Introduction to Neural Networks – Part 2

Adam Carter, EPCC, The University of Edinburgh
a.carter@epcc.ed.ac.uk

August 2025

www.archer2.ac.uk

Reusing this material

Partners

epcc

UKRI

Engineering and Physical Sciences Research Council

Natural Environment Research Council

epcc

THE UNIVERSITY of EDINBURGH

Hewlett Packard Enterprise

Convolutional Neural Networks

Convolutional Neural Networks - CNNs

- Specific type of Neural Networks (NNs)
- The popularity & success of Deep Learning owes much to CNNs
  - which have, in turn, benefitted from computational improvements allowing networks of ever-increasing depth
- CNNs can learn models which recognize objects in images and perform image classification
  - Learn directly from the images
- Input layer is large: one input for each pixel in the image
- Neurons in one layer not fully connected to all neurons in next layer
- Final output: Reduced to a single vector of probability scores
- Often illustrated and modelled in “3D”
  - Matrices of neurons in a layer rather than vectors
- CNNs have 2 key components:
  - Hidden layers – feature extraction part, where network performs convolution + pooling operations to detect features (e.g. stripes of a zebra, edges)
  - Classification: Fully connected layers serve as classifiers of these extracted features – assign probability

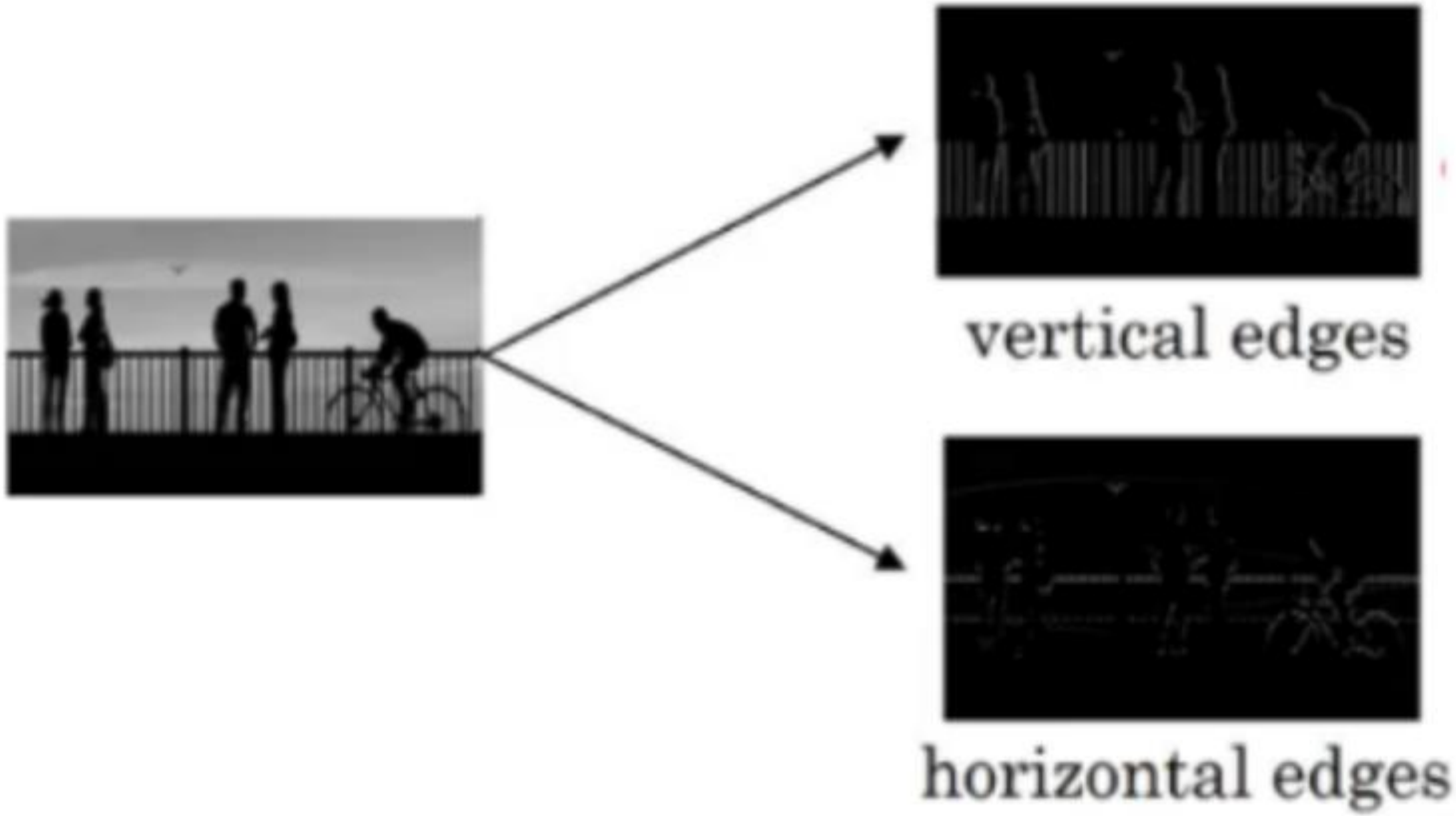
- CNNs are “just” deep neural networks with:
    - Particular kinds of layers which have
        - A particular kind of *connectivity* to previous layers, and
        - A particular kind of *activation function* which is designed to help identify features in images
- The two most important kinds of layers are
    - **convolution layers** which put the “C” in CNN, and
    - **pooling layers**

