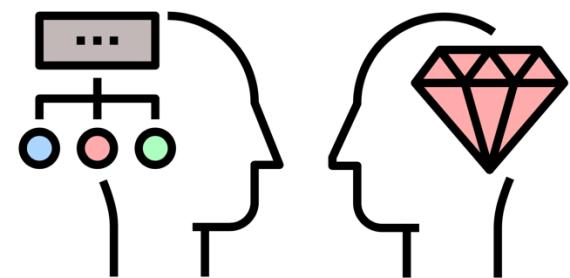


Machine Learning for Materials

6. Artificial Neural Networks

Aron Walsh

Department of Materials
Centre for Processable Electronics



Module Contents

1. Introduction
2. Machine Learning Basics
3. Materials Data
4. Crystal Representations
5. Classical Learning
- 6. Artificial Neural Networks**
7. Building a Model from Scratch
8. Accelerated Discovery
9. Generative Artificial Intelligence
10. Recent Advances

Class Outline

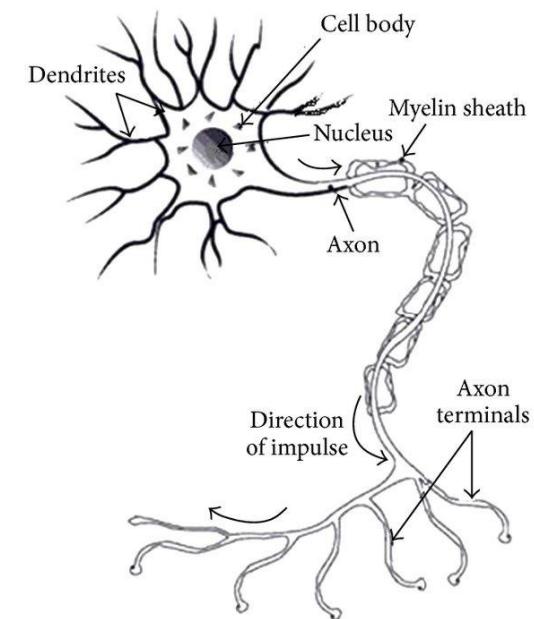
Artificial Neural Networks

- A. *From neuron to perceptron***
- B. *Network architecture and training***
- C. *Convolutional neural networks***

Artificial Neuron

Neurons transmit chemical and electrical signals in the brain. Artificial neurons mimic this behaviour using mathematical functions

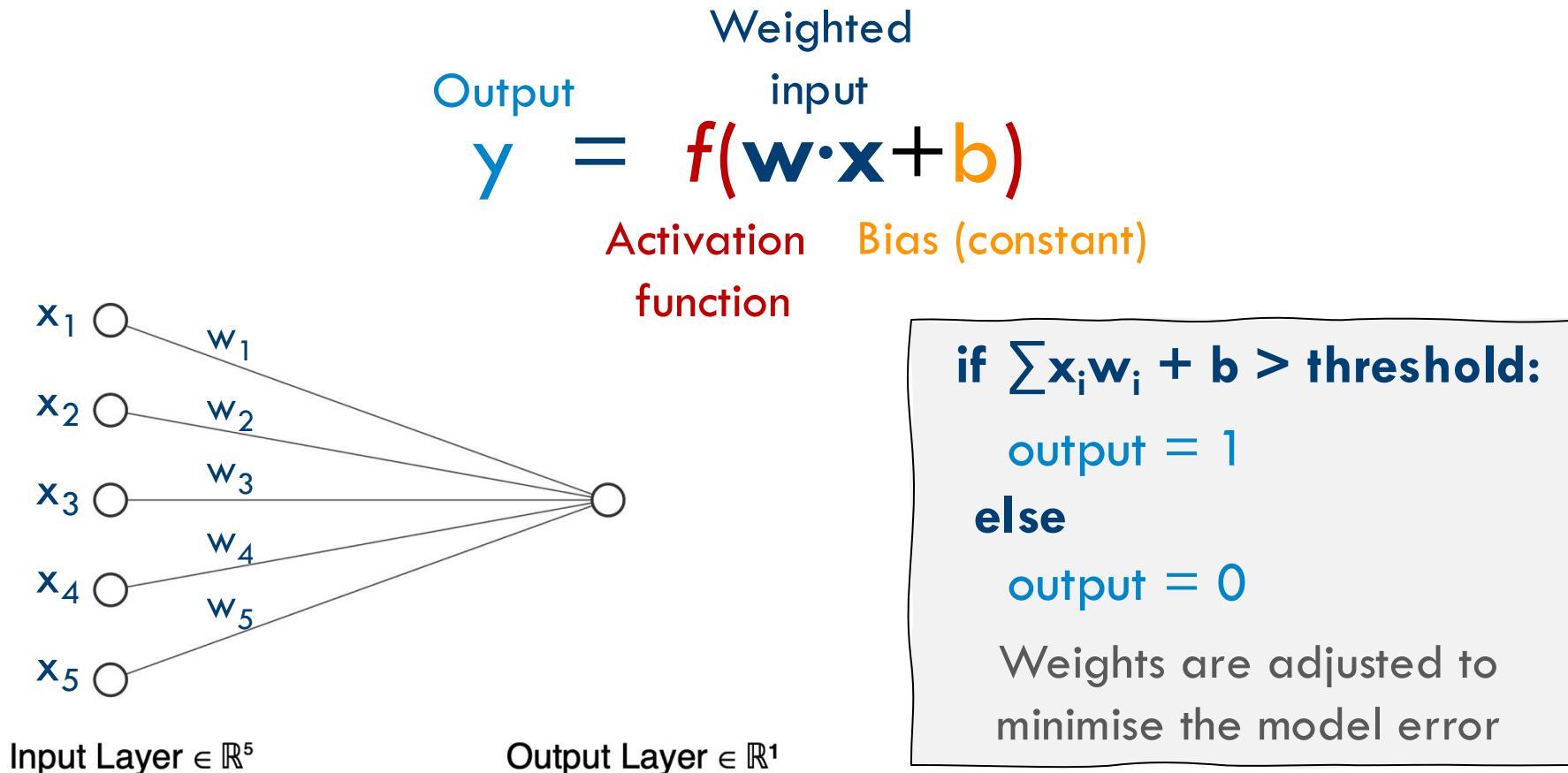
Biological neuron	Artificial neuron
Cell nucleus	Node
Dendrites	Input
Synapse	Weights (Interconnects)
Axon	Output



The human brain has $\sim 10^{11}$ neurons and 10^{15} synapses ($\sim 10^{15}$ FLOPS)

Artificial Neuron

The perceptron is a binary neural network classifier:
weighted inputs produce an output of 0 or 1





```
def perceptron(x, w, b):
    """
    Compute the output of a perceptron.

    Args:
        x (list of float): Input vector
        w (list of float): Weight vector
        b (float): Bias term

    Returns:
        int: Output of the perceptron (1 or 0)
    """

    # Calculate the weighted sum
    weighted_sum = sum(w_i * x_i for w_i, x_i in zip(w, x)) + b

    # Apply the activation function
    return 1 if weighted_sum > 0 else 0
```

The function “zip” pairs the elements of x and w together in tuples

Class Outline

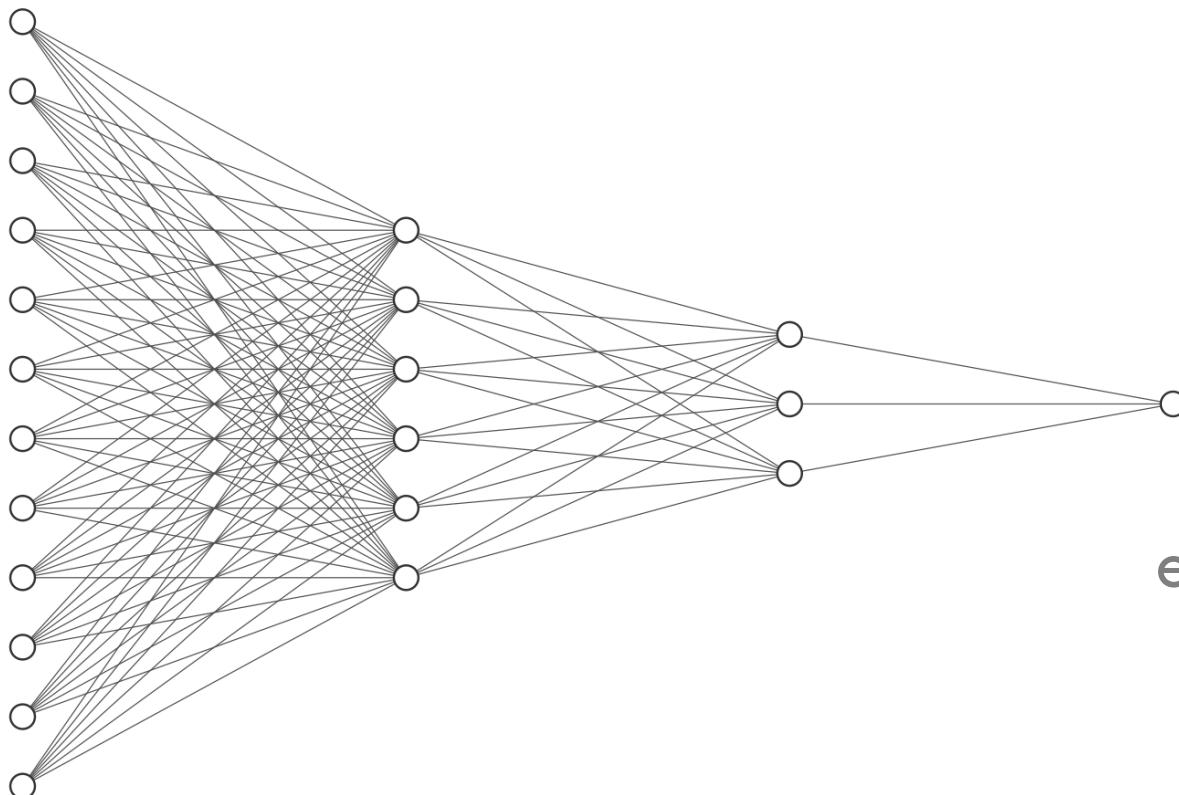
Artificial Neural Networks

- A. *From neuron to perceptron***
- B. *Network architecture and training***
- C. *Convolutional neural networks***

