Gülay Gürkan)

Learning outcomes

After this lecture and the tutorial, you will:

- Know who your module lecturers are and what they will teach.
- Understand basics of image classification and get to know an unsupervised classification technique, i.e., self-organising maps.
- Know the advantages (and shortcomings) of transfer learning approaches and the role of Big Data in training neural networks.
- Be able to implement a transfer learning approach for image classification in Python.

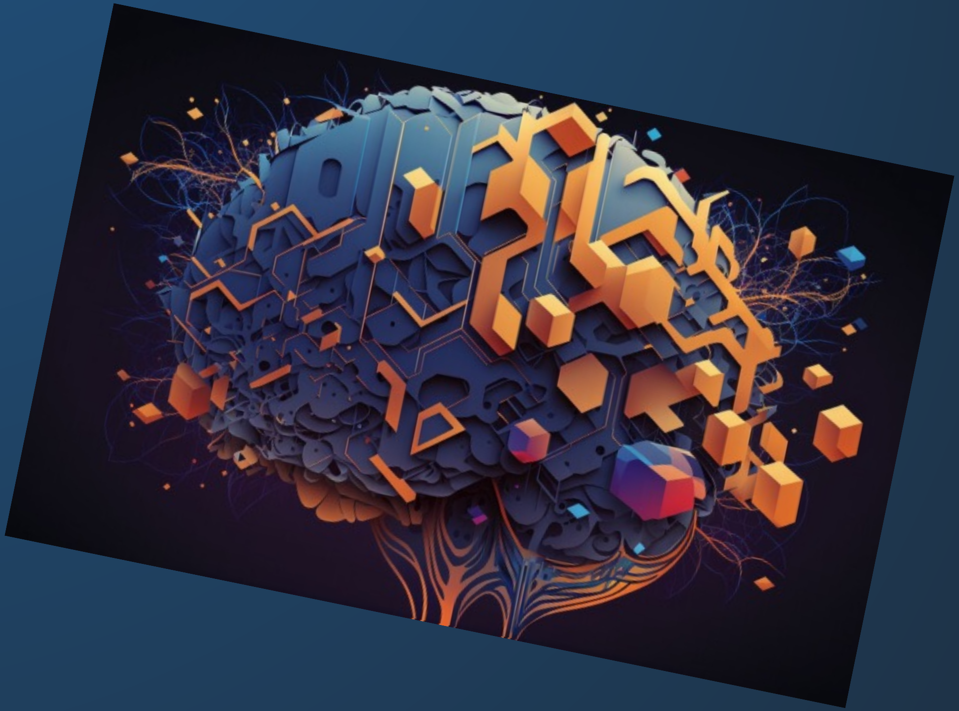
Course overview

Image classification: the basics

Transfer learning

Self-organising maps

Summary



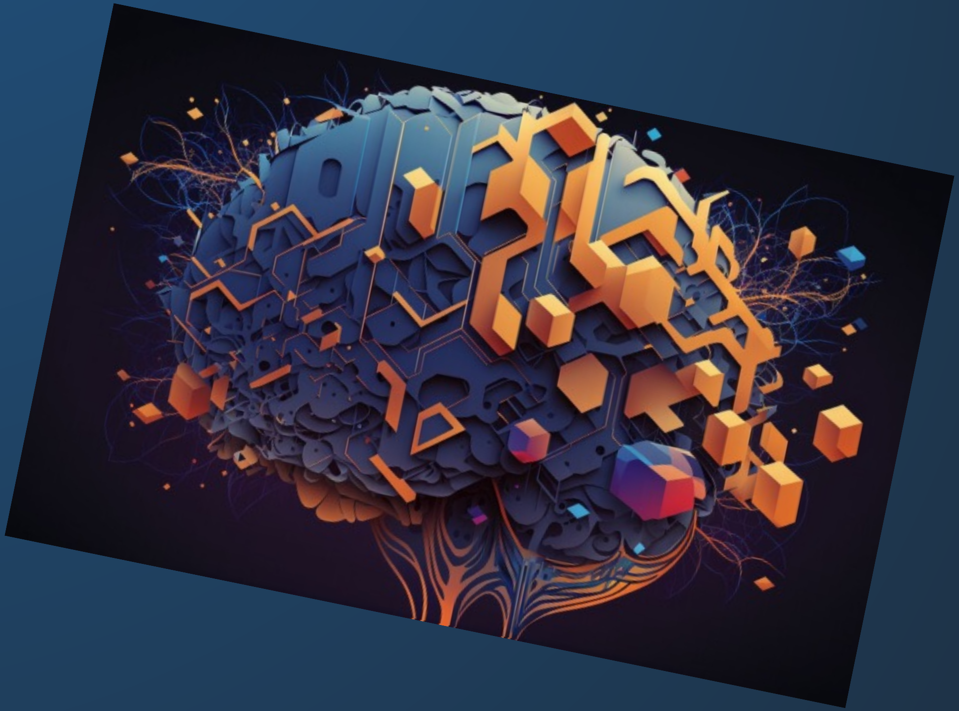
Course overview

Image classification: the basics

Transfer learning

Self-organising maps

Summary



Module lecturers

Who you will interact with this semester

- Lectures will be taught by three staff members from Centre for Astrophysics Research in Department of Physics, Astronomy and Maths:

Dr Vanessa Graber
(she/her)

Dr Rob Yates
(he/him)

Dr Peter Scicluna
(he/him)

Module lecturers

Who you will interact with this semester

- Lectures will be taught by three staff members from Centre for Astro-physics Research in Department of Physics, Astronomy and Maths:

Call me Vanessa
or Dr Graber

Dr Rob Yates
(he/him)

Module leader of
Research Methods

Module tutors

Who you will interact with this semester

- Tutorials will be taught by all three lectures and two additional tutors:

Dr Vandana Das
(she/her)

Dr Sunina Sharvy
(she/her)

Module overview

Part I – Dr Vanessa Graber

- 3 lectures on Image Processing with ML
  - Image classification
  - Transfer learning in computer vision
  - Object detection
  - Vision transformers
  - Methods for image segmentation
  - Big Data

1st assignment: image segmentation

Module overview
Part II – Dr Rob Yates

- 4 lectures on DS workflows:
  - Kaggle case studies
  - End-to-end ML cases
  - Data engineering
  - Advanced pre-processing & visualisation
  - Interpreting plots
  - Explainable AI
  - AutoML
  - Network analysis
  - Report writing

2nd assignment: Kaggle challenge

Module overview

Part III – Dr Peter Scicluna

- 4 lectures on Large Language Models
  - What are LLMs
  - How to use and adapt LLMs
  - Data, biases, scaling, models, and risks
  - Beyond current LLMs: models & applications
  - Research applications

3rd assignment: large language models

Course overview

Image classification: the basics

Transfer learning

Self-organising maps

Summary

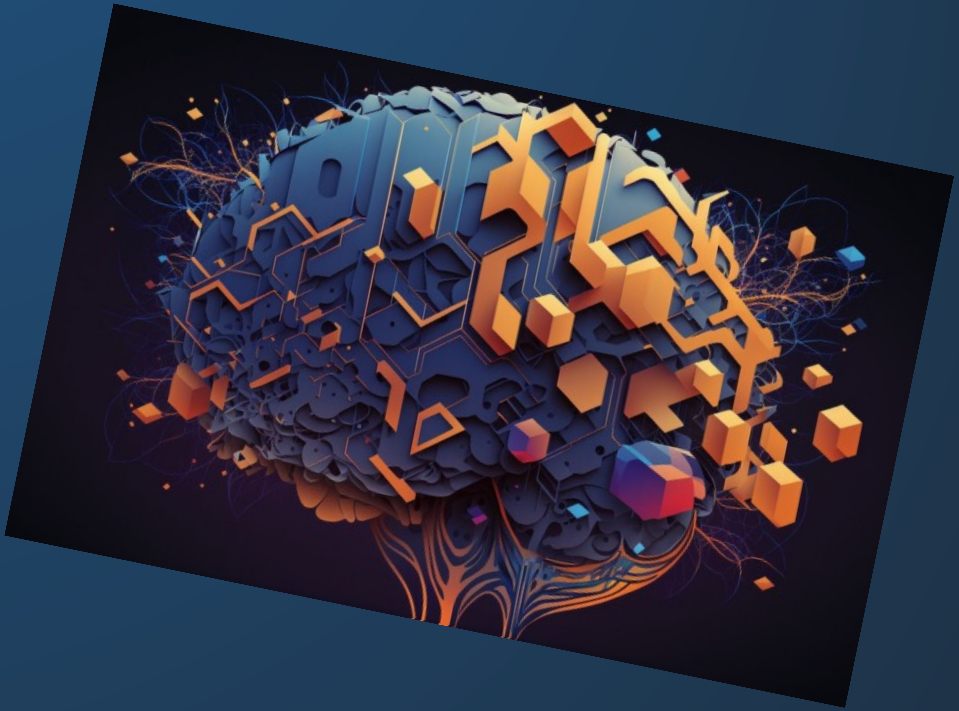


Image recognition

Overview

- Image recognition is a subfield of computer vision that teaches machines how to understand and “see” visual data.
- We typically distinguish the following tasks:
  - Image classification
  - Object detection
  - Image segmentation

In this lecture, we will focus on image classification.

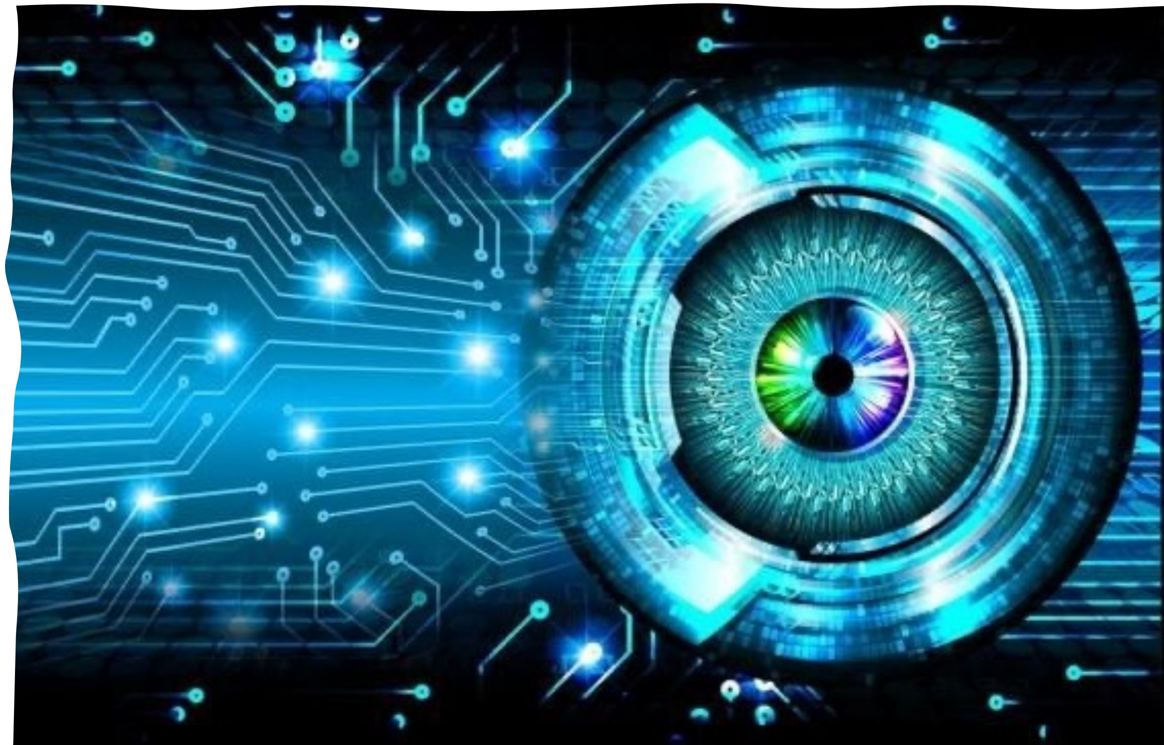
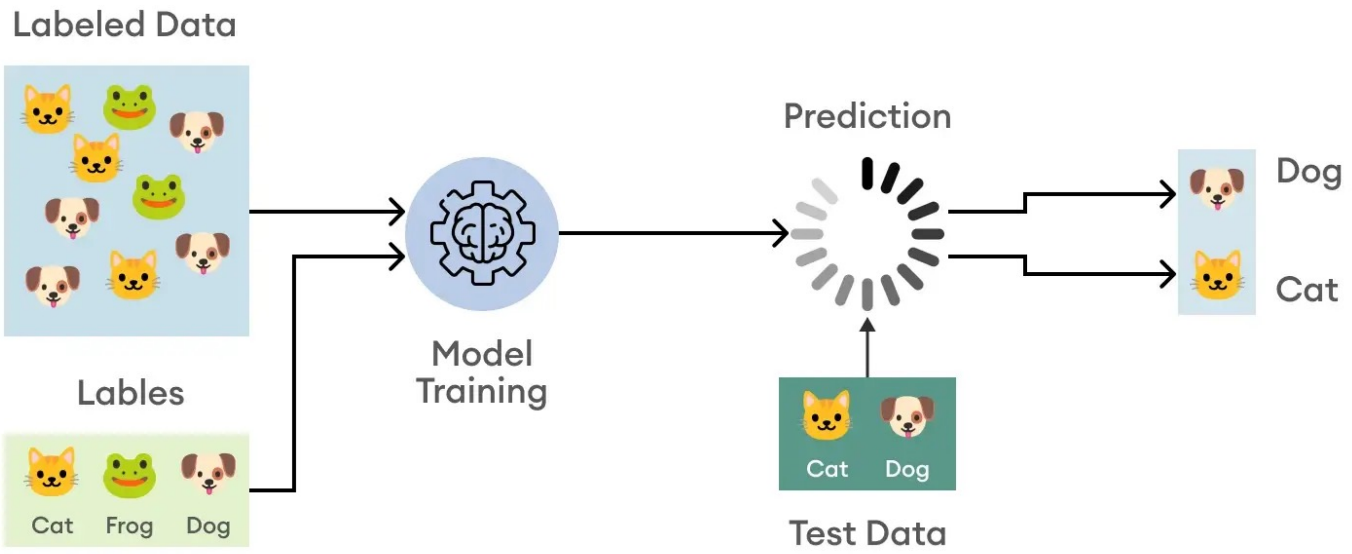