Convolution

- Using different **filters/kernels** on the input data to produce a **feature/activation map**
- Each filter corresponds to a receptive field
- After performing several convolutions (can be 100s of hidden layers), each with a different filter, we obtain a map of all features – the final output
- Training of CNNs:
  - Same as for regular NNs using backprop and gradient descent
  - Mathematically more complex due to convolutional operations

CNN example – Edge Detection

Start by taking a 6x6 grayscale image (one **channel**)



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

From https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/
(which in turn may have been taken from Andrew Ng's videos…)

Edge Detection

"filter" or "kernel"

$$\begin{bmatrix}6&3&0&2&0&1\\3&1&6&6&7&2\\6&3&8&4&5&9\\0&1&3&1&6&5\\0&6&0&0&9&4\\7&8&2&4&5&7\end{bmatrix} * \begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix} = \begin{bmatrix}\ &\ &\ &\ \\\ &\ &\ &\ \\\ &\ &\ &\ \\\ &\ &\ &\ \end{bmatrix}$$

convolution

Edge Detection

$$\begin{bmatrix} 6^{1} & 3^{0} & 0^{-1} & 2 & 0 & 1 \\ 3^{1} & 1^{0} & 6^{-1} & 6 & 7 & 2 \\ 6^{1} & 3^{0} & 8^{-1} & 4 & 5 & 9 \\ 0 & 1 & 3 & 1 & 6 & 5 \\ 0 & 6 & 0 & 0 & 9 & 4 \\ 7 & 8 & 2 & 4 & 5 & 7 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} \boxed{1} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

$$6 \times 1 + 3 \times 0 + 0 \times -1 + 3 \times 1 + 1 \times 0 + 6 \times -1 + 6 \times 1 + 3 \times 0 + 8 \times -1$$
$$= 6 + 0 + 0 + 3 + 0 + -6 + 6 + 0 + -8 = 1$$

Edge Detection

$$\overset{6\times6}{\begin{bmatrix} 6 & 3 & 0 & 2 & 0 & 1 \\ 3 & 1 & 6 & 6 & 7 & 2 \\ 6 & 3 & 8 & 4 & 5 & 9 \\ 0 & 1 & 3 & 1 & 6 & 5 \\ 0 & 6 & 0 & 0 & 9 & 4 \\ 7 & 8 & 2 & 4 & 5 & 7 \end{bmatrix}} * \overset{3\times3}{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}} = \overset{4\times4}{\begin{bmatrix} 1 & \boxed{-5} & & \\ & & & \\ & & & \\ & & & \end{bmatrix}}$$

$$3\times1 + 0\times0 + 2\times-1 + 1\times1 + 6\times0 + 6\times-1 + 3\times1 + 8\times0 + 4\times-1$$

$$= 3 + 0 - 2 + 1 + 0 - 6 + 3 + 0 - 4 = -5$$

Convolving a 6x6 image with a 3x3 filter

|epcc|

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

6 X 6 image

*

1	0	-1
1	0	-1
1	0	-1

3 X 3 filter

for vertical edges

=

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

- Dot product for first output element:
  - $3*1 + 0*0 + 1*(-1) + 1*1 + 5*0 + 8*(-1) + 2*1 + 7*0 + 2*(-1) = -5$
- Type of filter helps to detect horizontal or vertical edges
- Higher-magnitude values in output matrix will represent areas where edges have been located

$$\underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{vertical edge detection}} \qquad \underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}}_{\text{horizontal edge detection}}$$