Neural Network Architecture

Basic neural network: One or two layers

Deep neural network: Three or more layers



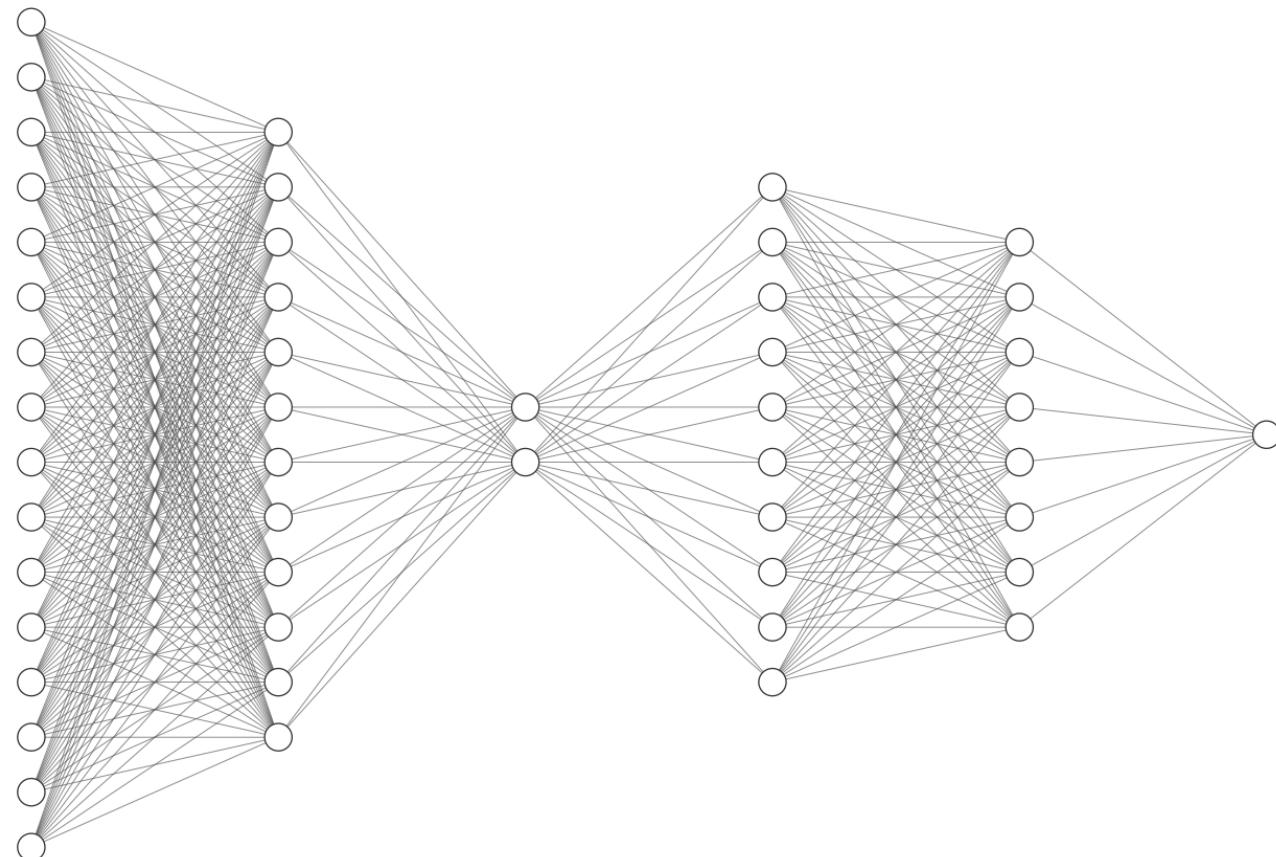
**Three layer
model**

(input layer is
excluded in counting)

Neural Network Architecture

Basic neural network: One or two layers

Deep neural network: Three or more layers



Five layer
model

Note the layer 2
bottleneck

Why? Compressed
representation

Input Layer $\in \mathbb{R}^{16}$

Hidden Layer $\in \mathbb{R}^{12}$

Hidden Layer $\in \mathbb{R}^2$

Hidden Layer $\in \mathbb{R}^{10}$

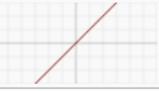
Hidden Layer $\in \mathbb{R}^8$

Output Layer $\in \mathbb{R}^1$

Activation Function

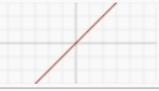
$w \cdot x + b$ is simply a linear combination.

Activation function $f(w \cdot x + b)$ introduces non-linearity

	Activation function	Derivative		
Identity		$f(x) = x$	$f'(x) = 1$	
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	Perceptron model
Logistic (a. k. a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	Popular in early models
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	Common for deep learning
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	Common for deep learning

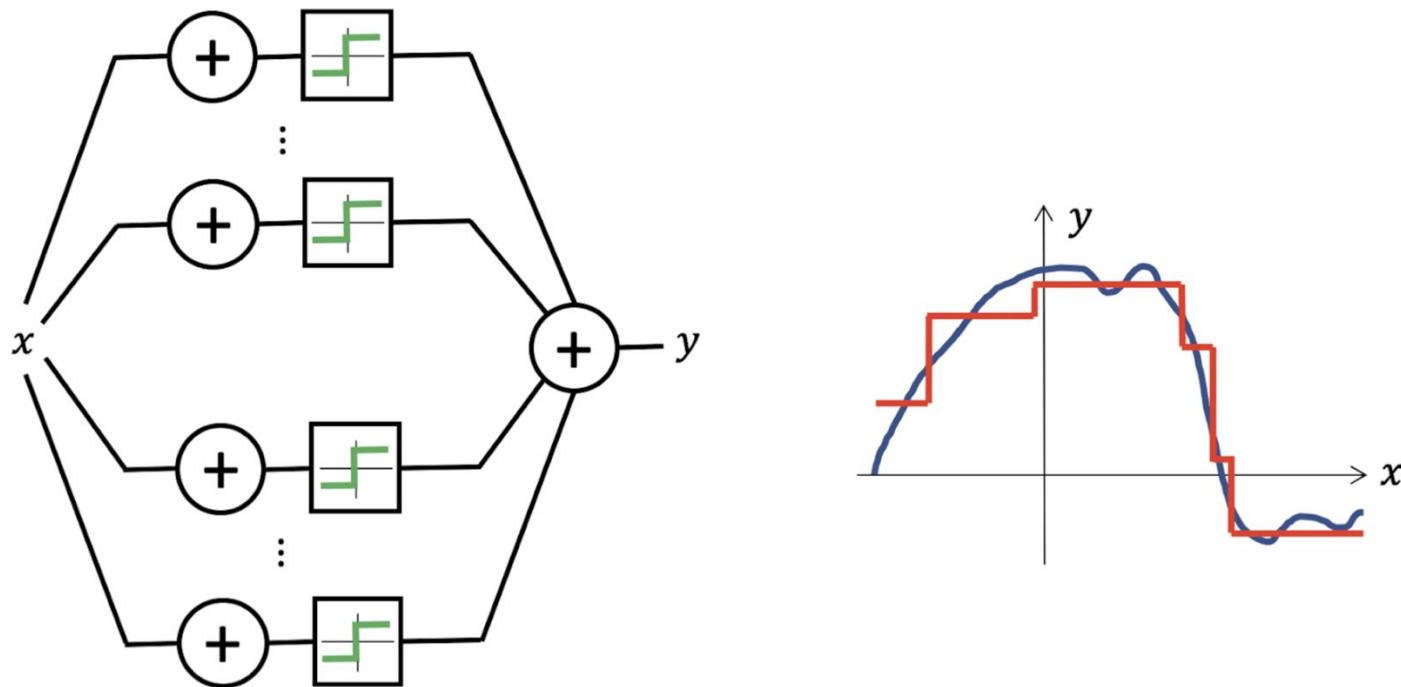
Activation Function

Corresponding weights and thresholds
are learned (fit) during model training

	Activation function	Derivative		
Identity		$f(x) = x$	$f'(x) = 1$	
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	Perceptron model
Logistic (a. k. a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	Popular in early models
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	Common for deep learning
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	Common for deep learning

Universal Function Approximators

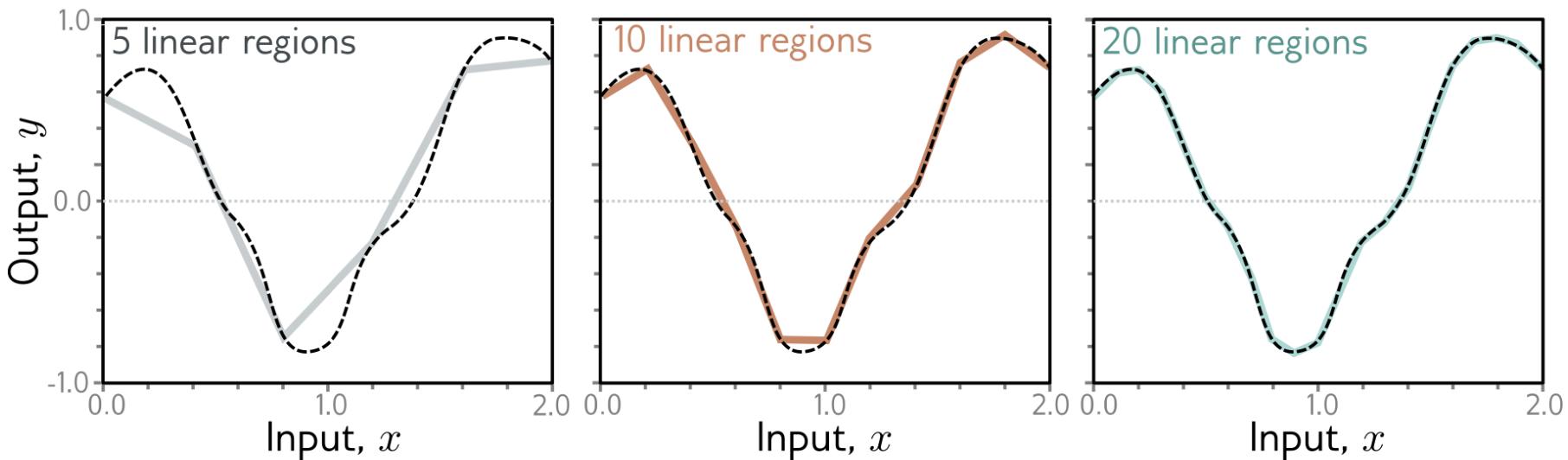
Multilayer neural networks can approximate any continuous function to any desired accuracy



Practical performance will depend on the number of hidden layers, choice of activation function, and training data