Image classification

The central idea

- Image classification is a fundamental task in the field of image processing using machine learning, where we categorise and assign labels to images or parts of images.

- For classification, we do not extract information from images (in contrast to object detection and image segmentation).



Credit: https://www.superannotate.com/blog/image-classification-basics

Image classification

Techniques

- When classifying images, we can distinguish those approaches that have access to labels and perform supervised ML and those approaches that do not require labels and perform unsupervised learning.

SUPERVISED METHODS

- Linear / logistic regression
- Decision trees
- Support vector machines
- Random forests
- Naïve Bayes methods
- K-nearest neighbour methods
- Neural networks

UNSUPERVISED METHODS

- Clustering algorithms
- Neural networks

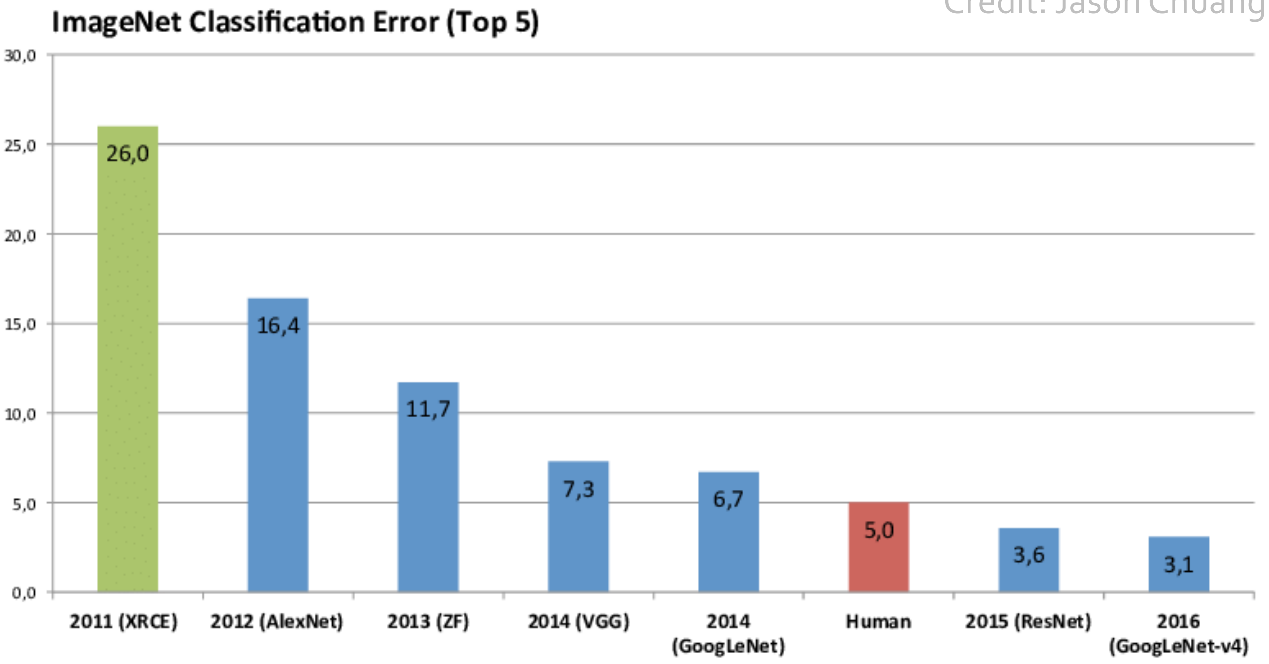
Neural networks have been outperforming traditional methods since 2012.

Image classification

The power of convolutional neural networks

- The availability of large (labelled) databases has led to a revolution in image recognition. In particular, ImageNet with >14 million hand-annotated images separated into >20,000 categories was crucial to the development of (convolutional) neural networks.

Credit: Jason Chuang



CNN components I

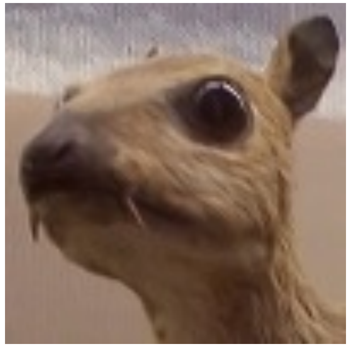
A reminder

- The primary component of CNNs are **convolutional layers**:

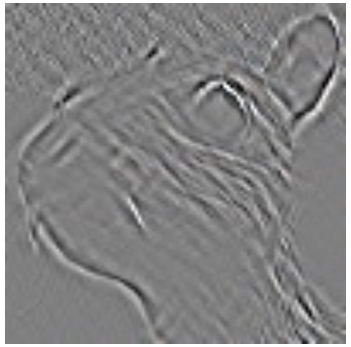
Credit: Michael Plotke, Crisluengo

Aim: extracting local patterns and features from input data

How: filters (kernels) slide over the input data and at each position, they perform element-wise multiplication and sum the results to produce feature maps



$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



Convolutional layers capture local patterns and preserve the spatial relationships between pixels.

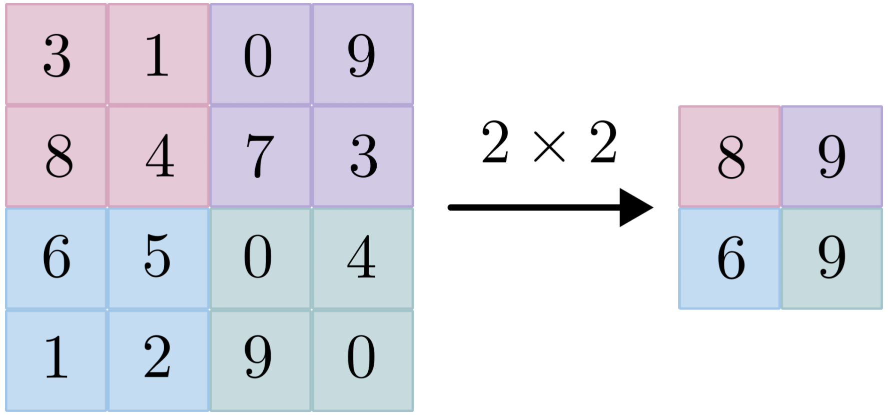
CNN components II

A reminder

- In addition, CNNs contain so-called **pooling layers**.

Aim: reduce spatial dimensions, speed up the computing time and enhance translational invariance

How: down-sample the spatial dimensions of input feature maps via, e.g., max pooling (selecting the maximum in a region) or average pooling (calculating the average)



3	1	0	9
8	4	7	3
6	5	0	4
1	2	9	0

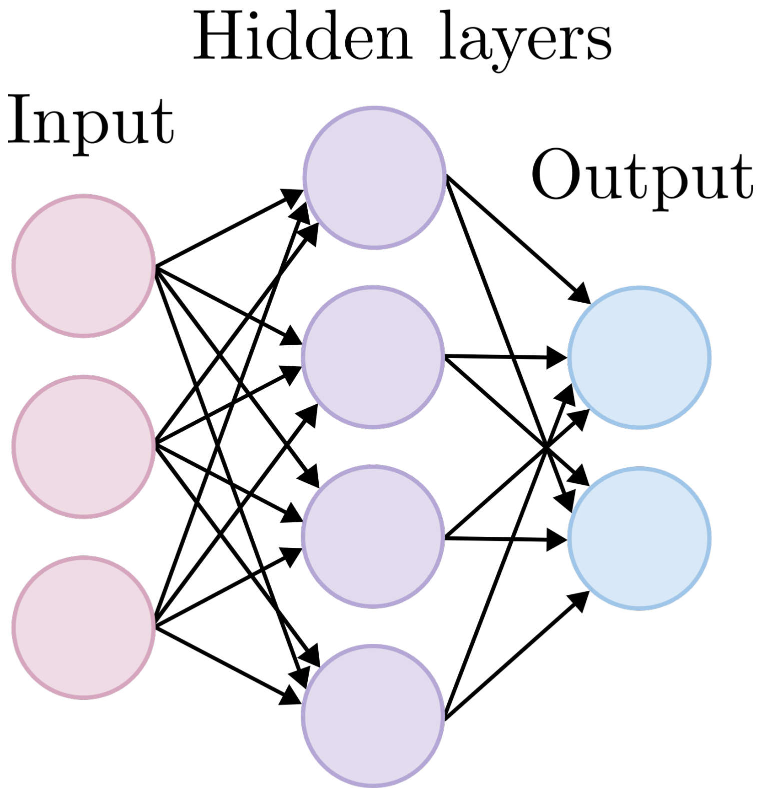
8	9
6	9

Pooling layers reduce the amount of information in the feature maps while retaining important features.

CNN components III

A reminder

- CNNs also contain fully connected layers.



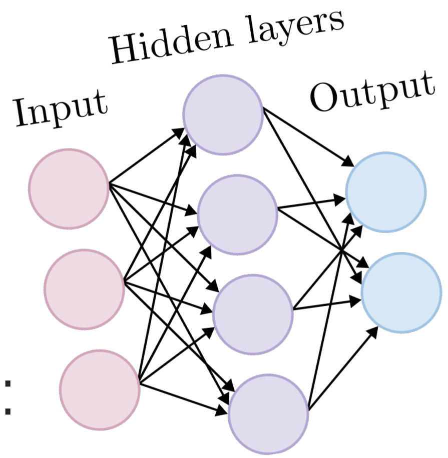
Aim: connect every neuron from previous layer to next layer to learn global patterns and relationships

How: each connection has a weight and each neuron a bias; the output is computed as a weighted sum of the inputs plus bias followed by an activation function

Fully connected layers are used in the final CNN stages to gather global information and make predictions.

CNN optimisation

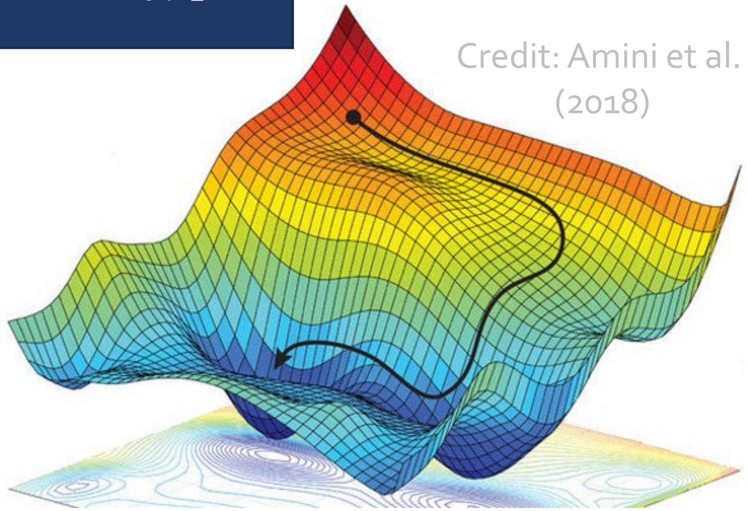
A reminder



- **Forward passes** through a NN are essentially matrix calculations of weighted sums and non-linear activation functions:

$$\begin{aligned}&\text{output}^{(\text{final layer})}\\&\quad= \text{activation function}\left[\text{bias}^{(\text{final layer})} + \text{weights}^{(\text{final layer})}\right.\\&\quad\left.* \left(\text{activation function}\left\{\text{bias}^{(\text{previous layer})} + \text{weights}^{(\text{previous layer})}(\dots)\right\}\right)\right]\end{aligned}$$

- In supervised approaches, we define a **cost function** to determine how good the network prediction is with respect to our ground truths. Network weights and biases are then updated using the back propagation algorithm (which is again a lot of linear algebra).