epcc

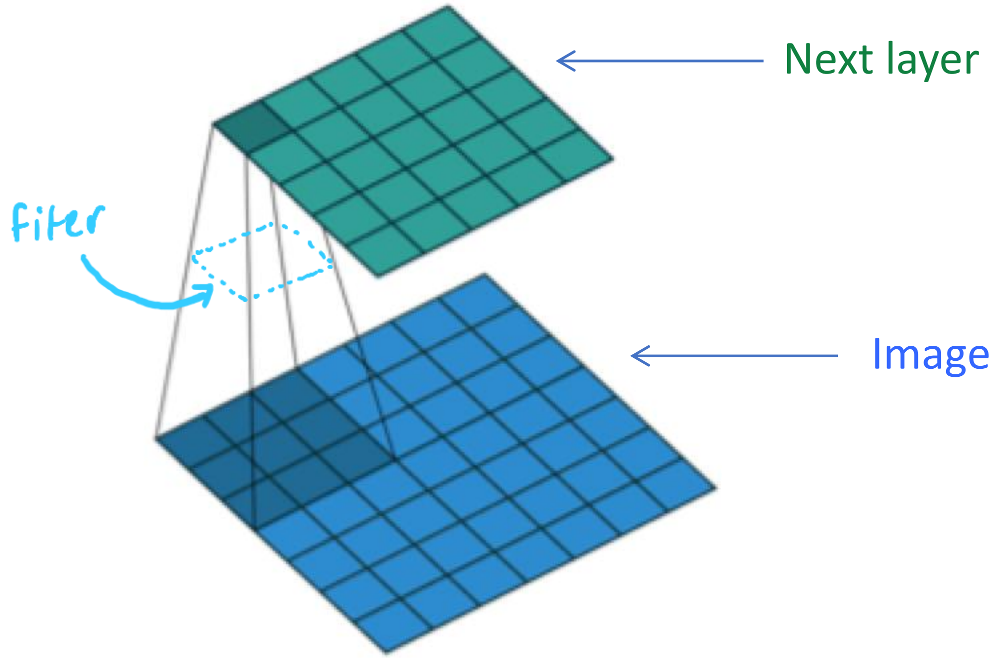
$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

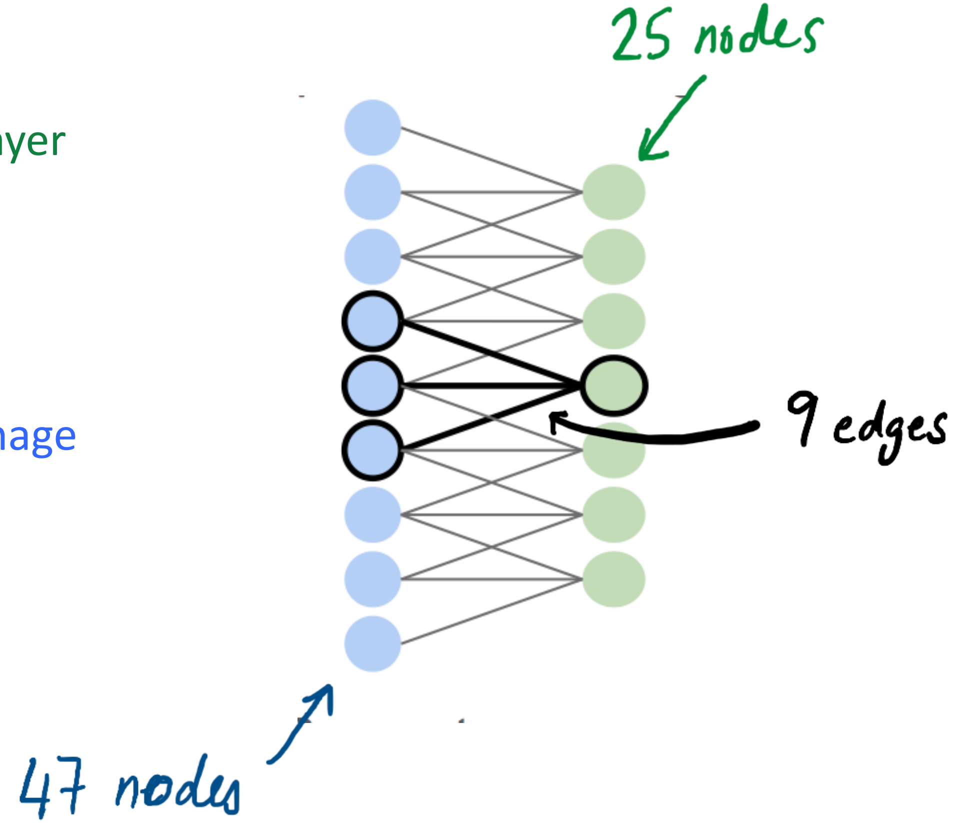
but in a CNN we let the NN find useful filters itself

$$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}$$

Convolution

- CNNs slide the same kernel of weights across their input (**compute dot product**), thus have local sparse connectivity + tied weights