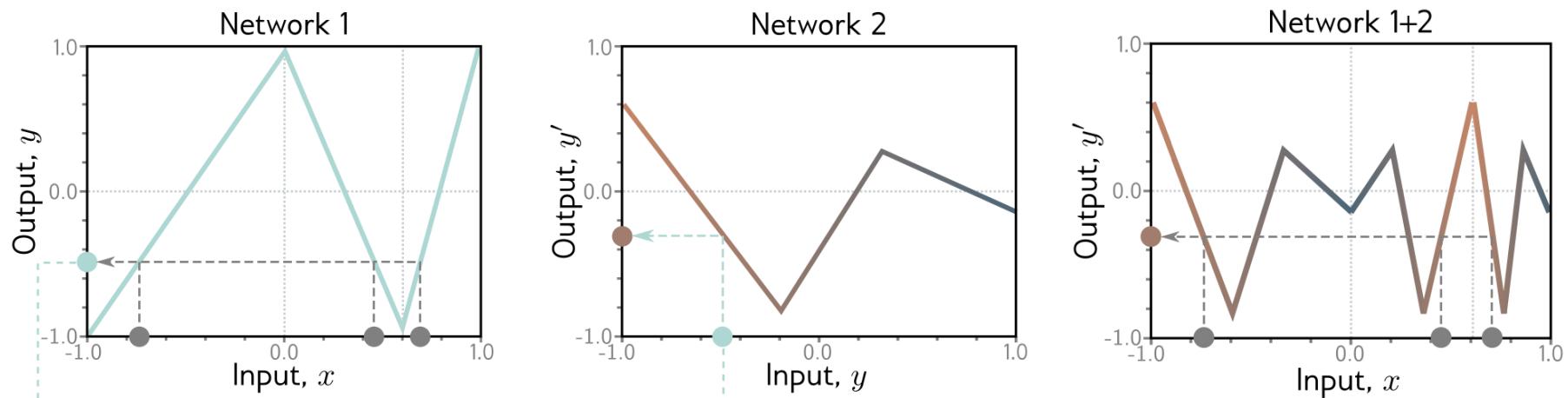
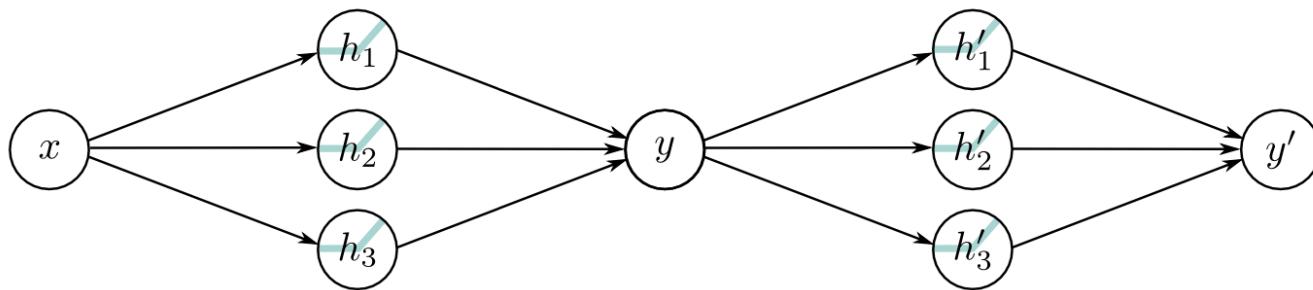
Universal Function Approximators

Multilayer neural networks can approximate any continuous function to any desired accuracy



Approximation of a 1D function (dashed line)
by a piecewise linear model

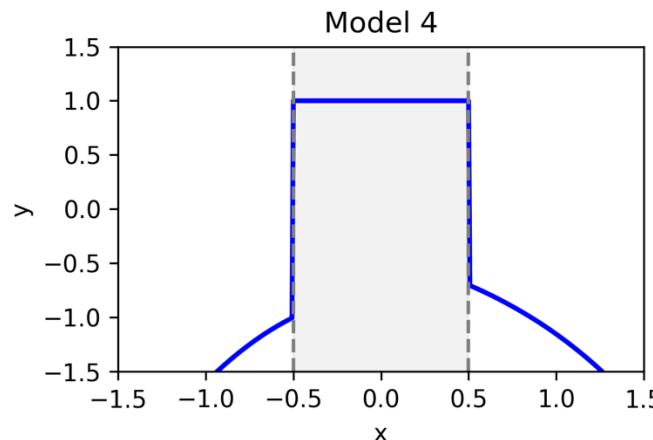
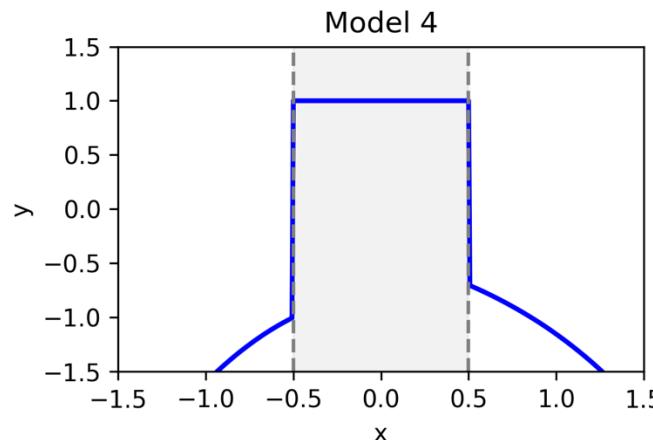
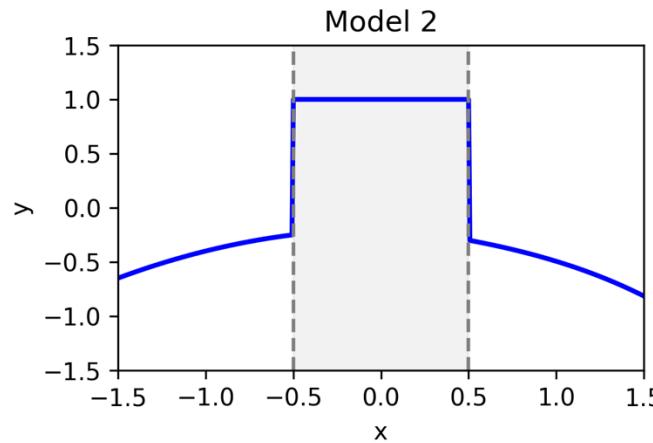
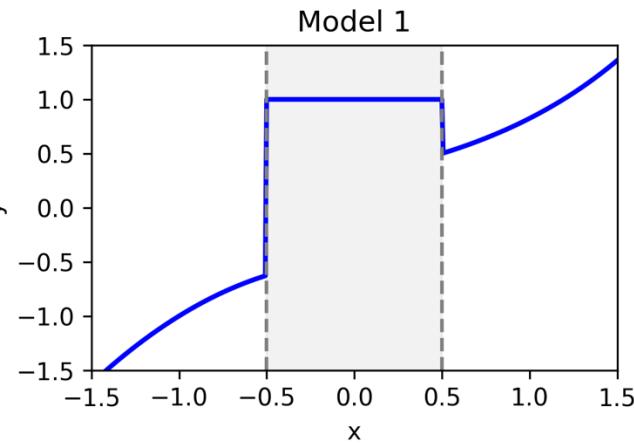
Universal Function Approximators



The combination of two single-layer networks
with three hidden units each

Universal Function Approximators

Extrapolation outside training region is
not guaranteed (no fixed functional form)



Four models with the
same performance
(in grey region)

Types of Layer in Deep Learning

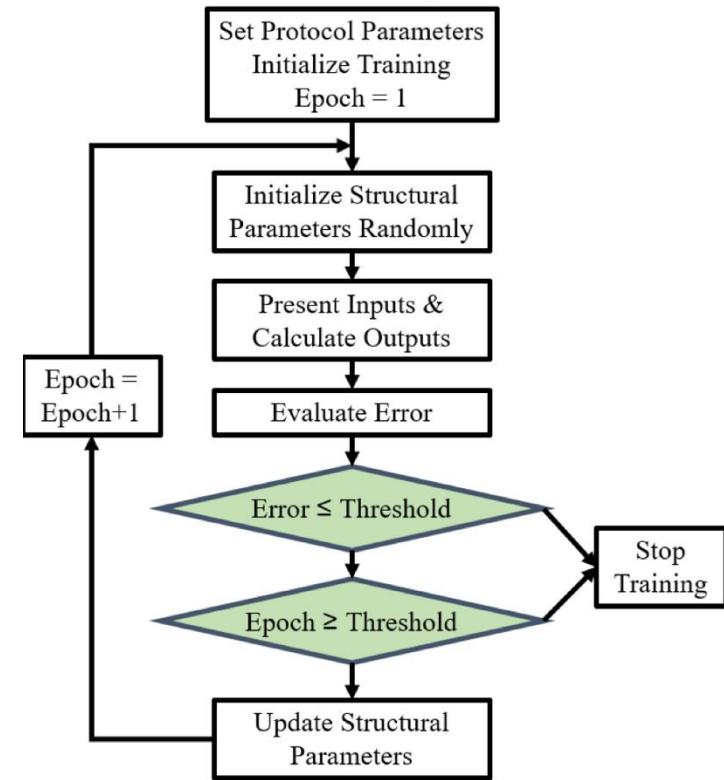
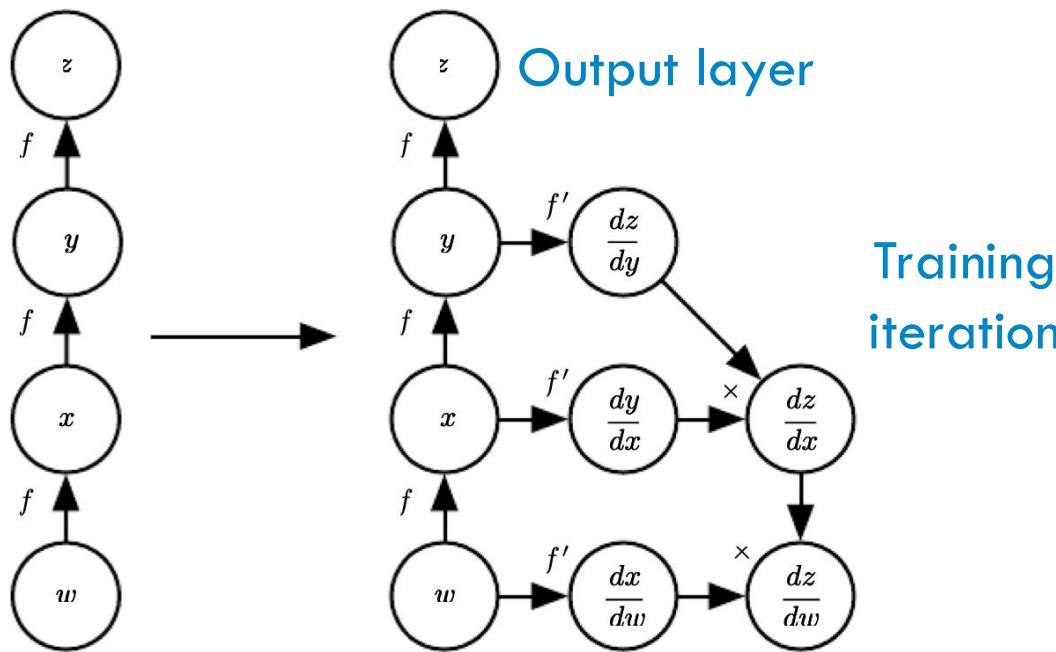
Layers are combined to learn representations and capture data patterns effectively

- **Dense (fully connected):** neurons connected to every other neuron
- **Convolutional:** filter applied to grid-like input, extracting features
- **Pooling:** reduce spatial dimensions, retaining key information
- **Recurrent:** incorporate feedback loops for sequential data flow
- **Dropout:** randomly zero out inputs to mitigate overfitting in training
- **Embedding:** map categorical variables into continuous vectors
- **Upscaling:** increase spatial resolution of feature maps

Backpropagation (Backprop)

An algorithm used to adjust the weights of a neural network using gradient descent (from the output layer)

Application of the chain rule



Backpropogation (Backprop)

Backprop efficiently computes gradients, enabling networks to learn parameters by error minimisation

Limitations

- Slow training
- Failure to converge
- Local minima

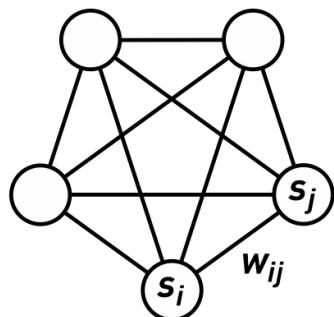
Improvements

- *Stochastic gradient descent*
(use random subset of data)
- *Batch normalisation*
(avoid vanishing gradients)
- *Adaptive learning rates*
(more robust convergence)