Credit: Amini et al. (2018)

Supervised vs unsupervised learning

The problem

- Image classification with CNNs has seen a lot of attention since the success of AlexNet in 2012 and is now a **well-studied problem**.

- However, the key to successfully training CNNs is the **availability of large amounts of data** to optimise the many free parameters of our neural network. Moreover, we require labels for our data that are not necessarily easy to come by.

What do we do if our datasets are limited, or we do not have access to any (or large amounts of) labelled data?

Question time

Mentimeter quiz

Let’s take a few minutes to recap what we have discussed so far:

Go to menti.com and enter the code 6461 2115.

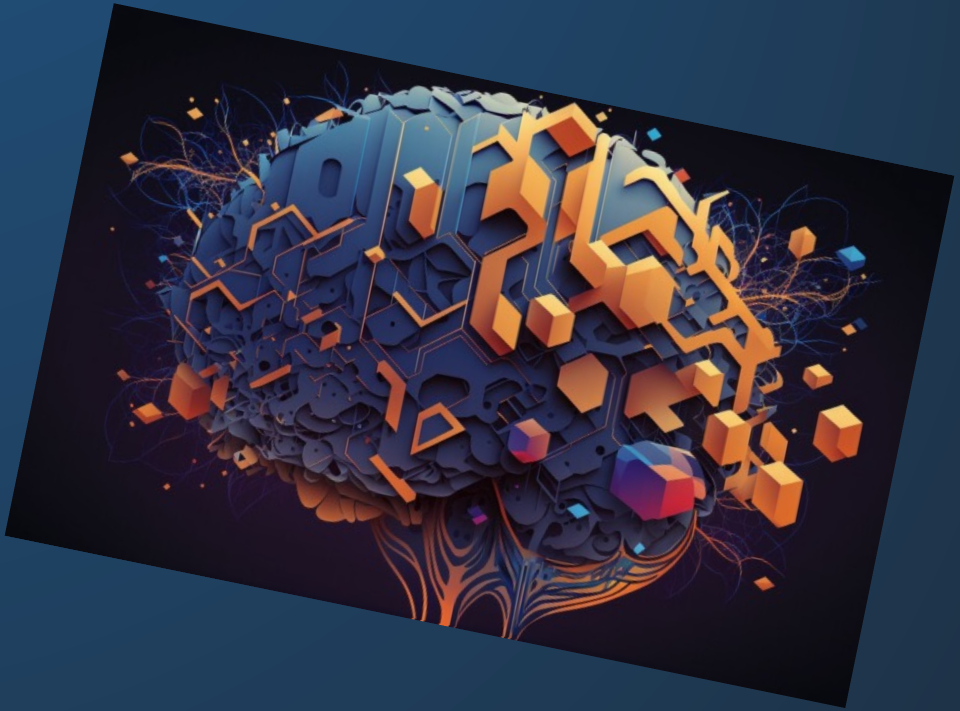
Course overview

Image classification: the basics

Transfer learning

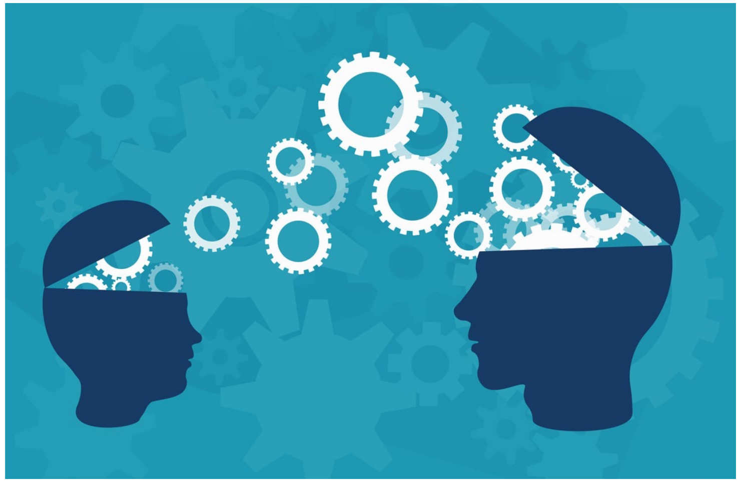
Self-organising maps

Summary



Transfer learning
Overview

- When performing transfer learning, we first train a **base network** on a base dataset to perform a base task (e.g., classify ImageNet images).
- We then repurpose the learned features and transfer them to a second target network that we train on a different target dataset and task.
- This process will work well if the features are sufficiently general, i.e., they must be suitable to analyse both the base and the target task.



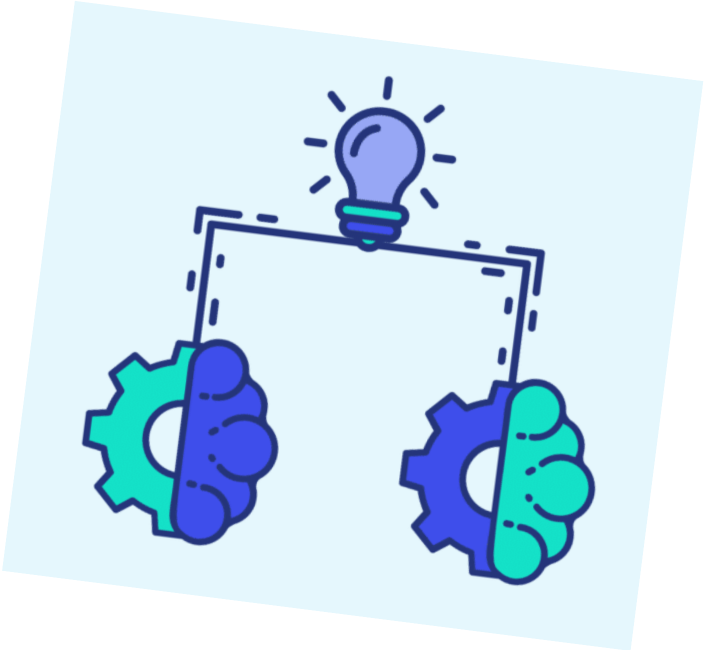
Transfer learning allows us to perform computationally efficient and effective ML on smaller datasets by taking advantage of prior knowledge (weights) from previously trained networks.

Crucial also for object detection and segmentation (see next lectures).

Transfer learning

Details

- Transfer learning is popular in computer vision and natural language processing. When processing images, we use the following steps:
  - Load layers from a previously trained base model such as VGG16, ResNet, etc.
  - Freeze weights and biases in these layers so that pre-trained information is not lost in future training rounds.
  - Add several new trainable layers on top of the frozen layers to turn old features into predictions on new data.
  - Train the new layers on your new dataset.
  - (optional) Fine-tune your model by unfreezing all layers (or part of them) and retraining them on the new data with a low learning rate (task specific improvements by incremental adaptation of pretrained features).



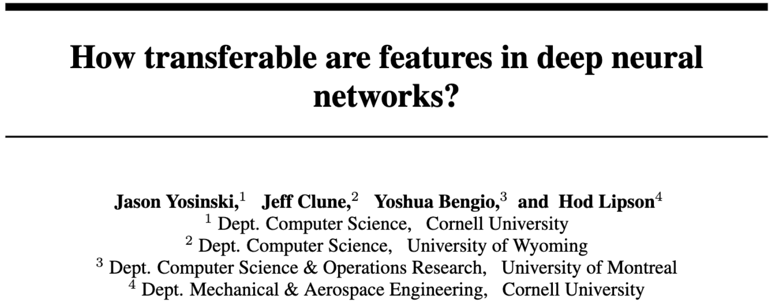
Potential issues

When transfer learning fails

- Despite its potential, transfer learning can fail when
  - **high-level features** at the end of the base model **are insufficient** to differentiate classes in the target dataset. E.g., a pretrained model could be excellent at identifying windows but struggle to determine whether a window is closed or open. Retraining base layers can help in this case.
  - the **target dataset is too small** to provide sufficient information to optimise the trainable layers, which results in overfitting on the new dataset.
  - the **datasets are not very similar**. E.g., in the case that the base model was optimised on animal pictures, it might struggle to identify images of plants and transferring features may lead to poor accuracies.

Learnt features

General vs. specific



How transferable are features in deep neural networks?

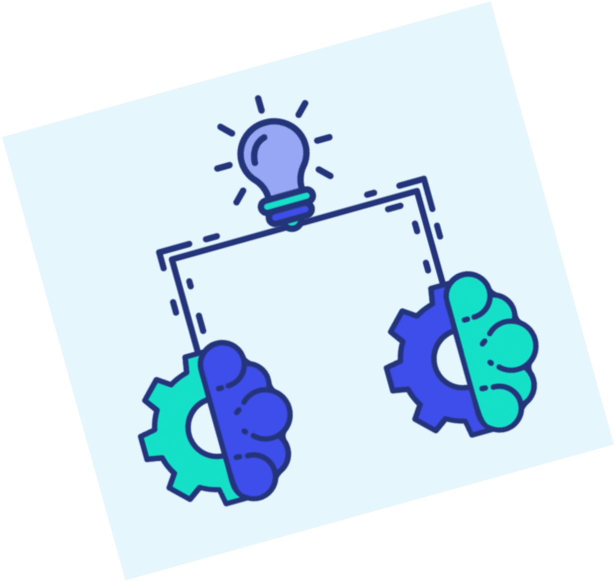
Jason Yosinski,[1] **Jeff Clune,**[2] **Yoshua Bengio,**[3] **and Hod Lipson**[4]
[1] Dept. Computer Science, Cornell University
[2] Dept. Computer Science, University of Wyoming
[3] Dept. Computer Science & Operations Research, University of Montreal
[4] Dept. Mechanical & Aerospace Engineering, Cornell University

- The discussion so far raises the question:

How transferable are features in deep neural networks?

- Yosinski et al. (2014) break down this question into 3 further questions:

1. Can we quantify the degree to which a particular layer is general or specific?
2. Does the transition occur suddenly at a single layer, or is it spread out over several layers?
3. Where does this transition take place: near the first, middle, or last layer of the network?



General features

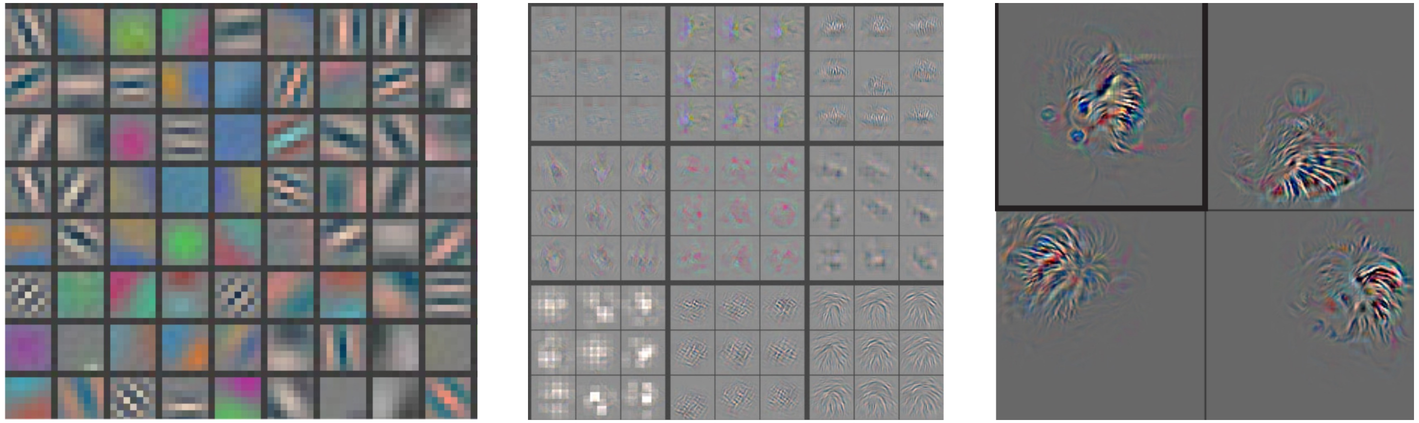
Gabor filters

- From experience, we know that neural networks learn general features first, while the last layers encode problem-specific features.