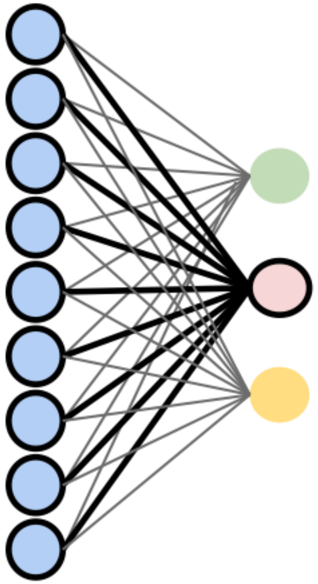




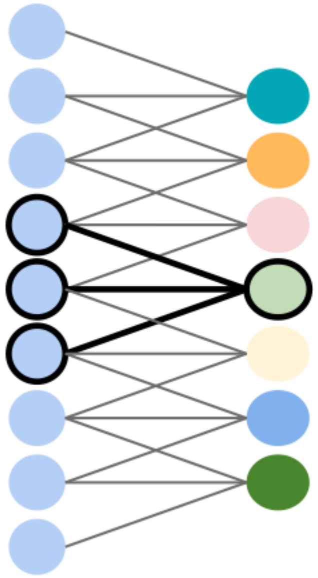
Connectivity

Fully connected (dense):

Every neuron is connected to all components of the input vector

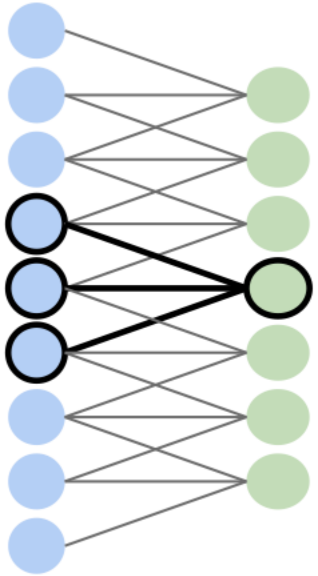


Sparse connectivity



Every neuron is only affected by a limited input “receptive field”; 3 in this example.