Nobel Prize in Physics 2024

“Foundational discoveries and inventions that enable machine learning with artificial neural networks”

Hopfield network



Deterministic
recurrent network of
binary nodes
(for local minima)

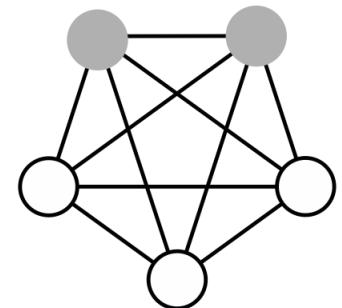


Ill. Niklas Elmehed © Nobel Prize Outreach

John J. Hopfield

Prize share: 1/2

Boltzmann Machine



Stochastic
recurrent network of
visible/hidden nodes
(for global minima)



Ill. Niklas Elmehed © Nobel Prize Outreach

Geoffrey E. Hinton

Prize share: 1/2

Class Outline

Artificial Neural Networks

- A. *From neuron to perceptron***
- B. *Network architecture and training***
- C. *Convolutional neural networks***

Quiz

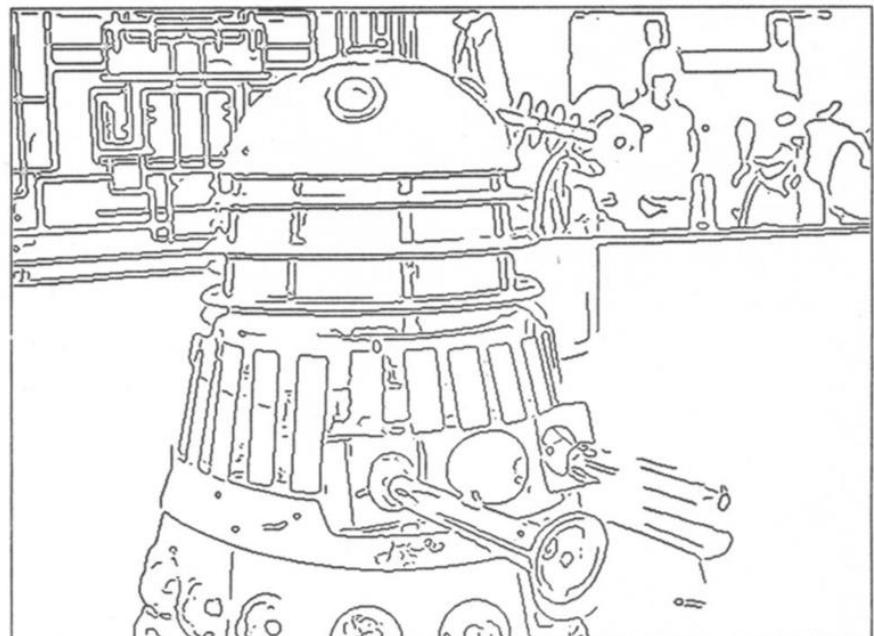
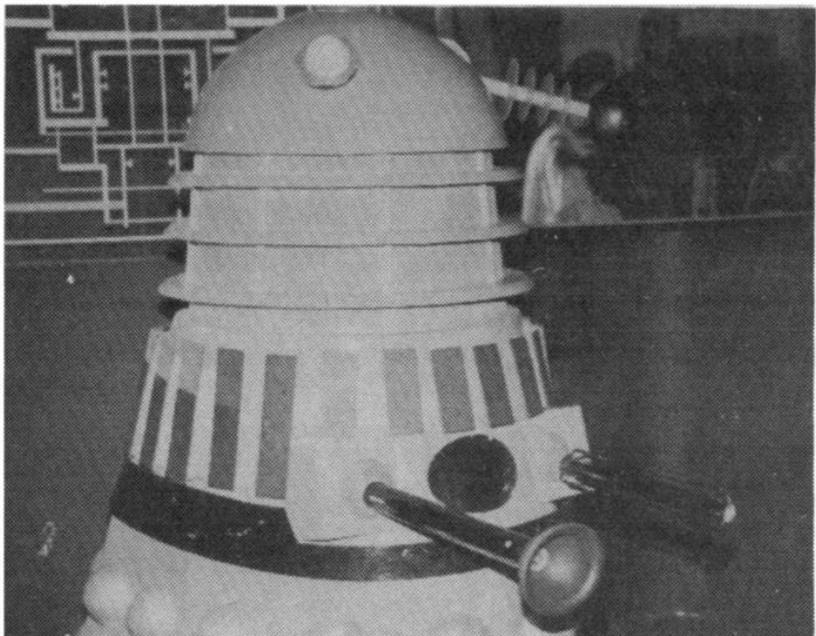
What features could you use to separately cluster dogs and cats?



Feature Identification

Significant progress has been made for image processing in the field of computer vision

Algorithmic edge detection (e.g. intensity gradients)

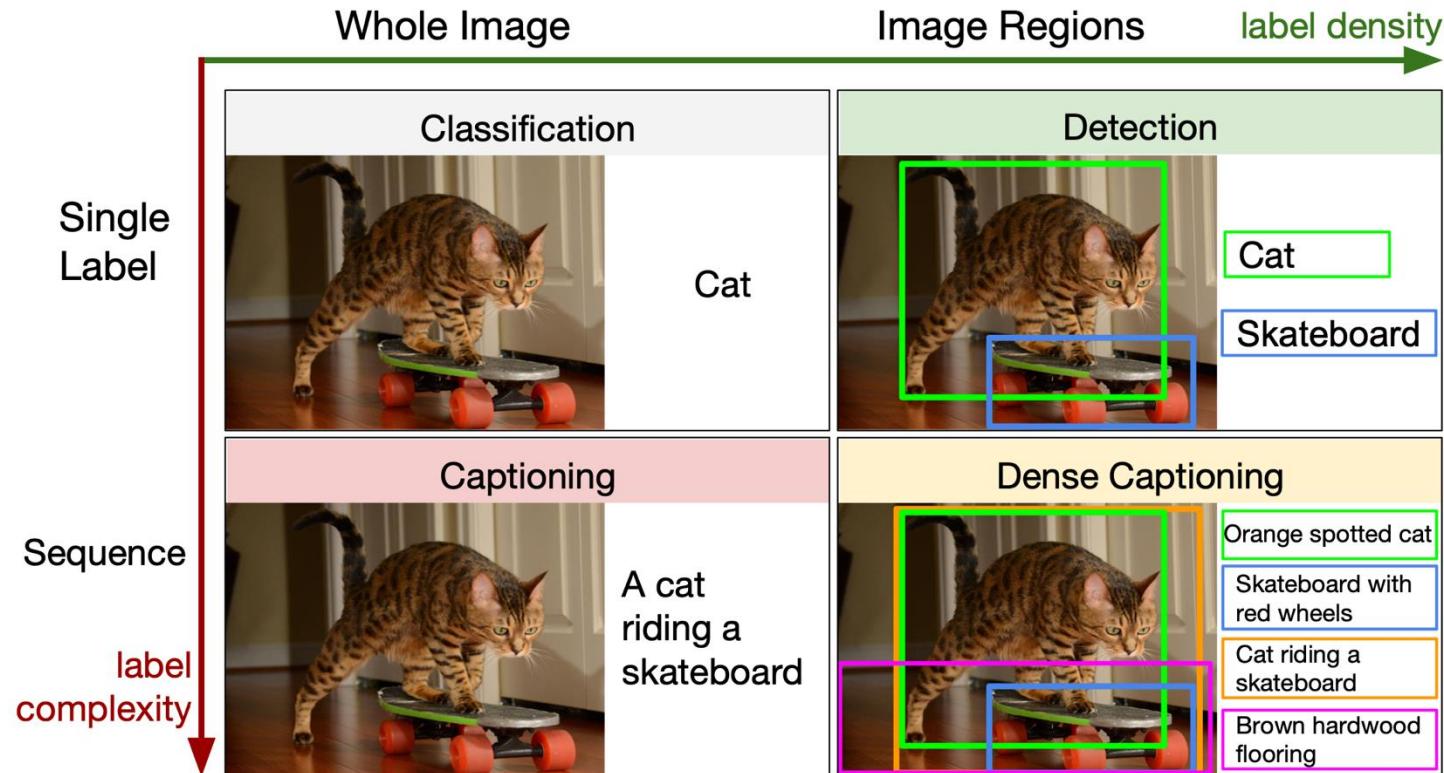


Popular packages for classical filters include `imagej` and `scikit-image`

Feature Identification

Significant progress has been made for image processing in the field of computer vision

Deep learning image and language model (e.g. DenseCap)



Images as Arrays

Pixels are a convenient representation, but inefficient, e.g. 1 MP image = 1,000,000 pixels

Image



Array of pixels

\sqrt{d}



\sqrt{d}

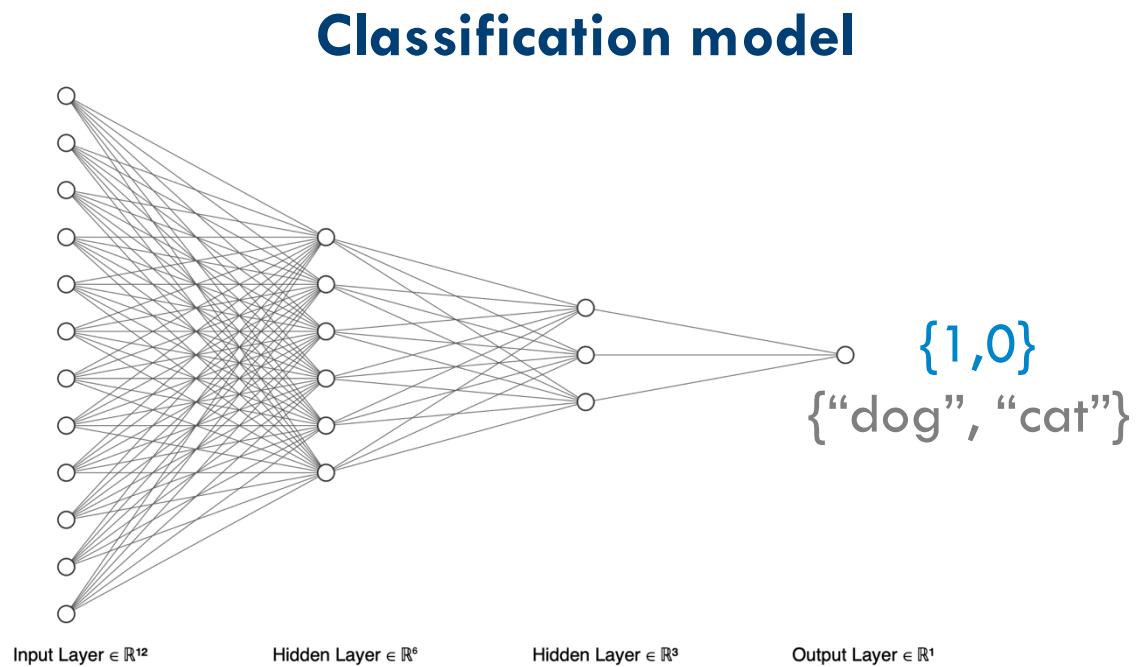
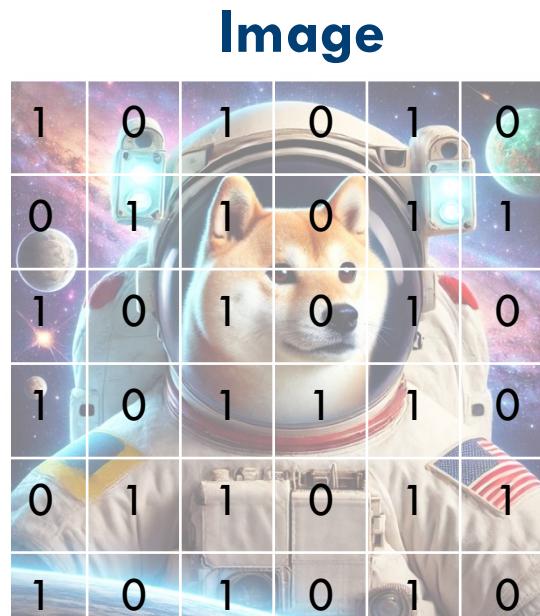
Feature vector