General features

Gabor filters

- From experience, we know that neural networks learn general features first, while the last layers encode problem-specific features.
- In particular, neural networks trained on images tend to first learn features that are similar to so-called **Gabor filters** or **colour blobs** independent of the dataset or objective (supervised vs. unsupervised ML).

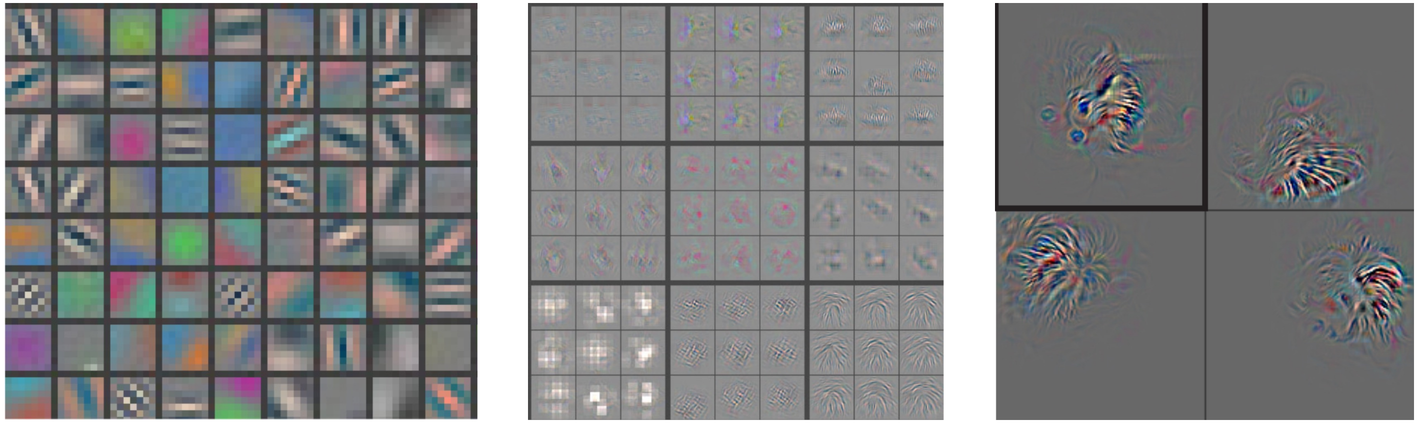


See Zeiler and Fergus (2013) for visualisations of feature maps.

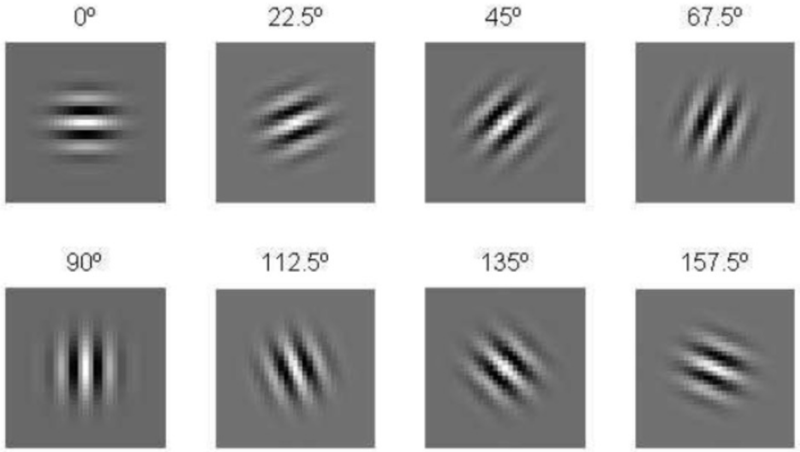
General features

Gabor filters

- From experience, we know that neural networks learn general features first, while the last layers encode problem-specific features.
- In particular, neural networks trained on images tend to first learn features that are similar to so-called **Gabor filters** or **colour blobs** independent of the dataset or objective (supervised vs. unsupervised ML).



See Zeiler and Fergus (2013) for visualisations of feature maps.



Gabor filters from Ferreira et al. (2009)

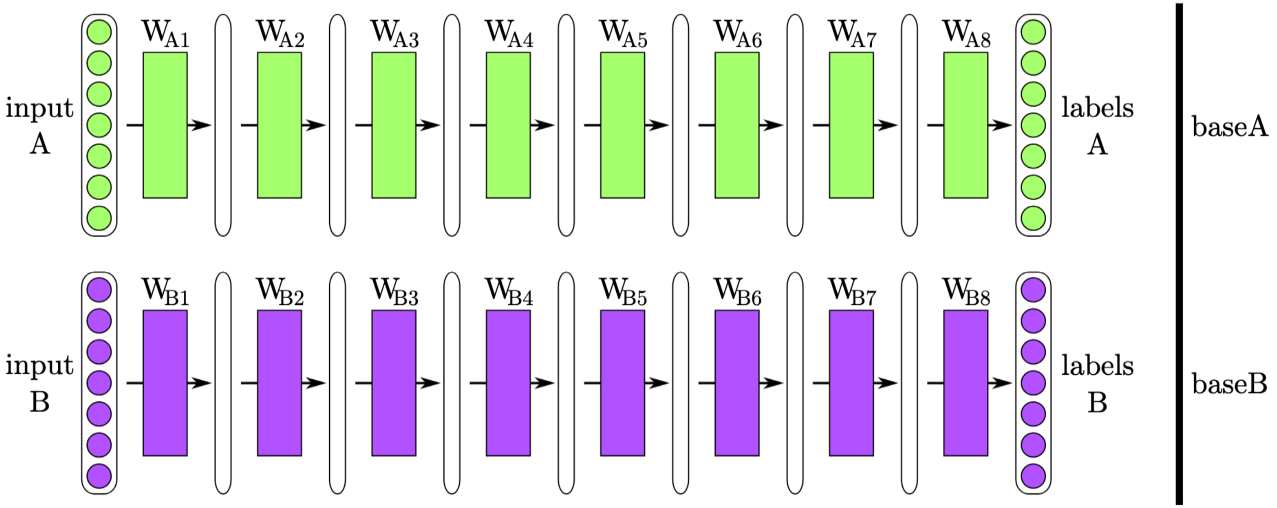
General vs. specific features

A systematic approach (Yosinski et al. 2014) I

- To understand the details of the transition from generic to specific Yosinski et al. used a **systematic approach** to analyse learnt features.

- For this experiment, they first create **two tasks A and B** by randomly splitting the 1000 ImageNet classes into two groups each containing 500 classes and around 645,000 example images.

- They then train two eight-layer CNNs on both datasets to obtain the **baseA and the baseB networks**.

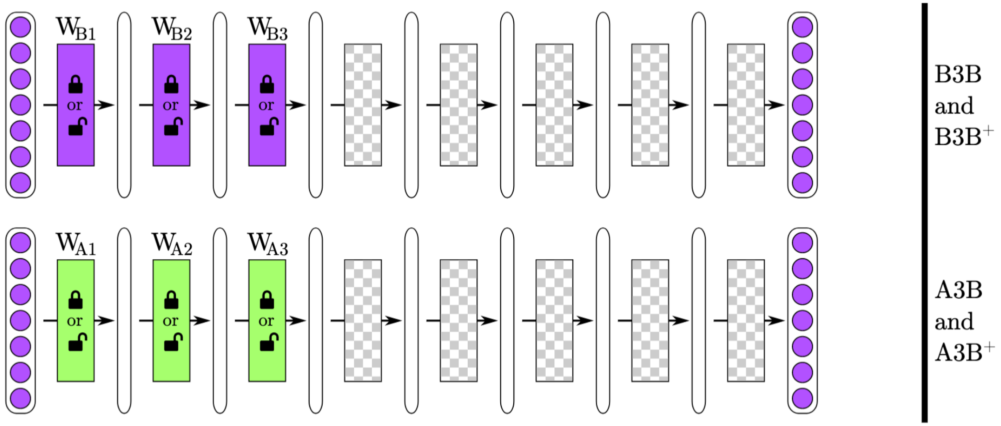


General vs. specific features

A systematic approach (Yosinski et al. 2014) II

- Next, they copied the first n layers of the base networks and **trained several new networks** (in the example below, we set n=3) :
  - A *selffer* (control) network B3B: first 3 layers are copied from baseB and frozen. The remaining 5 layers are initialised randomly and then trained on B.
  - A *transfer* (test) network A3B: first 3 layers are copied from baseA and frozen. The remaining 5 layers are initialised randomly and then trained on B.
  - Add the cases, B3B$^+$ and A3B $^+$ where first three layers are not frozen, but all can learn.
  - Repeat in both directions and learn B3A, etc.

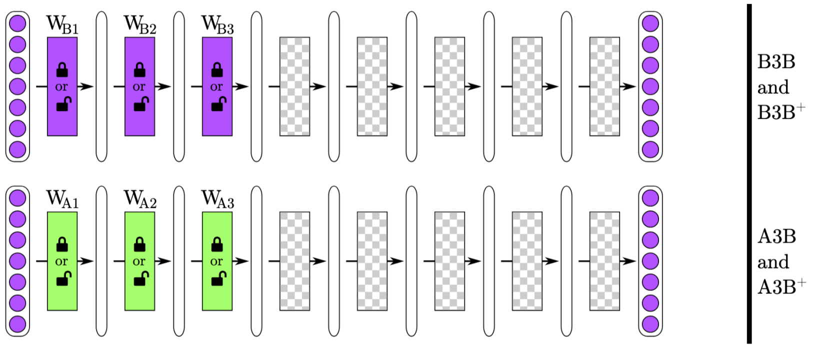
The idea: if A3B performs as well as baseB, there is evidence that the n=3 layer features are general. If the performance suffers, then the layers are specific to A.

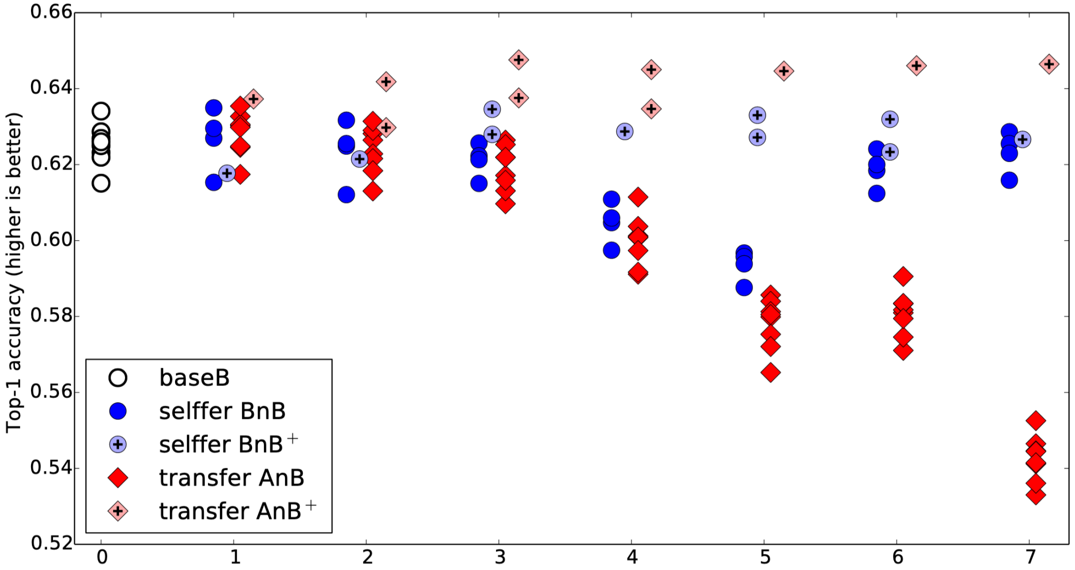


General vs. specific features

Results (Yosinski et al. 2014) I

- By studying the accuracy of the networks for different n, they find
  - Saving the first (n=1,2) layers of baseB and retraining on the same dataset B, the same performance can be achieved.
  - Saving the first n=3,4,5,6 layers of baseB and retraining on the same dataset B, the **performance surprisingly drops**.
  - This is due to so-called **co-adaption** between different layers, i.e., the features are not independent but interact in a complex, fragile way with each other. This cannot be relearnt in the upper layers alone.
  - Drop does not appear for the fine-tuned network BnB⁺, where all layers can learn.