Sparse connectivity + parameter sharing



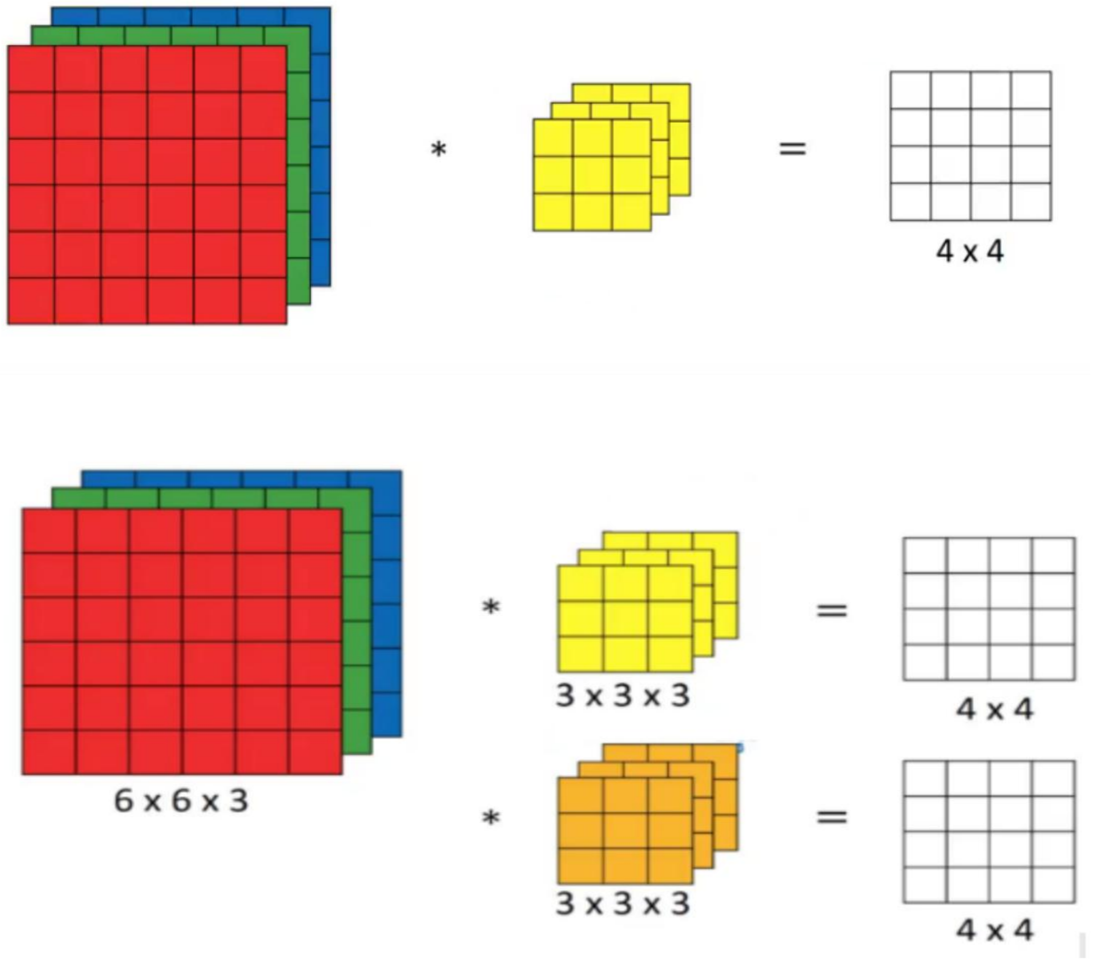
Parameters are shared (tied weights) across all neurons