Decision boundaries are difficult to define, e.g.
to distinguish between animals based on pixels alone

Images as Arrays

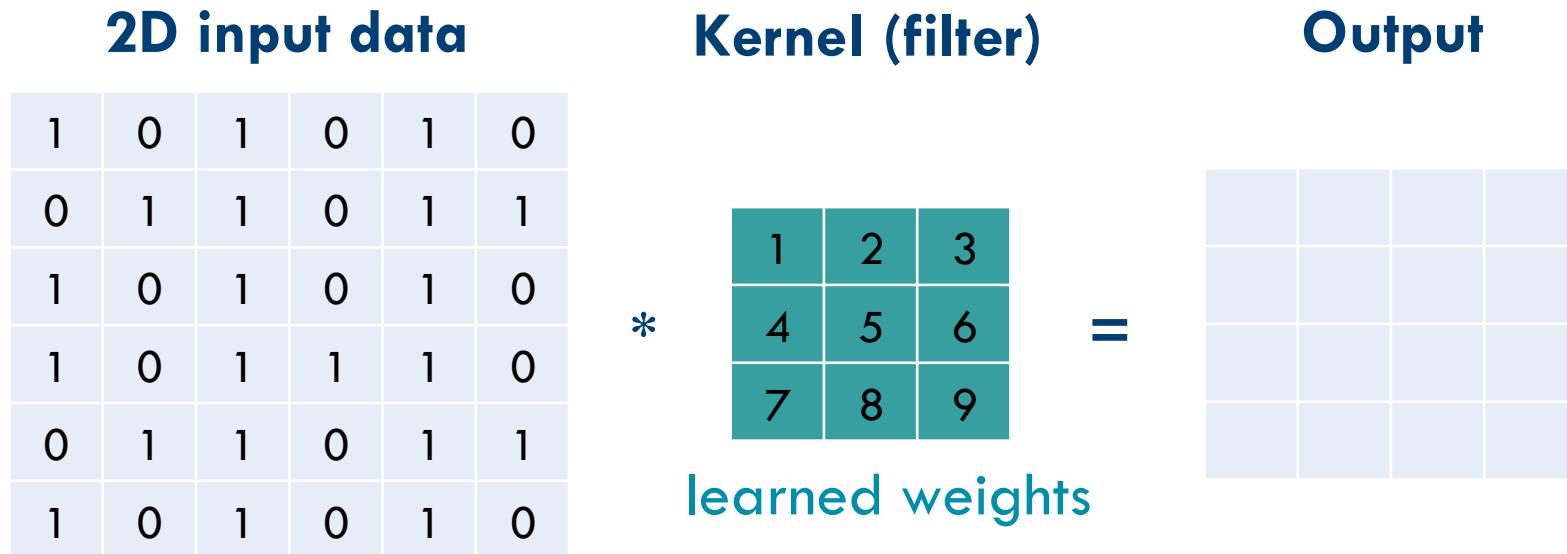
Each layer in a deep neural network can improve the representation of the preceding layer



The initial sparse input is densified as it passes through each layer of the network

Convolutional Filters

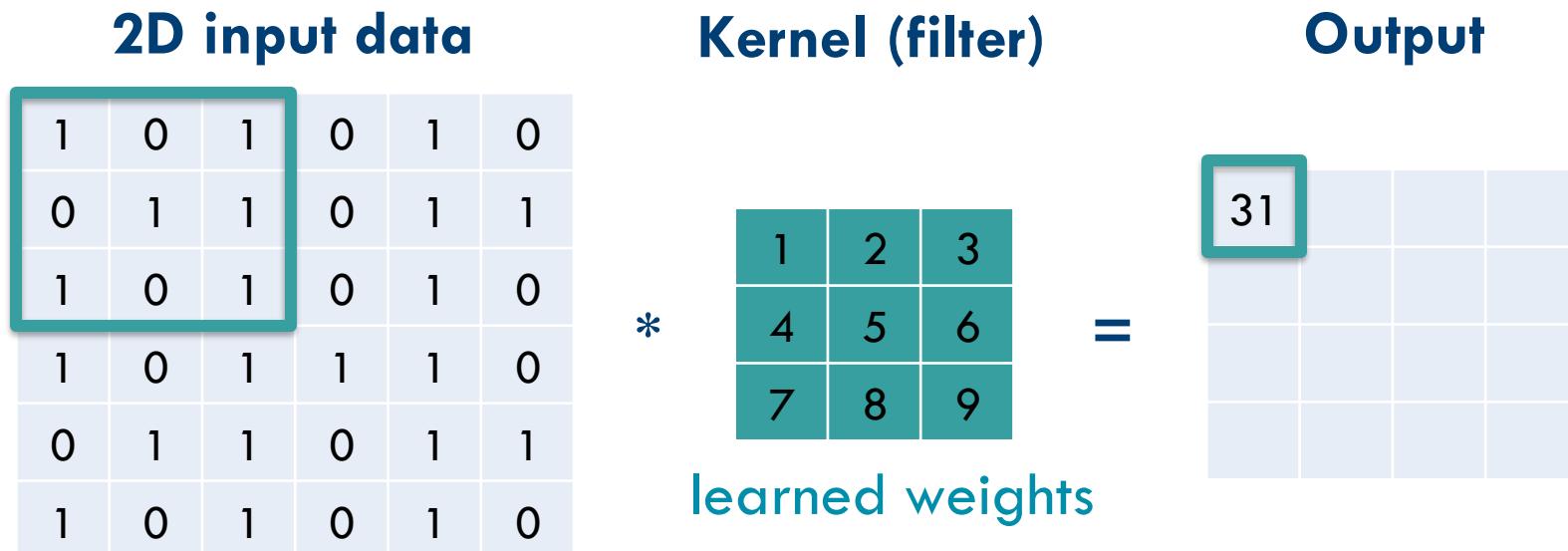
Small matrices (kernels) that extract features from data by performing localised operations



Kernel passes over the input data, capturing patterns at different locations, enabling the network to learn and detect specific features

Convolutional Filters

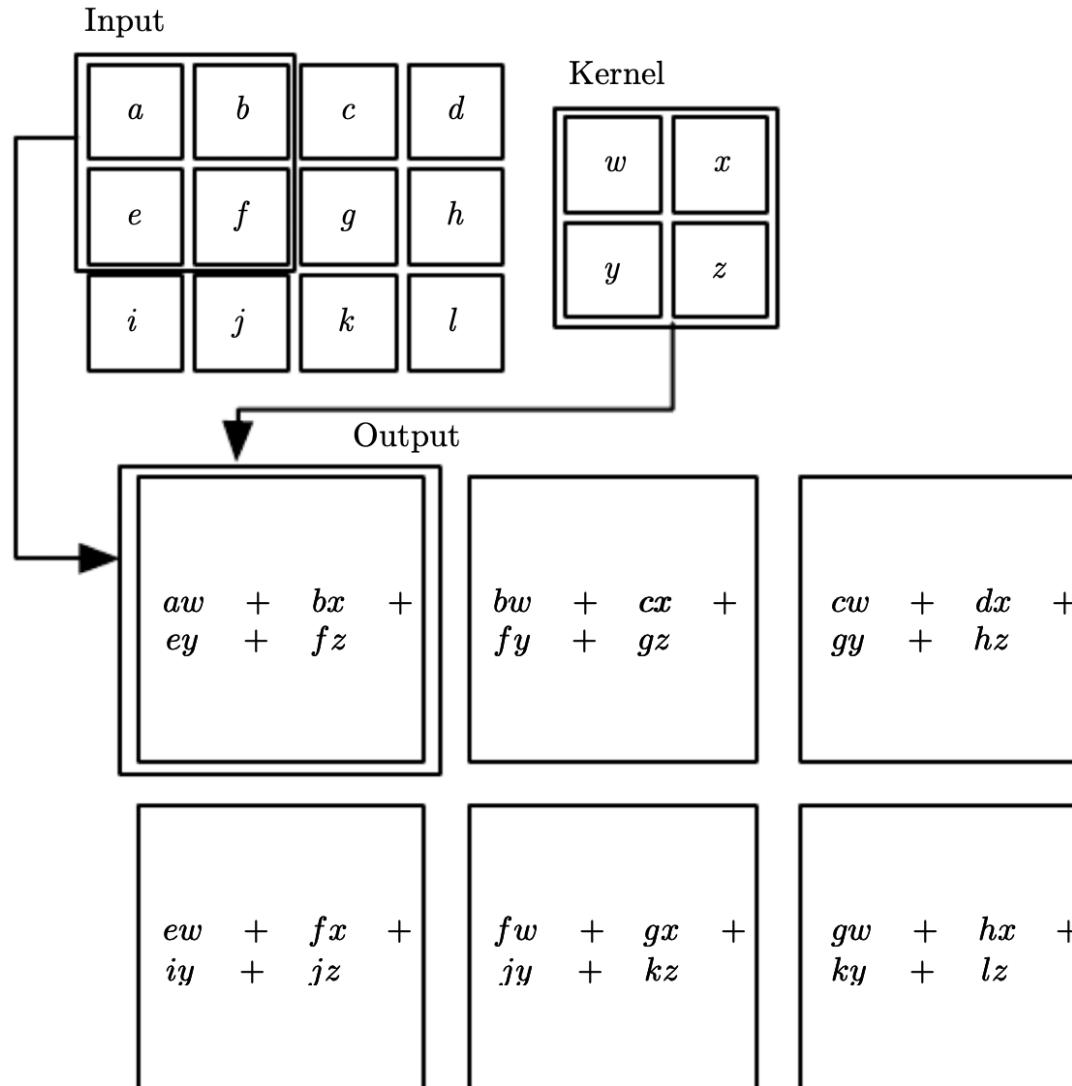
Small matrices (kernels) that extract features from data by performing localised operations



Sum of element-wise products:

$$1*1+0*2+1*3+0*4+1*5+1*6+1*7+0*8+1*9 = 31$$

Convolutional Filters



Quiz

What would these kernels do to an image?