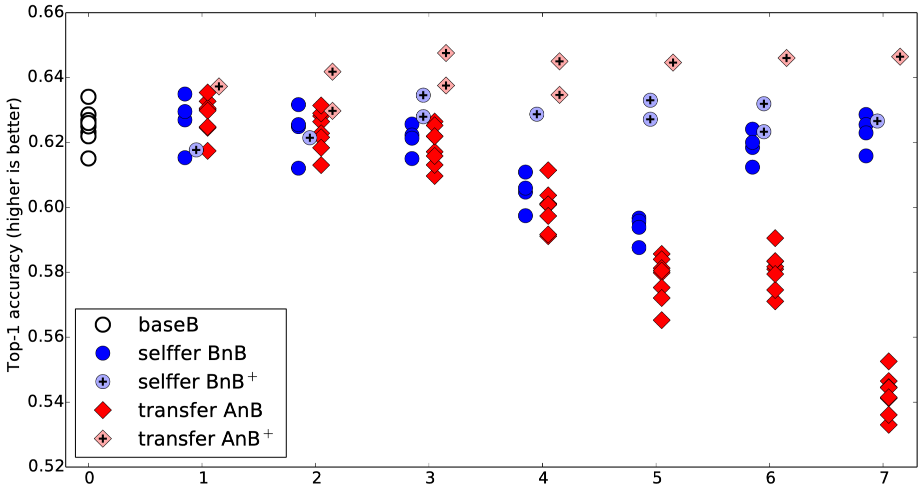


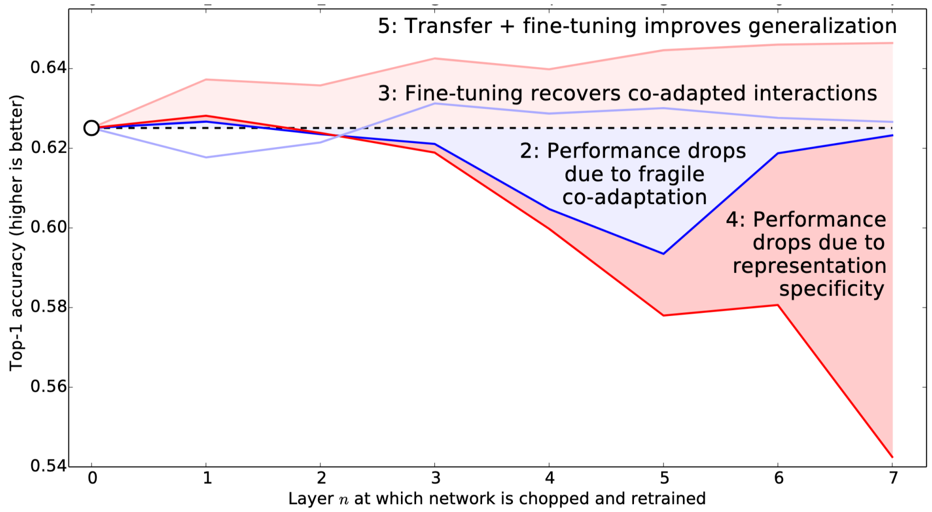


General vs. specific features

Results (Yosinski et al. 2014) II

- By studying the accuracy of the networks for different n, they find
  - Saving the first (n=1,2) layers of baseA and retraining on the dataset B, the same performance can be achieved. **Features in the first two layers are general**!
  - Saving the first (n=3,4,5,6,7) layers of baseA and retraining on the B, the **performance drops** due to (i) co-adaption (for layers 3,4,5) and (ii) more specific features (for layers 6,7).





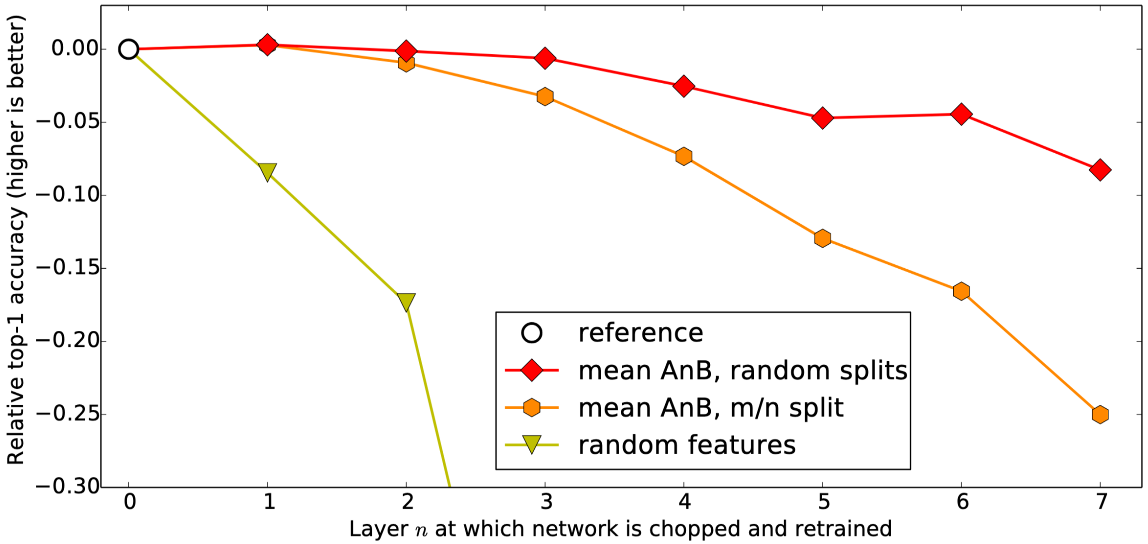
Transferring features and then fine-tuning them (AnB$^+$) results in networks that generalise better than those trained directly on the target dataset (BnB$^+$).

General vs. specific features

Results (Yosinski et al. 2014) III

- They repeated the experiment with two different sets A and B that were not randomly sampled from ImageNet (e.g., A and B are similar and share some overlap) but instead hand-picked to create **different A and B sets**.

The transferability gap when using frozen features grows more quickly as n increases for dissimilar tasks (hexagons) than similar tasks (diamonds). The drop is 8% vs 25% in the final layer for similar tasks vs dissimilar tasks. However, transfer learning performs still better than a random initialisation.

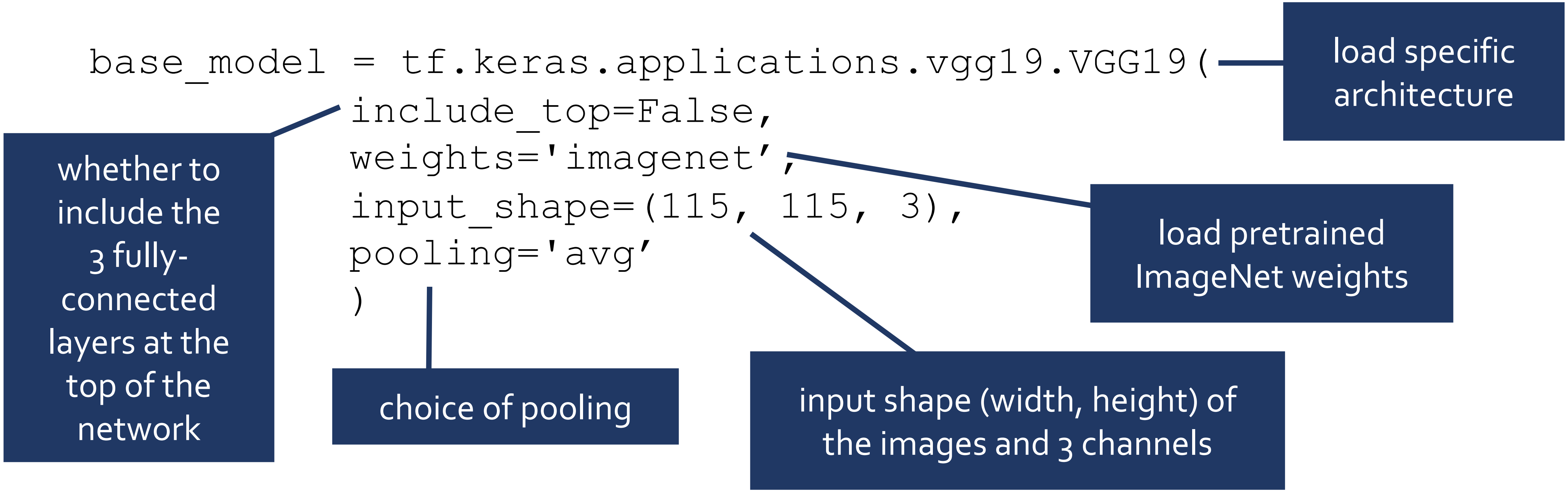


Python implementation

TensorFlow and Keras library

- As you will see during the **first tutorial**, implementing a transfer learning approach with TensorFlow and Keras is straightforward.

- Loading a specific model can be achieved as follows:



```
base_model = tf.keras.applications.vgg19.VGG19(
                include_top=False,
                weights='imagenet’,
                input_shape=(115, 115, 3),
                pooling='avg’
                )
```

load specific architecture

whether to include the 3 fully-connected layers at the top of the network

load pretrained ImageNet weights

choice of pooling

input shape (width, height) of the images and 3 channels

Python implementation II

TensorFlow and Keras library

- Once the base model is loaded, we can add additional layers:

```
model = tf.keras.models.Sequential()
model.add(base_model)
model.add(tf.keras.layers.Dense(256,
     activation='relu'))
model.add(tf.keras.layers.Dense(5,
     activation='softmax'))
base_model.trainable=False
```

make an instance of the sequential class

add base model

add a dense layer with 256 neurons and Rectified Linear Unit activation function

add a dense layer with 5 neurons and softmax activation function for a classification task

freeze the base model

Python implementation II

TensorFlow and Keras library

- Once the base model is loaded, we can add additional layers:

```
model = tf.keras.models.Sequential()
model.add(base_model)
model.add(tf.keras.layers.Dense(256,
     activation='relu'))
model.add(tf.keras.layers.Dense(5,
     activation='softmax'))
base_model.trainable=False
```

make an instance of the sequential class

add base model

add a dense layer with 256 neurons and Rectified Linear Unit activation function

add a dense layer with 5 neurons and softmax activation function for a classification task

freeze the base model

See first tutorial for more details!

Question time

Mentimeter quiz

Let’s take a few minutes to recap what we have discussed so far:

Go to menti.com and enter the code 3356 0743.

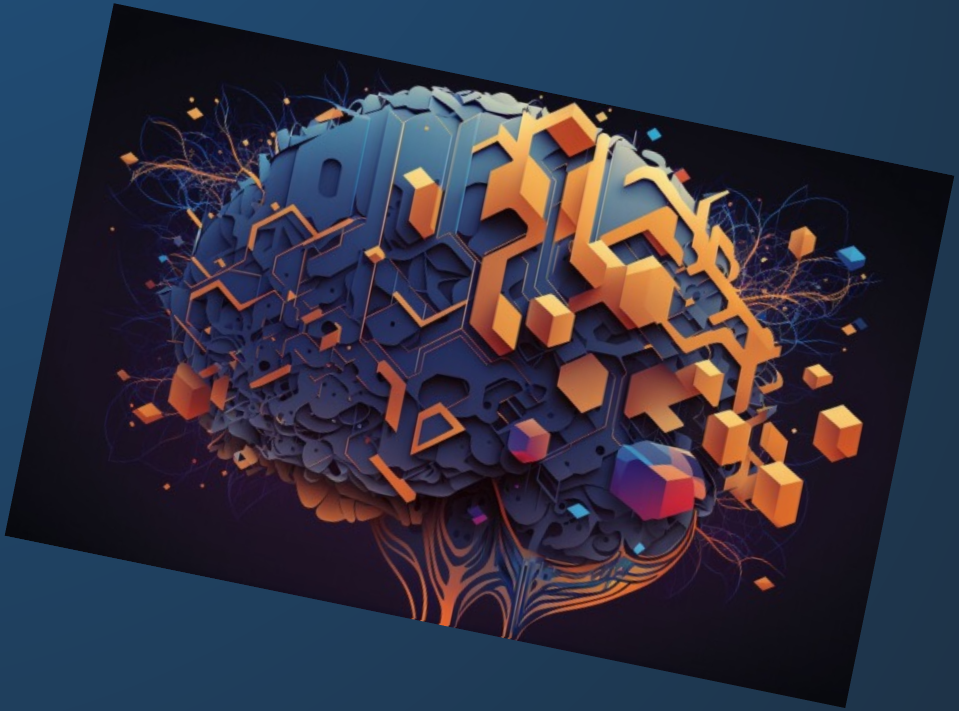
Course overview

Image classification: the basics

Transfer learning

Self-organising maps

Summary



The lack of labels

Supervised vs. unsupervised learning

- We discussed earlier how CNNs learn by defining a loss function and performing **gradient descent through back propagation**.