From: M, Mustafa, Introduction to Deep Learning, NERSC

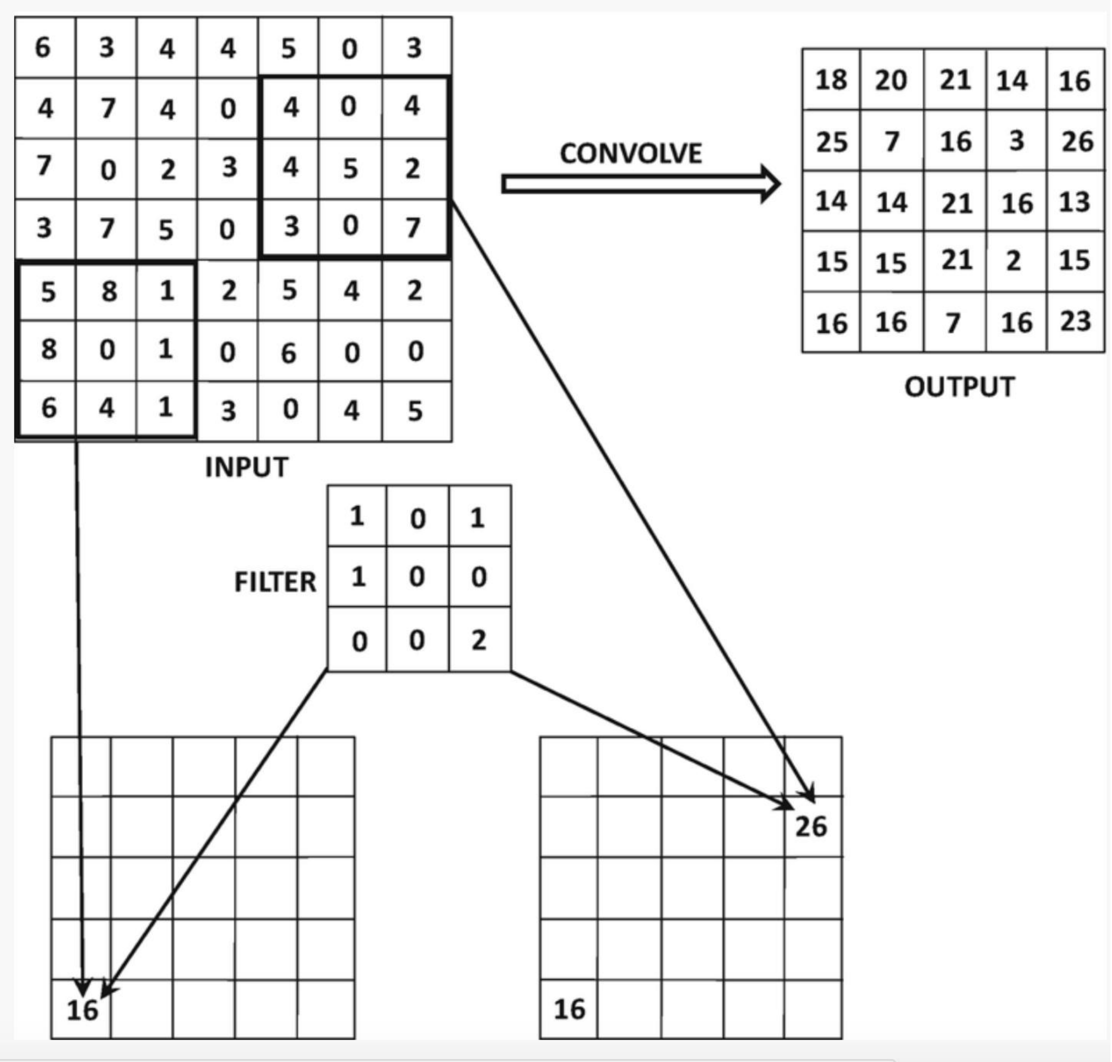
Shared weights + Biases

- Weights + biases are usually all the same for all neurons in a given layer
- So all hidden neurons **detect the same hidden feature** (e.g. edge) in different regions of the image
- This makes the network tolerant re translations of objects in an image – it doesn’t matter where the object is in the image

CNNs with multiple channels



From https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/



From https://link.springer.com/chapter/10.1007/978-3-319-94463-0_8

Padding

- From https://link.springer.com/chapter/10.1007/978-3-319-94463-0_8

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

PAD $\Longrightarrow$

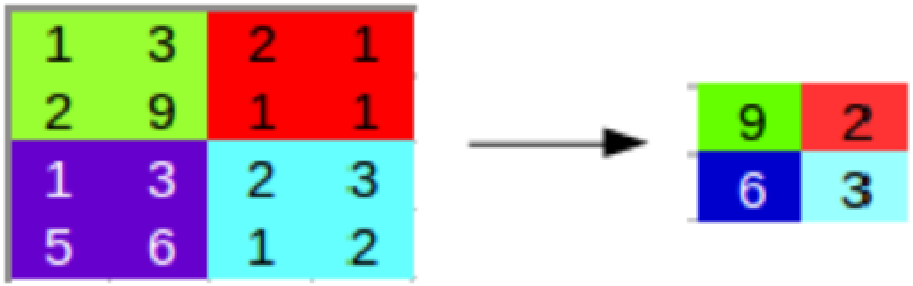
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	6	3	4	4	5	0	3	0	0
0	0	4	7	4	0	4	0	4	0	0
0	0	7	0	2	3	4	5	2	0	0
0	0	3	7	5	0	3	0	7	0	0
0	0	5	8	1	2	5	4	2	0	0
0	0	8	0	1	0	6	0	0	0	0
0	0	6	4	1	3	0	4	5	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Pooling

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

Input

Applying **‘max’ pooling** with a filter of stride = 2, size = 2
(Pooling layer hyperparameters)