Kernel A

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Kernel B

-1	-1	-1
2	2	2
-1	-1	-1

Kernel C

-1	-1	-1
-1	8	-1
-1	-1	-1

Original Image



An image of the proposed room-temperature superconductor LK-99

Quiz

What would these kernels do to an image?

Kernel A

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

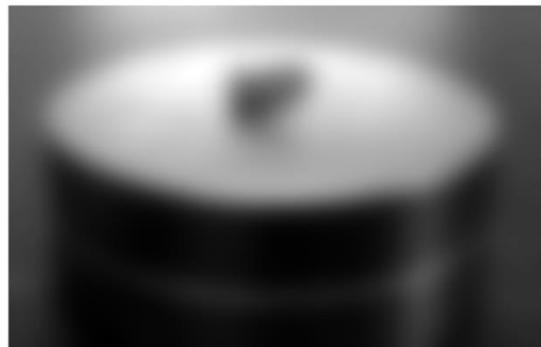
Kernel B

-1	-1	-1
2	2	2
-1	-1	-1

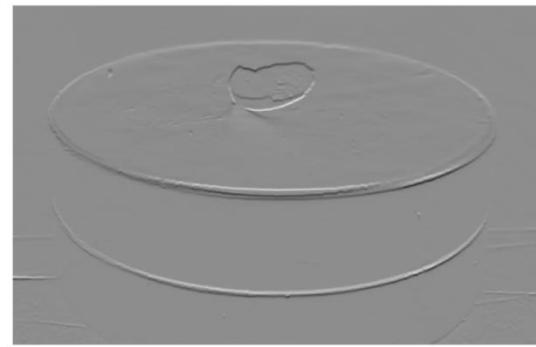
Kernel C

-1	-1	-1
-1	8	-1
-1	-1	-1

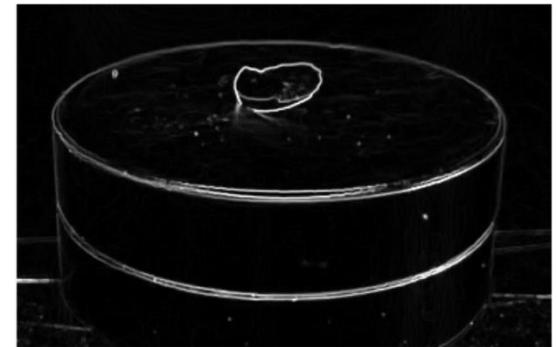
Blur



Horizontal lines



Edge detection



Convolutional Neural Networks (CNN)

A type of neural network used for processing data with a grid-like topology (images, time series...)

80322-429 80206

40004 14310

37878 05153

~~35502~~ 75216

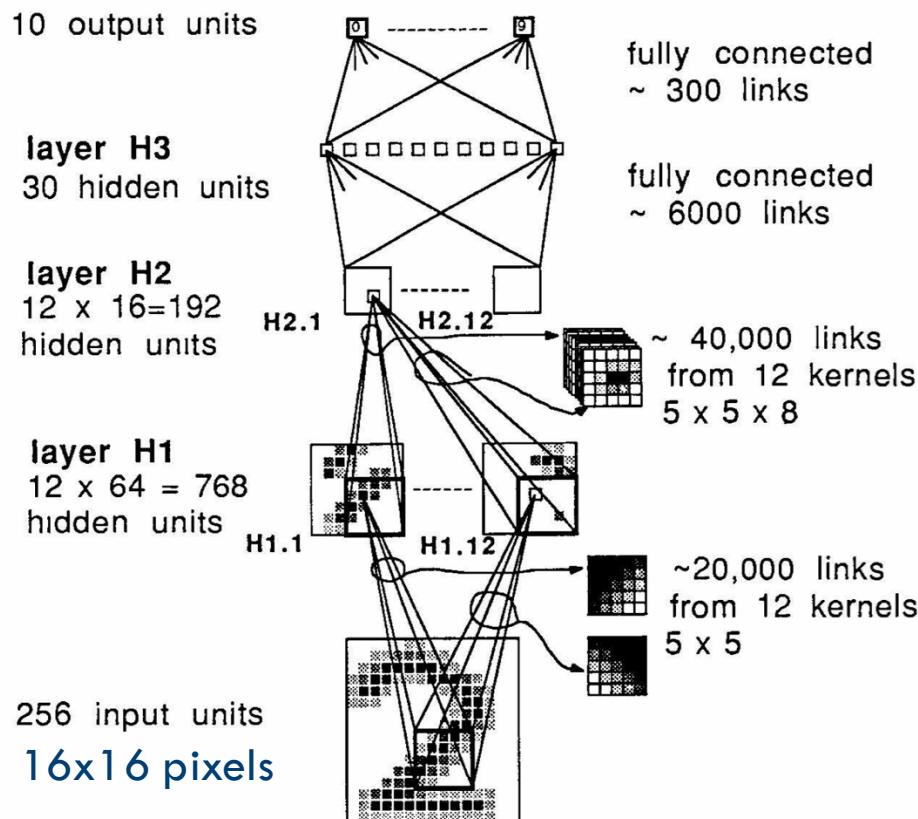
35460 44209

US Postal Service Challenge

Computer recognition of handwritten zip codes

Convolutional Neural Networks (CNN)

A type of neural network used for processing data with a grid-like topology (images, time series...)



Output

$\{0,1,2,3,4,5,6,7,8,9\}$

Model

1000 neurons

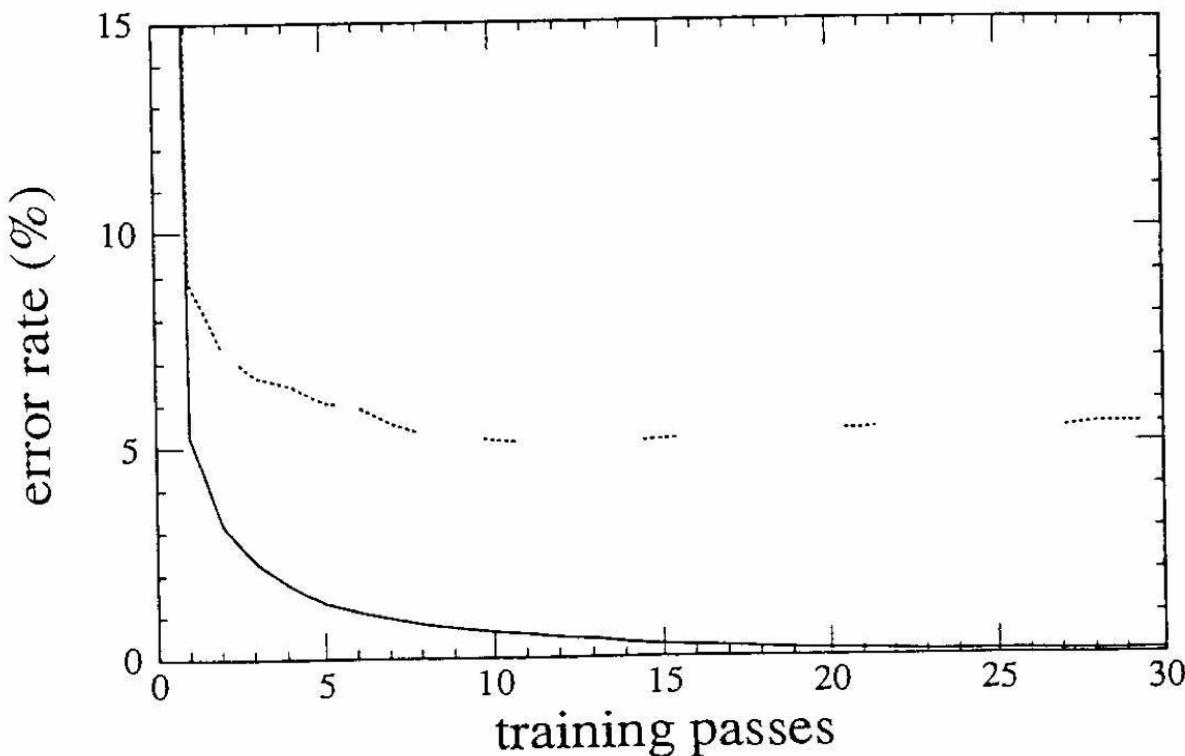
9760 parameters

Input

Direct images rather than
feature vectors

Convolutional Neural Networks (CNN)

A type of neural network used for processing data with a grid-like topology (images, time series...)



Training

7291 examples

Testing

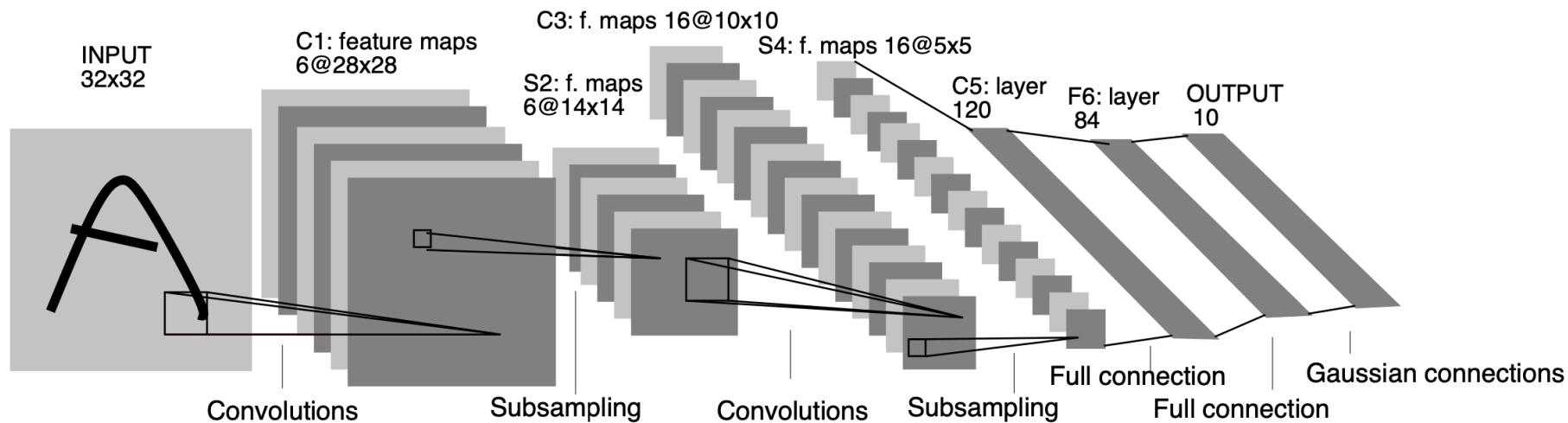
2007 examples

(included hand-chosen
ambiguous or
unclassifiable samples)

Convolutional Neural Networks (CNN)

LeNet-5 was the fifth evolution of this network
and became a standard in the field

Higher resolution input
(MNIST dataset)



C_n = convolutional layer n (extract features)