- However, this **supervised optimisation** approach **requires labels**. These are often not available or difficult to come by, e.g., in object detection or image segmentation.

There are alternative artificial neural network approaches that are unsupervised and closely inspired by the analogy with our human brains.

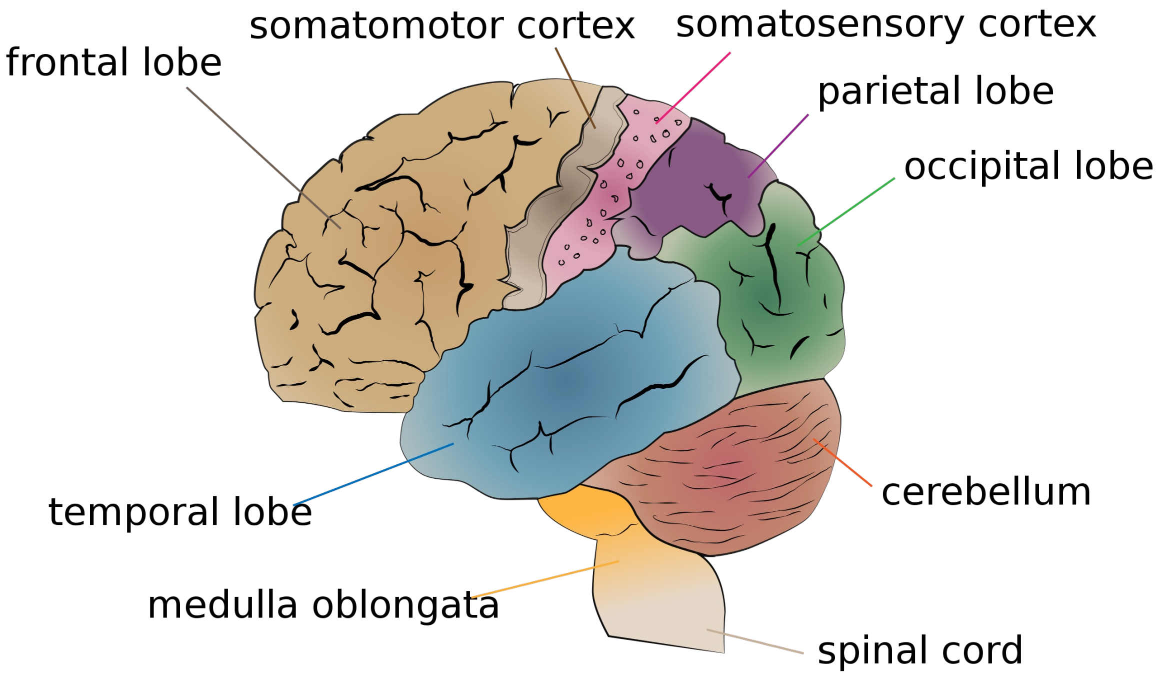
Self-organising maps (SOMs)

Self-organising maps

Overview

- Self-organising maps are also referred to as Kohonen maps/networks (Kohonen 1982) perform unsupervised learning by using a **competitive learning algorithm** that is based on **dimensionality reduction**.

- These networks are closely connected to the neurobiological concept of **topographic map formation**. In these maps, a spatial location of an output neuron corresponds to a particular domain or feature from the complex input space.

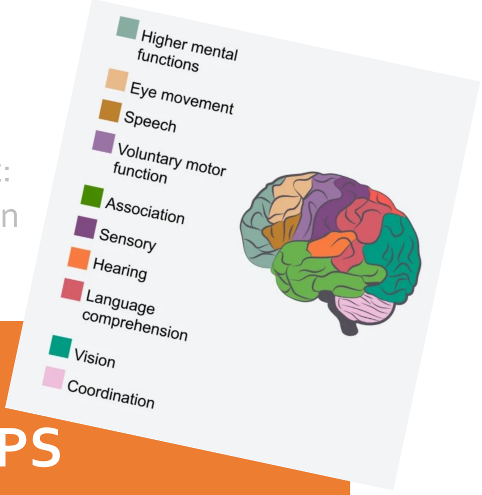


Credit: Jkwchui

Connection to topographic maps

Brains vs SOMs

Credit: eveleen



BRAINS

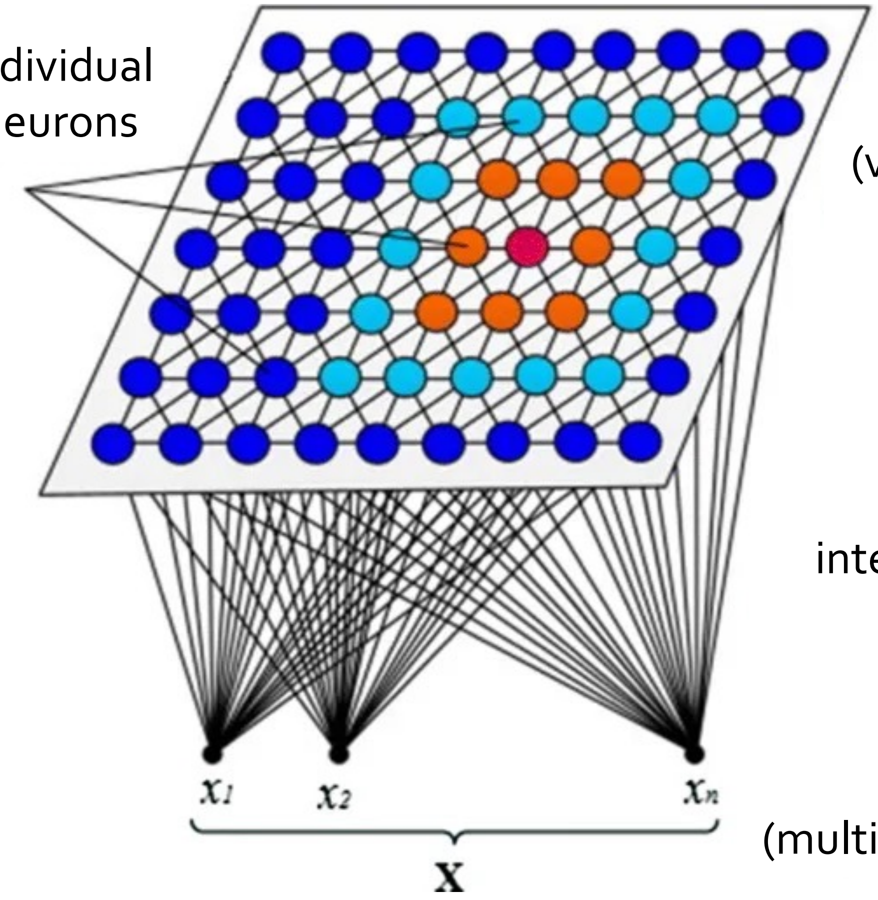
- Different regions of the cerebral cortex are specialising in processing specific features or functions like visual processing, motor control and language.
- Different regions are organised in a spatial manner with adjacent areas representing similar features/functions. This spatial organisation is often referred to as topographic mapping.

SELF-ORGANISING MAPS

- SOMs are designed to represent patterns in input data in a way that preserves their similarity. Neighbouring nodes respond to similar input patterns, enabling the extraction of features.
- SOMs are also organised spatially which allows them to represent high-dimensional input in a low-dimensional (typically 2D) space.

Self-organising maps

An example



individual neurons

(visible) lower-D output layer

interconnected weights

(multi-D) input layer

Adapted from Faradonbeh et al. (2019)

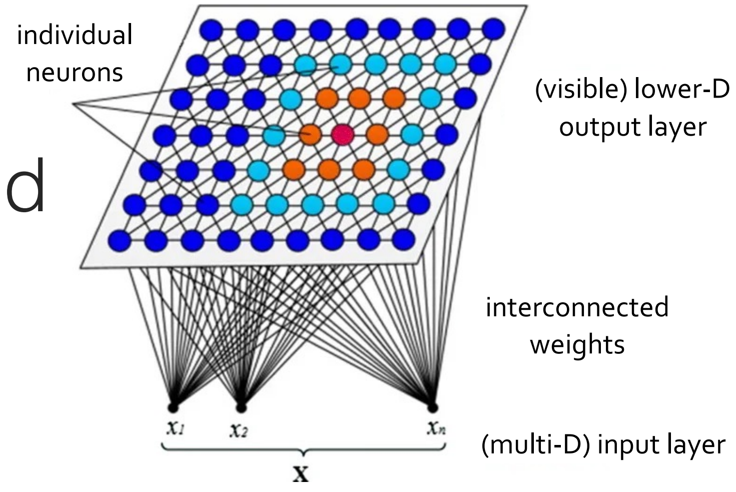
- In an SOM, weights belong to individual output neurons and **no activation functions** are present.
- Instead of being the result of a sum of weights as in standard CNN training, the SOM output neurons are directly represented by their weights (coordinates).

SOMs allow us to uncover categories within complex input data, independent of the input dimensions.

Self-organisation

Steps 1 and 2 – Initialisation and Competition

Adapted from Faradonbeh et al. (2019)



individual neurons

(visible) lower-D output layer

interconnected weights

(multi-D) input layer

- In the SOM, neural network neurons (also called nodes or reference vectors) are arranged in a single rectangular or hexagonal 2D grid. The network training then proceeds in four steps.

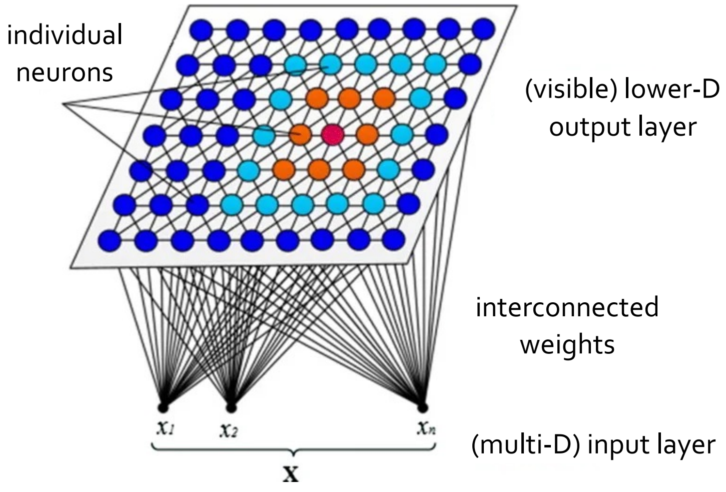
- **Step 1 – Initialisation**: Weights are initialised with random values.

- **Step 2 – Competition**: For a random input vector, we determine the node for which the weights are closest to the input (the Best Matching Unit, BMU) based on some discriminant function. A common choice is the squared Euclidean distance. This closest neuron (BMU) "**wins**" (the "*winner takes it all*" scheme).

Self-organisation

Steps 3 and 4 – Cooperation and Adaption

Adapted from Faradonbeh et al. (2019)



- **Step 3 – Cooperation:** The winning neuron determines the spatial location of a topological neighbourhood of excited neutrons. Over time, this neighbourhood will get smaller.

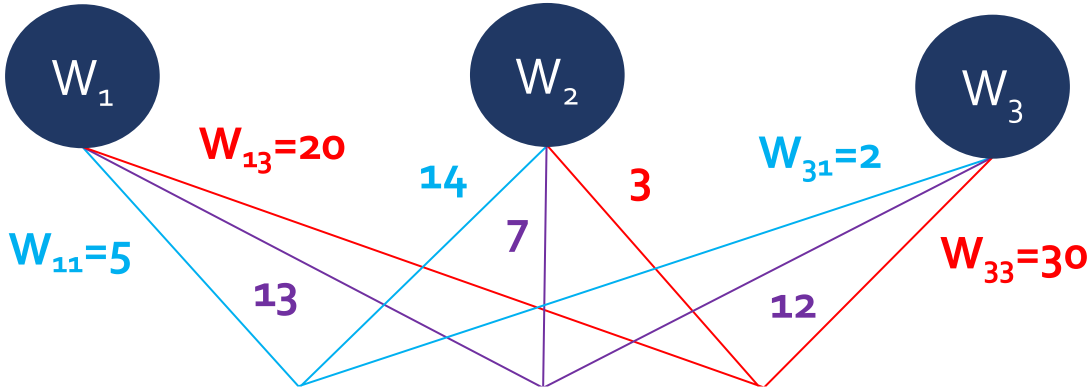
- **Step 4 – Adaption**: The winning weight vector (and the other excited neurons) are rewarded by adjusting their weights to resemble the sample input even more. The further away from the BMU, the less the weights get altered and the nodes learn.