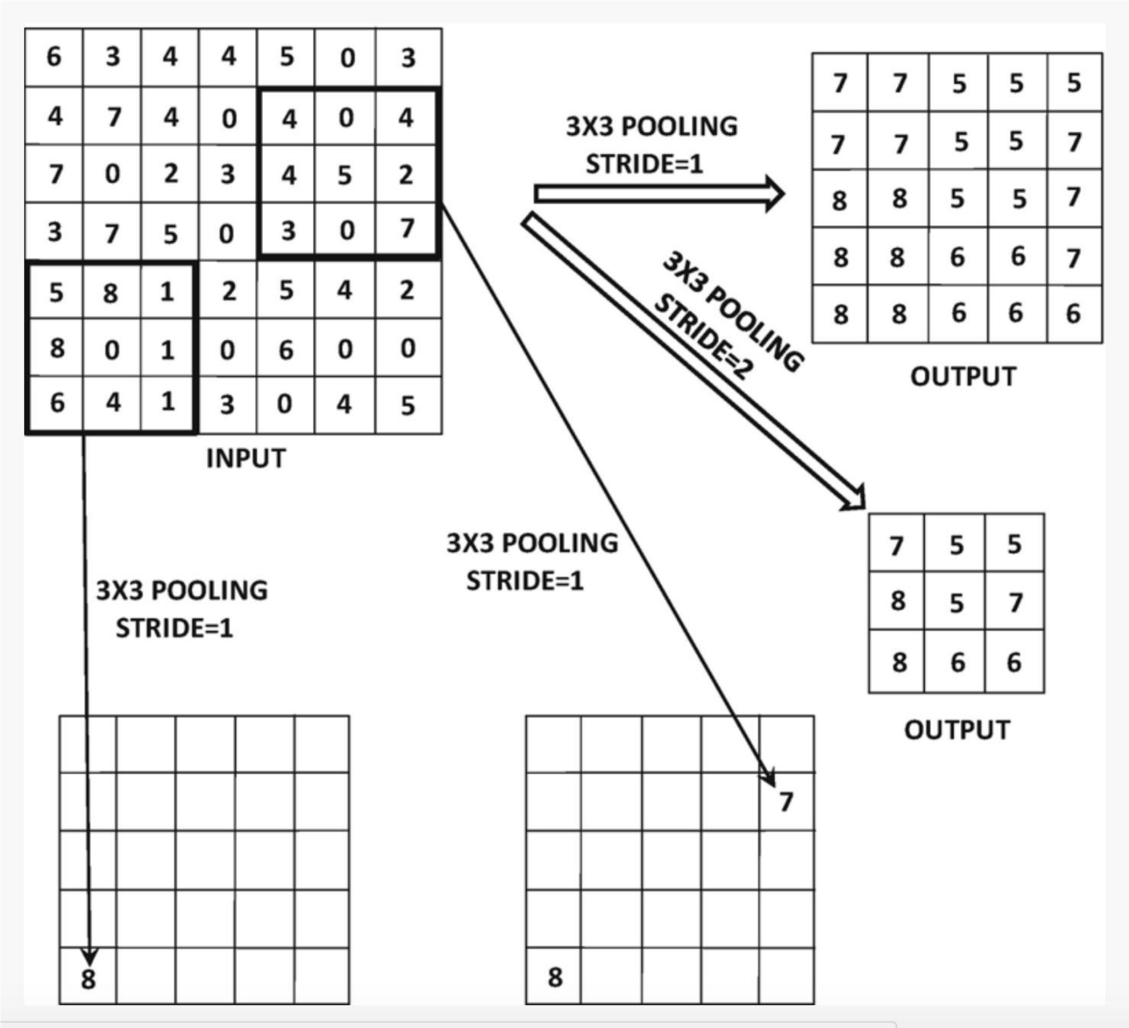


Strides in Padding & Pooling

- Stride: Size of the step the convolution filter moves each time. Usually (and in example on previous slides) slides pixel by pixel. Stride > 2 is uncommon, bad on accuracy if too large.
- Padding: Size of feature map is always smaller than input. Corner pixels are covered less often than centre pixels. To prevent it from shrinking: add layer of zero-valued pixels.
- Pooling:
  - Makes the model invariant to small local translations of input
  - ‘Max’ (introduces non-linearity + greater translational invariance) or ‘Average’ pooling are most common pooling layers
  - Used to reduce dimension of input feature map (condenses output of small regions) , reduces #parameters that the model needs to learn => speeds up training time; stride > 1
  - Controls overfitting
  - Interleafed with convolution/ReLu layers, but less frequently applied