S_n = sub-sampling layer n (reduce spatial dimensions)



```
import torch.nn as nn

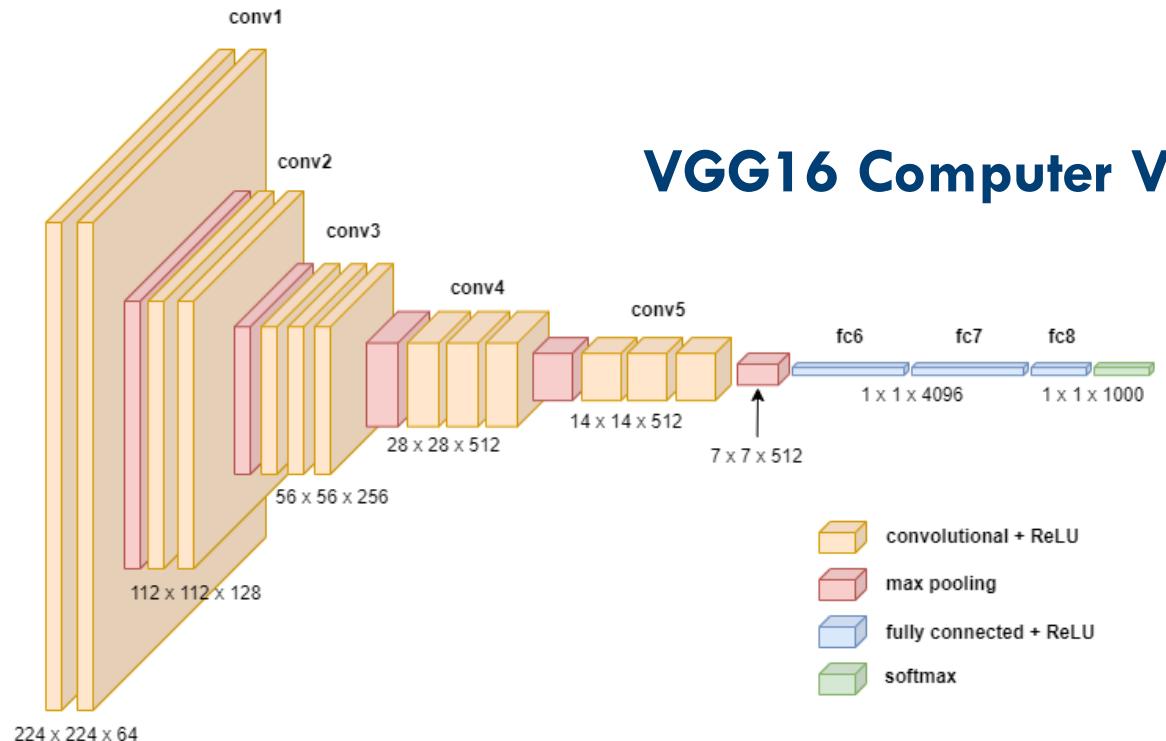
class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5)
        self.conv2 = nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5)
        self.fc1 = nn.Linear(in_features=16*5*5, out_features=120)
        self.fc2 = nn.Linear(in_features=120, out_features=84)
        self.fc3 = nn.Linear(in_features=84, out_features=10) # Assuming 10 classes

    def forward(self, x):
        x = nn.functional.relu(self.conv1(x))
        x = nn.functional.max_pool2d(x, kernel_size=2)
        x = nn.functional.relu(self.conv2(x))
        x = nn.functional.max_pool2d(x, kernel_size=2)
        x = x.view(-1, 16*5*5)
        x = nn.functional.relu(self.fc1(x))
        x = nn.functional.relu(self.fc2(x))
        x = self.fc3(x)
        return x

# Create an instance of the LeNet-5 model
lenet5_model = LeNet5()
```

Towards State of the Art (SOTA)

Modern deep learning models combine many layer types with 10^3 - 10^{12} parameters



VGG16 Computer Vision Model

Softmax is an activation function common in the output layer of a neural network for classification tasks

Towards State of the Art (SOTA)

Modern deep learning models combine many layer types with 10^3 - 10^{12} parameters

Input vector

$$\begin{bmatrix} 3 \\ 6 \\ 7 \\ 11 \\ 4 \end{bmatrix}$$

Softmax

$$\sigma(z_i) = \frac{e^{z_i/T}}{\sum_{j=1}^n e^{z_j/T}}$$

Partition function

Class probability

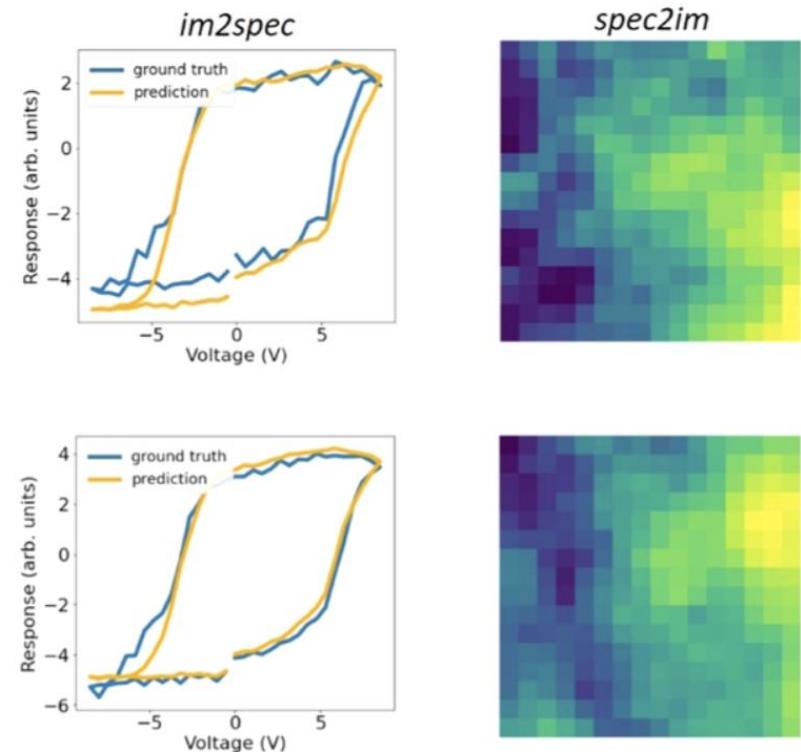
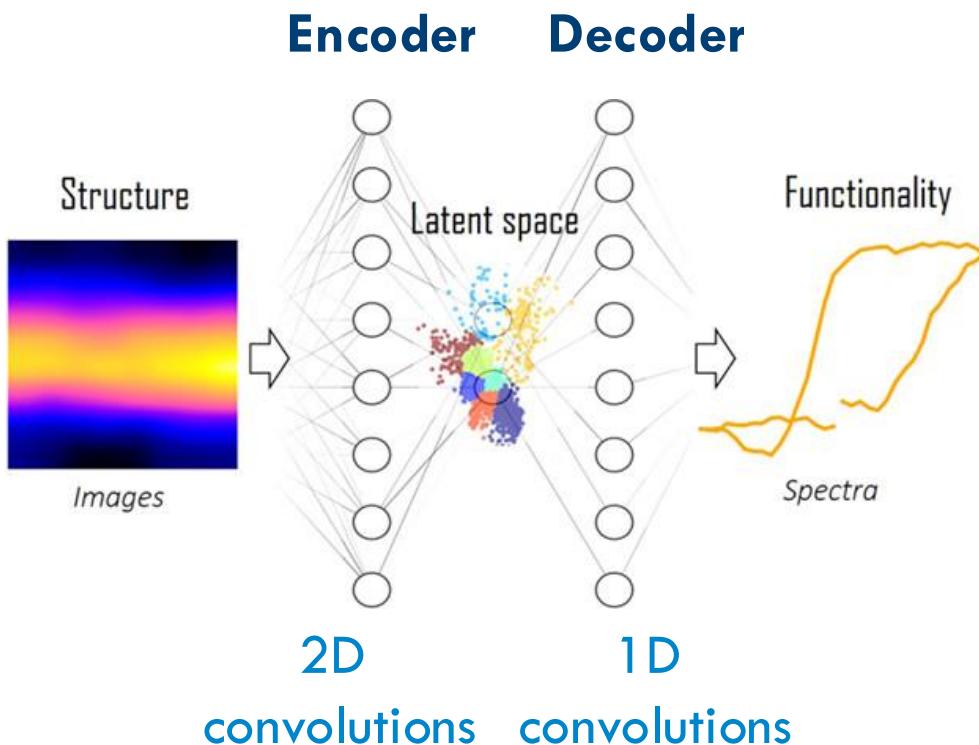
$$\begin{bmatrix} 0.00 \\ 0.03 \\ 0.04 \\ 0.92 \\ 0.01 \end{bmatrix}$$

Appearance of the Boltzmann distribution

(deep learning models often borrow from statistical mechanics)

CNN for Ferroelectrics

Convolutional network to map between local domain structures and switching behaviour



Application to Materials

Graph Convolutional Neural Networks (**GCNN**)
are designed for graph-structured data

Pixel-based
convolutions

1	4	5
7	3	6
6	1	7

6

Graph-based
convolutions

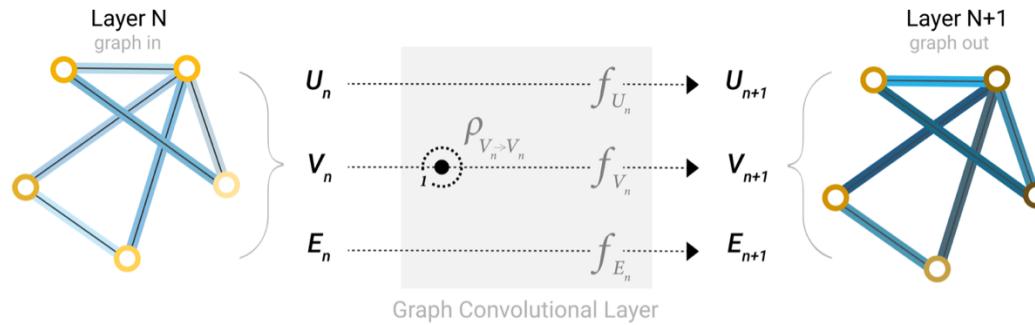


Information is stored on each piece of the graph,
i.e. vectors associated with the nodes, edges and global attributes

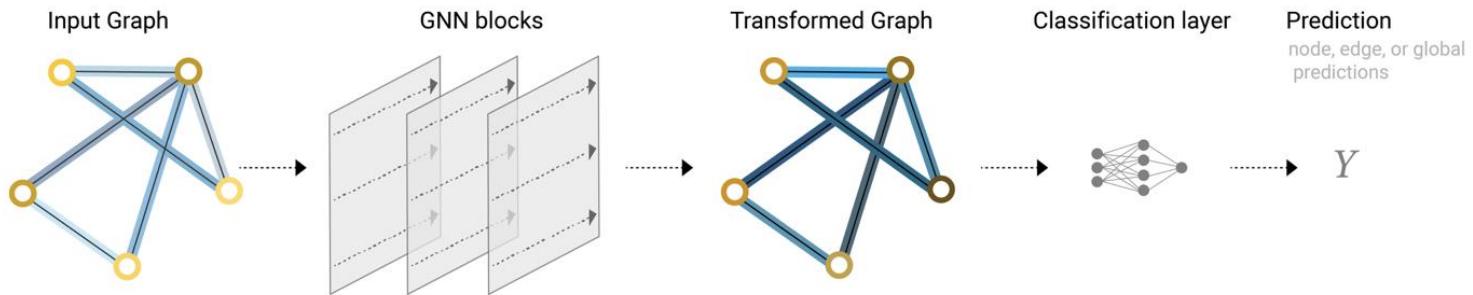
Application to Materials

Input graphs can be transformed and tailored
for regression or classification tasks
(V = vertex; E = edge, U = global attribute)