Steps 2 to 4 are then repeated for all input vectors for a (large) number of cycles.

Self-organising maps

The training process in practice I

- Let us look at a specific example where we have a 1D output map with 3 neurons, W_1, W_2 and W_3.
- The corresponding weights W_{ij} are initialised at random.
- We are also given a range of input layers that correspond to 1D vectors with 3 entries, x_1, x_2, x_3.
- For our competitive learning, we calculate the Euclidean distance of each neuron for one test input.

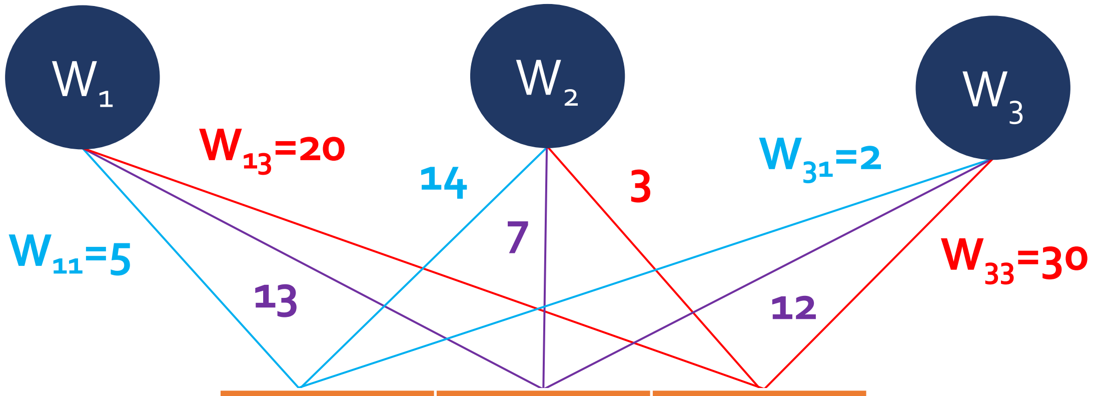


x_1	x_2	x_3
0.10	0.20	0.13
0.50	0.30	0.70
…	…	…

Self-organising maps

The training process in practice I

- Let us look at a specific example where we have a 1D output map with 3 neurons, W_1, W_2 and W_3.
- The corresponding weights W_{ij} are initialised at random.
- We are also given a range of input layers that correspond to 1D vectors with 3 entries, x_1, x_2, x_3.
- For our competitive learning, we calculate the Euclidean distance of each neuron for one test input.



x_1	x_2	x_3
0.10	0.20	0.13
0.50	0.30	0.70
…	…	…

$$d_1 = \|\mathbf{x} - \mathbf{W}_1\| = \sqrt{\sum_j^3 (x_j - W_{1j})^2}$$

$$= \sqrt{(0.5-5)^2 + (0.3-13)^2 + (0.7-20)^2} = 23.5$$

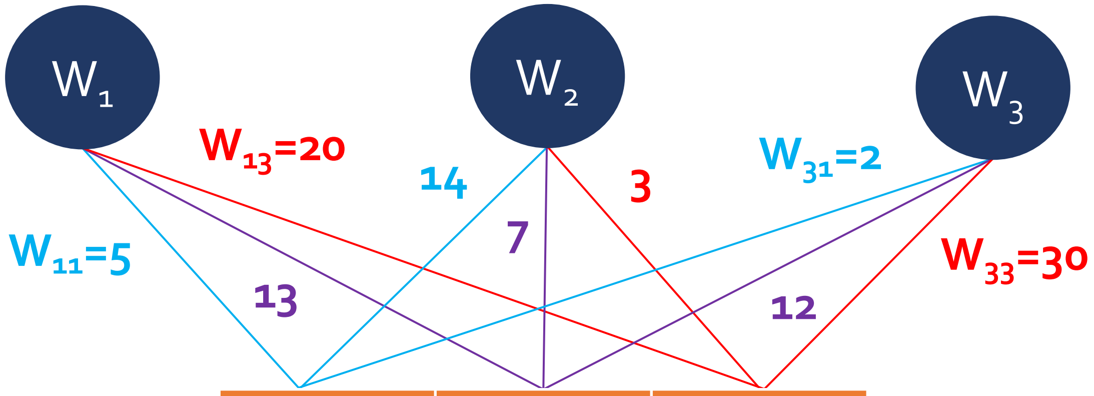
$$d_2 = 15.2$$

$$d_3 = 31.6$$

Self-organising maps

The training process in practice I

- Let us look at a specific example where we have a 1D output map with 3 neurons, W_1, W_2 and W_3.
- The corresponding weights W_{ij} are initialised at random.
- We are also given a range of input layers that correspond to 1D vectors with 3 entries, x_1, x_2, x_3.
- For our competitive learning, we calculate the Euclidean distance of each neuron for one test input.



x_1	x_2	x_3
0.10	0.20	0.13
0.50	0.30	0.70
…	…	…

$$d_1 = \|\mathbf{x} - \mathbf{W}_1\| = \sqrt{\sum_j^3 (x_j - W_{1j})^2}$$

$$= \sqrt{(0.5-5)^2 + (0.3-13)^2 + (0.7-20)^2} = 23.5$$

$d_2 = 15.2$

$d_3 = 31.6$

Self-organising maps

The training process in practice II

- We have found that W_2 is our winner neuron (BMU). In our case, the neighbourhood is simple as we only have two more neurons. However, in practice, we could also choose a Gaussian region around the BMU.

Self-organising maps

The training process in practice II

- We have found that W_2 is our winner neuron (BMU). In our case, the neighbourhood is simple as we only have two more neurons. However, in practice, we could also choose a **Gaussian region** around the BMU.

- The **update of our network weights** for iteration s+1 is then as follows

$$\mathbf{W}_{\mathrm{i}}^{s+1} = \mathbf{W}_{\mathrm{i}}^{s} + \mathrm{LR}^{s} * \theta_{\mathrm{i}}^{s}(\mathrm{BMU}) * [\mathbf{x} - \mathbf{W}_{\mathrm{i}}^{s}]$$

where LR is the learning rate & $\theta_{\mathrm{i}}^{s}(\mathrm{BMU})$ the neighbourhood function:

Self-organising maps

The training process in practice II

- We have found that W_2 is our winner neuron (BMU). In our case, the neighbourhood is simple as we only have two more neurons. However, in practice, we could also choose a **Gaussian region** around the BMU.
- The **update of our network weights** for iteration s+1 is then as follows

$$\mathbf{W}_\mathrm{i}^{s+1} = \mathbf{W}_\mathrm{i}^{s} + \mathrm{LR}^{s} * \theta_\mathrm{i}^{s}(\mathrm{BMU}) * [\mathbf{x} - \mathbf{W}_\mathrm{i}^{s}]$$

where LR is the learning rate & $\theta_\mathrm{i}^{s}(\mathrm{BMU})$ the neighbourhood function:

$$\mathrm{LR}^{s} = \mathrm{LR}_0 \exp\left[-\frac{s}{s_n}\right]$$

$$\theta_\mathrm{i}^{s}(\mathrm{BMU}) \propto \exp\left[-\frac{\|\mathbf{W}_\mathrm{i}^{s} - \mathbf{W}_\mathrm{BMU}^{s}\|}{2\,(\sigma^{s})^2}\right]$$

$$\sigma^{s} = \sigma_0 \exp\left[-\frac{s}{s_n}\right]$$

Self-organising maps

The training process in practice II

- We have found that W_2 is our winner neuron (BMU). In our case, the neighbourho simple as we only have two more neurons. However, in practice, a **Gaussian region** around the BMU.

- The **update** then as follows

where LR is the learning rate & $\theta_i($ ood function:

The LR and the size of the neighbourhood σ depend on the epoch and several hyperparameters. Both decrease with time so that the map eventually converges.

$$LR^s = LR_0 \exp\left[-\frac{s}{s_n}\right]$$

$$\theta_i^s(\mathrm{BMU}) \propto \exp\left[-\frac{\|\mathbf{W}_i^s - \mathbf{W}_{\mathrm{BMU}}^s\|}{2\,(\sigma^s)^2}\right]$$

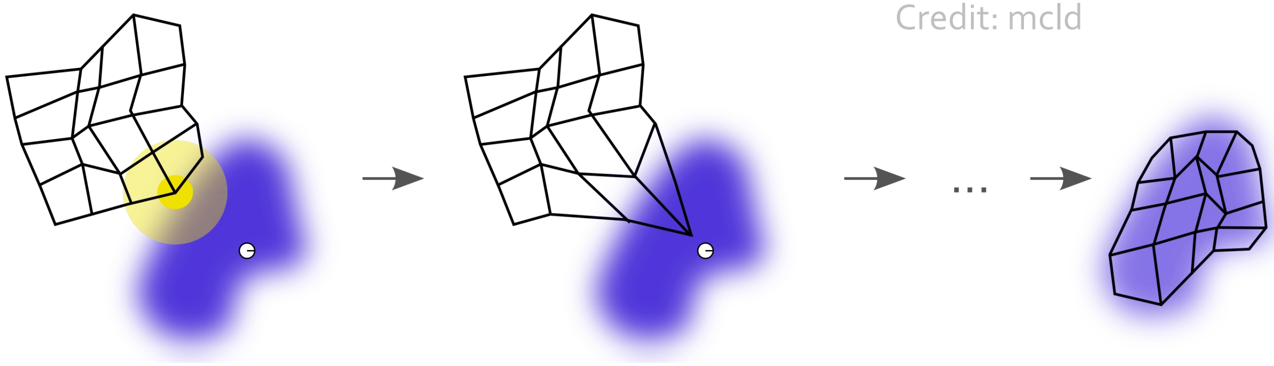
$$\sigma^s = \sigma_0 \exp\left[-\frac{s}{s_n}\right]$$

Self-organising maps

Dimensionality reduction

- We can illustrate this training process as follows:

Yellow neuron is the BMU which moves closer to the (white) training sample.



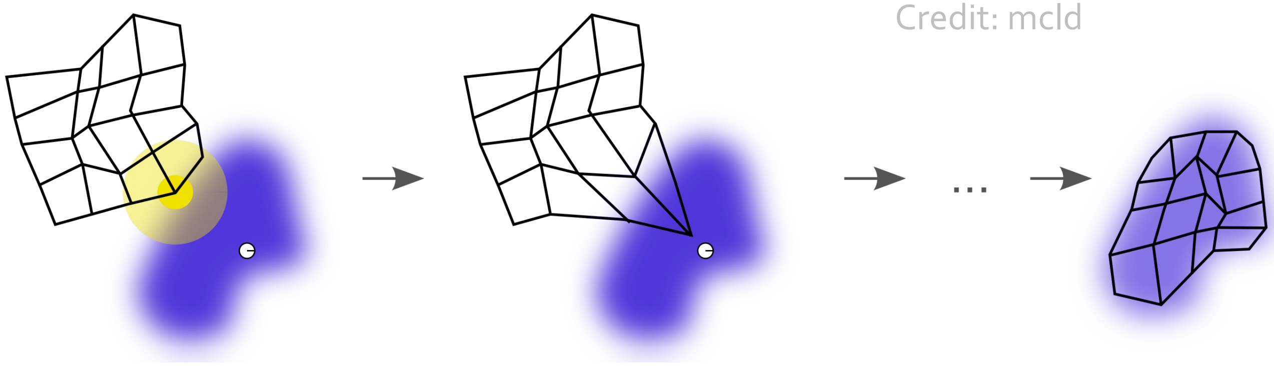
Credit: mcld

Self-organising maps

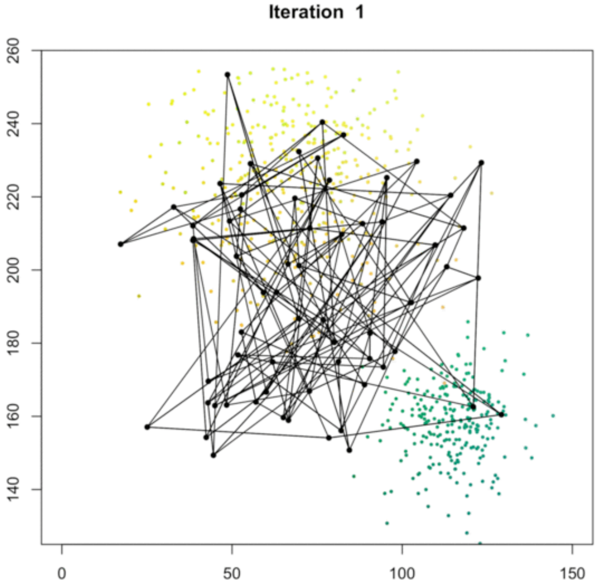
Dimensionality reduction

- We can illustrate this training process as follows:

Yellow neuron is the BMU which moves closer to the (white) training sample.



Credit: mcld



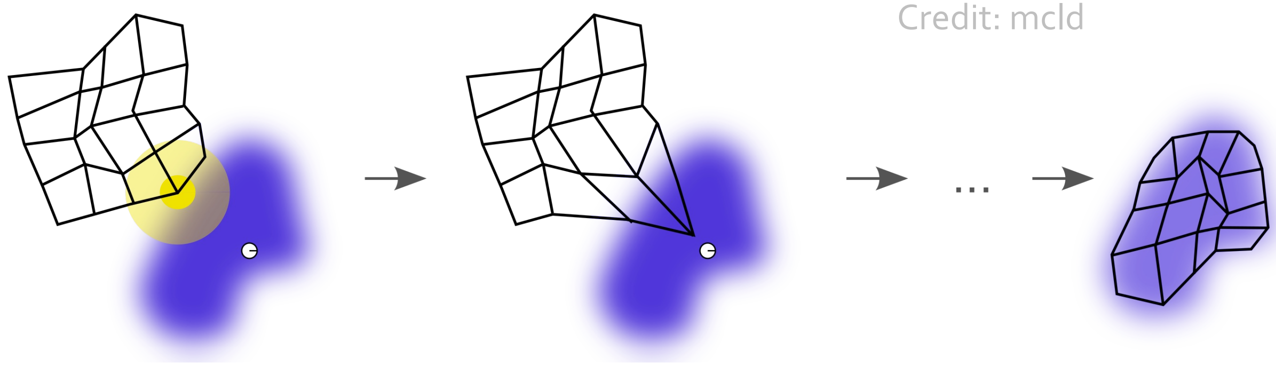
Credit: https://algobeans.com/2017/11/02/self-organizing-map/

Self-organising maps

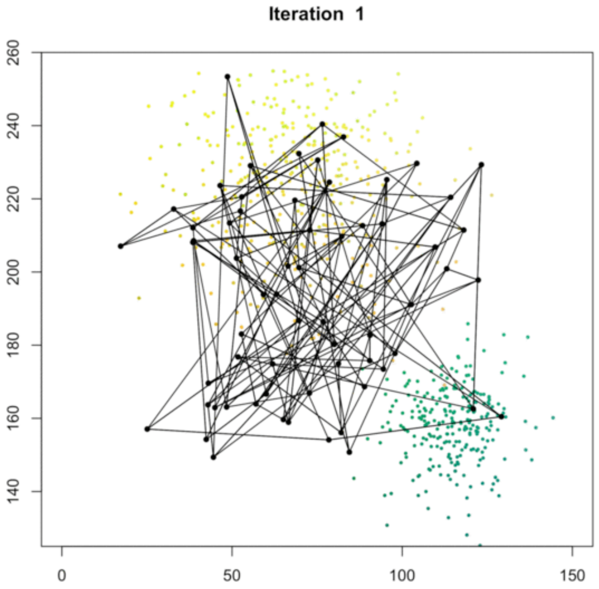
Dimensionality reduction

- We can illustrate this training process as follows:

Yellow neuron is the BMU which moves closer to the (white) training sample.



Credit: mcld



- Ultimately, dimensionality reduction in SOMs depends on the distance measure between neurons used to map the input data onto the 2D plane.

Credit: https://algobeans.com/2017/11/02/self-organizing-map/

Self-organising maps

General applications and some disadvantages

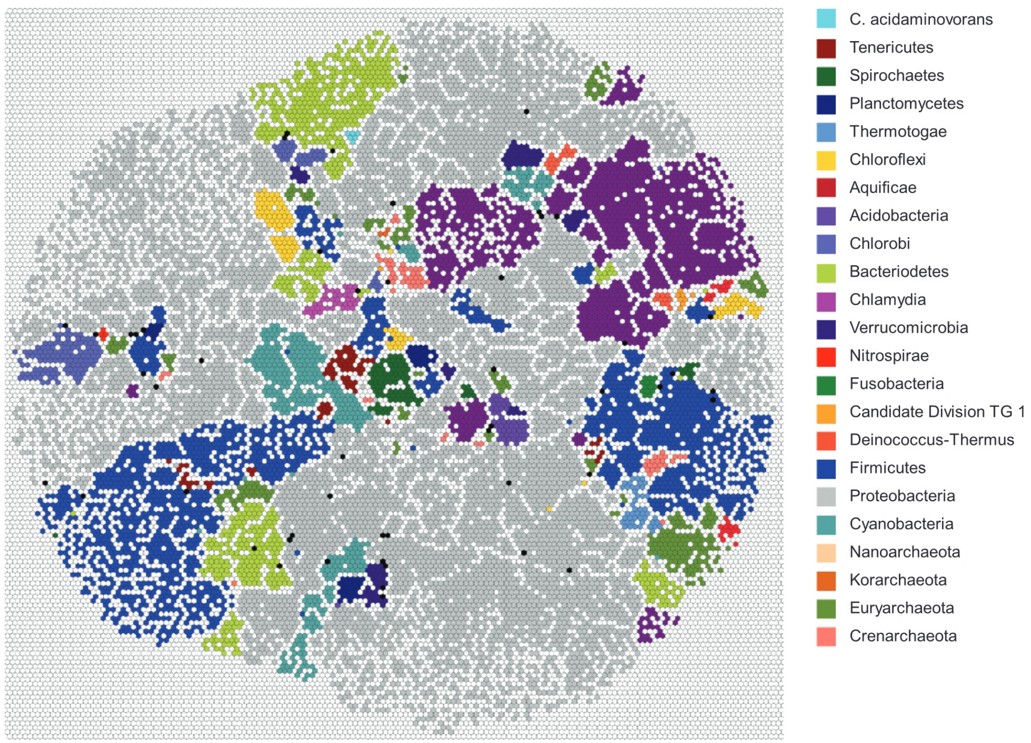
- Self-organising maps have found a **wide range of applications** where labelling data is often difficult. These include, for example,
  - Image classification
  - Text mining & information detection
  - Anomaly detection, e.g., in banking
  - Failure mode and effects analysis
  - Gene expression analysis
  - Finding representative data in large datasets
  - Astronomy wide-field surveys
- However, SOMs also have **some shortcomings**:
  - Training SOMs takes time due to distance calculations.
  - They do not build a generative data model and understand its formation.
  - SOMs can be difficult to train for categorical or mixed-type data.

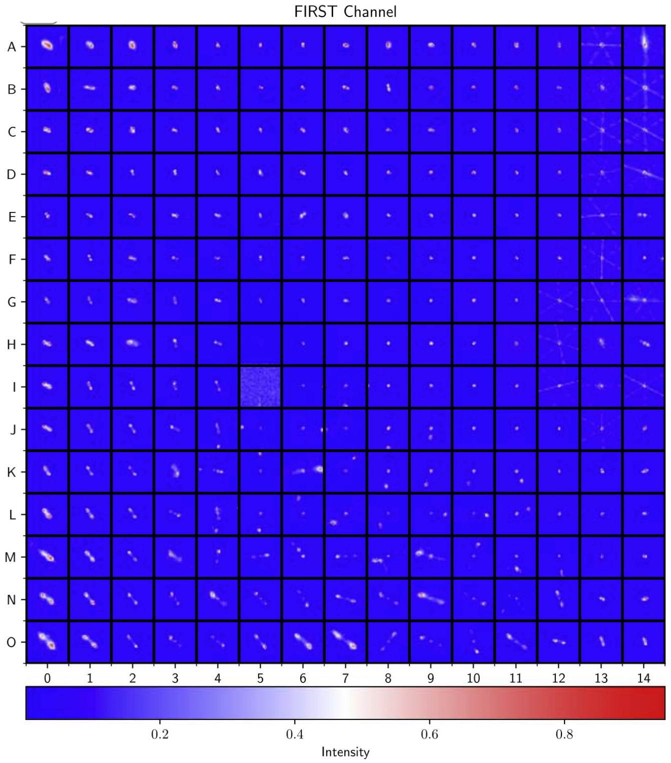
Self-organising maps

Two example applications

SOM of DNA sequences of Bacteria & Archaea to analyse biodiversity (Weber et al. 2011)

SOM of Radio Galaxy Zoo images to identify spatial features (Galvin et al. 2019)





Question time

Mentimeter quiz

Let’s take a few minutes to recap what we have discussed so far:

Go to menti.com and enter the code 3686 2739.

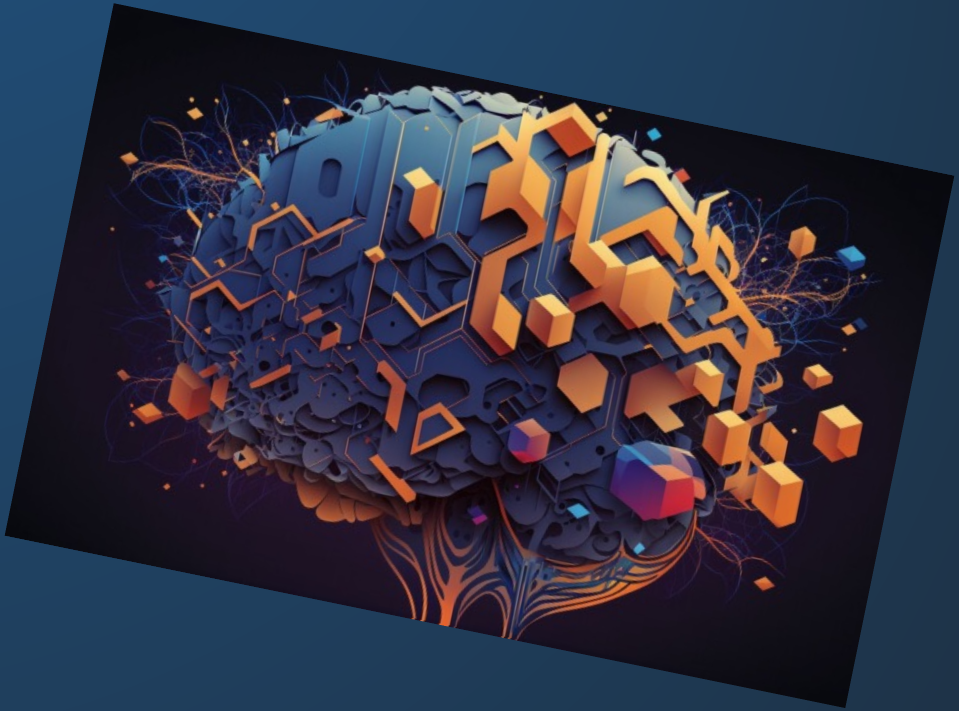
Course overview

Image classification: the basics

Transfer learning

Self-organising maps

Summary



Further reading

Summary of references discussed today

- Kohonen, *Self-organized formation of topologically correct feature maps*, Biological Cybernetics, 43, 59 (1982)
- Yin, T*he Self-Organizing Maps: Background, Theories, Extensions and Applications,* Studies in Computational Intelligence, 115, 715 (2008)
- Ferreira et al., *Modifications and Improvements on Iris Recognition,* Conference Proceedings, BIOSIGNALS, Portugal (2009)
- Weber et al., *Practical application of self-organizing maps to interrelate biodiversity and functional data in NGS-based metagenomics*, ISME Journal, 5, 918 (2011)
- Zeiler and Fergus, V*isualizing and Understanding Convolutional Networks*, preprint arXiv:1311.2901 (2013)
- Yosinski et al., *How transferable are features in deep neural networks?*, Advances in Neural Information Processing Systems 27 (NeurIPS 2014)
- Galvin et al., *Radio Galaxy Zoo: Knowledge Transfer Using Rotationally Invariant Self-organizing Maps*, PASP, 131, 108009 (2019)

Summary

- We discussed the course outline and your lecturer for this module.
- Image classification is a fundamental task in image processing, where we assign labels to images. This is generally simpler than object detection and segmentation, which we will cover next.
- Large labelled datasets like ImageNet were crucial to obtain the superior performance of neural networks in image classification.
- When labels are difficult to obtain or limited data is available, transfer learning allows us to use knowledge from previously trained models, because the first layers of a CNN learn generic image features.
- Additionally, self-organising maps can be used for unsupervised learning. Inspired by topographic maps, they allow us to reduce the dimensionality of complex data via a competitive learning approach.