Graph update

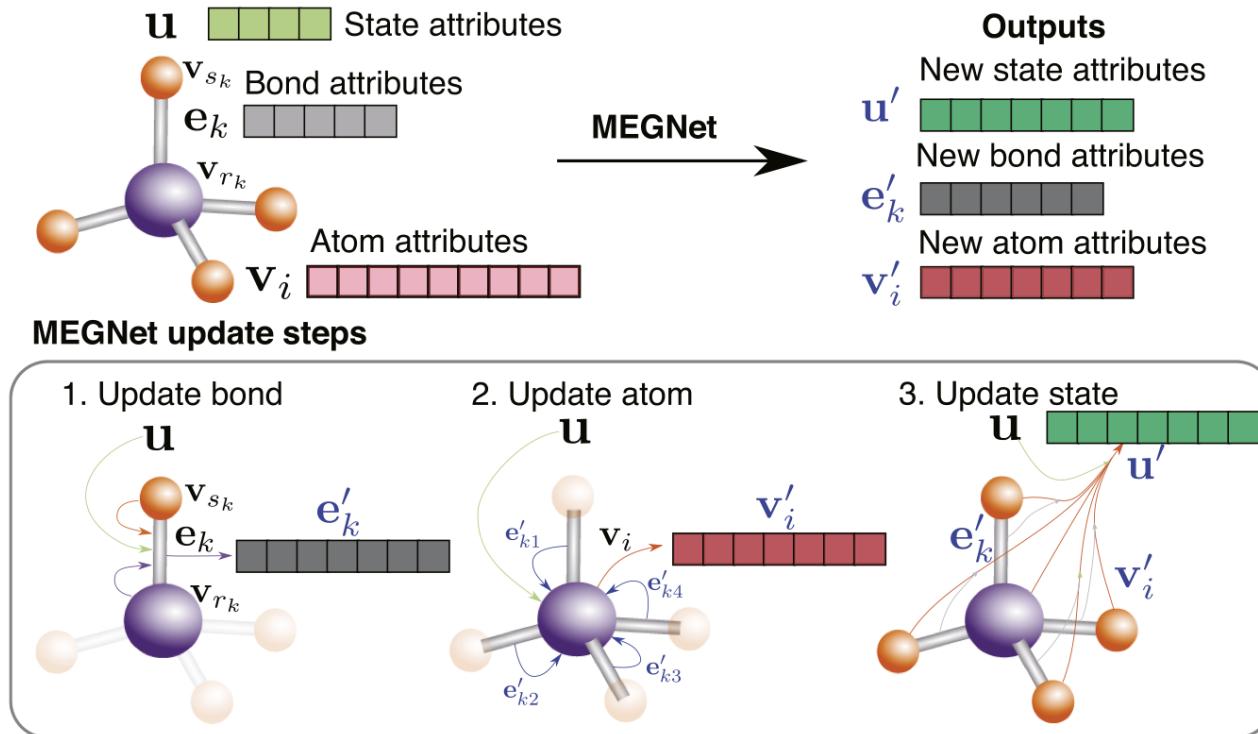


End-to-end prediction



Application to Materials

Crystal Graph Convolutional Neural Networks (CGCNN) are being used for materials modelling

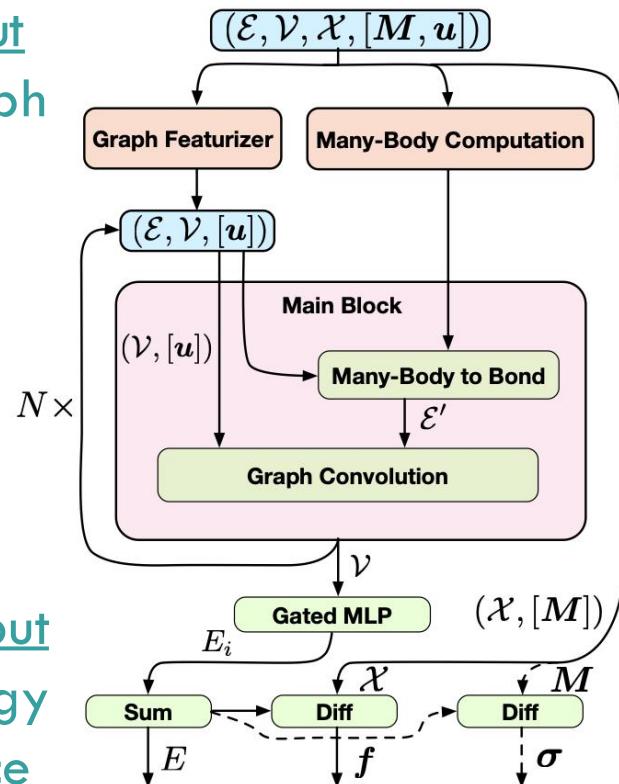


MEGNet (two-body connections); <https://github.com/materialsvirtuallab/matgl>

Application to Materials

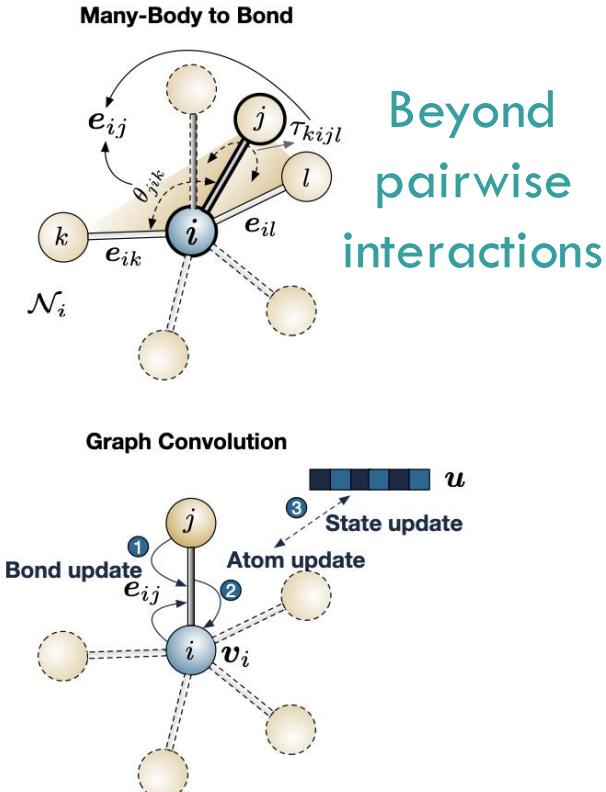
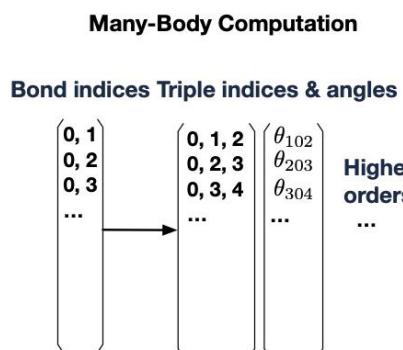
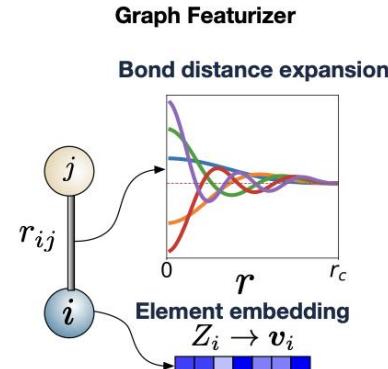
A new generation of universal force fields that can
predict energies and forces for any material

Input
Graph



Output
Energy
Force
Stress

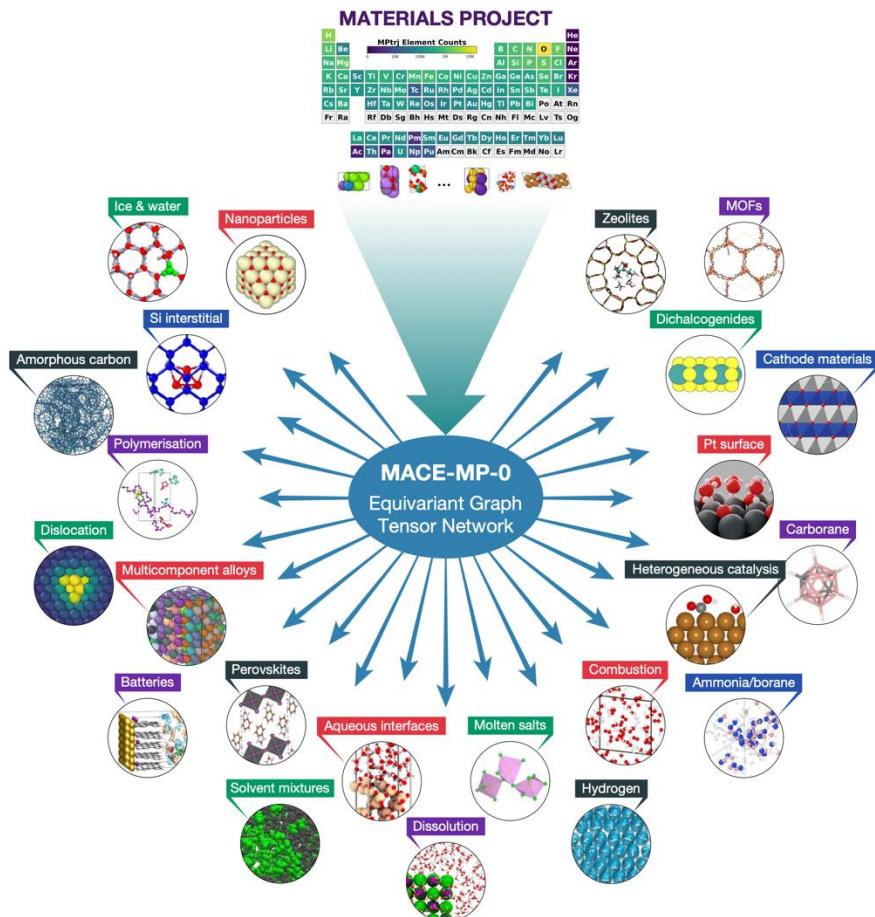
M3GNet; <https://github.com/materialsvirtuallab/matgl>



Beyond
pairwise
interactions

Application to Materials

A new generation of universal force fields that can
predict energies and forces for any material



```
from ase.io import read, write
from mace.calculators import mace_mp

calc = mace_mp(model="medium")

# Initial energy and structure parameters
material = read('./data/Sn5S4Cl2.cif')
material.calc=calc
print(material.get_potential_energy())

print("Initial Energy:", material.get_potential_energy())

print("Initial Cell Parameters:")
print(material.cell)
```

Class Outcomes

1. Describe how a perceptron works
2. Distinguish between different types of deep learning architectures
3. Specify how convolutional neural networks work and can be applied to materials problems

Activity:

Learning microstructure