Max Pooling example

- From https://link.springer.com/chapter/10.1007/978-3-319-94463-0_8

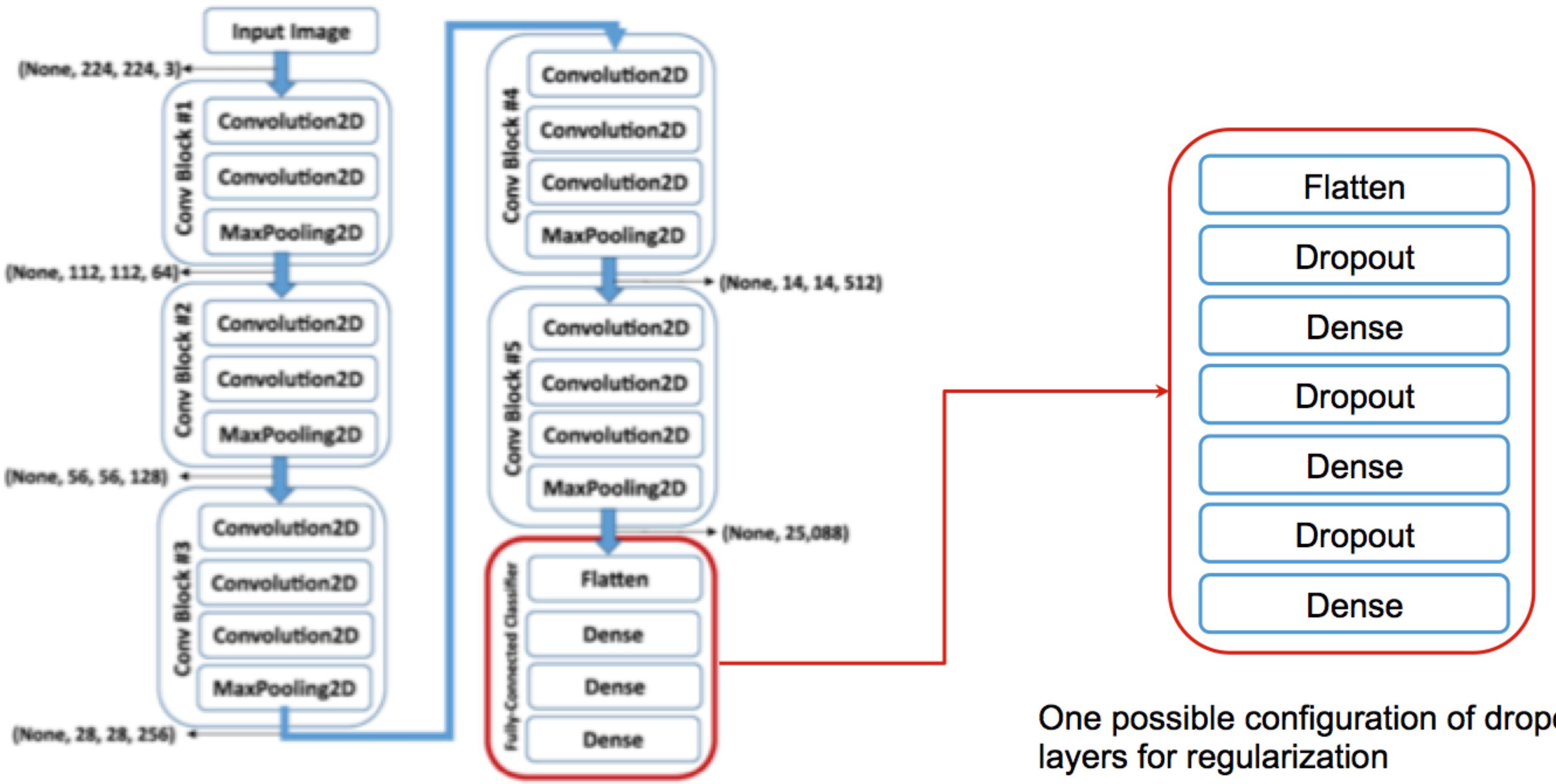


Classification – fully connected layer

- After convolution + pooling layers
- A few fully connected layers (like regular NN)
- Can only accept 1D data
  - Conversion from 3D -> 1D (e.g. 'flatten' in Python)
- Where most parameters are
  - E.g. Two fully connected layers with 4096 hidden units have $4096^2 \sim 16$ million weights

(Convolutional layers have more activations)

Example for a full CNN architecture
…pulling all the bits together from the previous slides



One possible configuration of dropout layers for regularization

A schematic of VGG-16 Deep Convolutional Neural Network (DCNN) architecture

trained on ImageNet (2014 ILSVRC winner)

Dropout

- Dropping out units/neurons during the training phase =>
  - These neurons are not considered during a particular forward or backward propagation
- Chosen at random
- To avoid overfitting
- Used between fully connected layers of the classification part to improve the generalization error

Residual Networks and Depth

- Increasing network depth: (see Goodfellow et al. arXiv:1312.6082)
  - Accuracy increases
  - Outperform wider models with same number of parameters
  - Harder to train due to the vanishing gradient problem
  - (exploding gradients are also possible!)

- As gradient is back-propagated to earlier layers:
  - Repeated multiplication can make gradient very small
  - Solution: Skip Connections

Skip Connections

- Partial architecture of **ResNet**
  - Connections between layers i and (i+r), r>1
- Copy features from earlier layers i into later ones (i+r)
- Inputs classified with a small amount of non-linearity (simple ones) will skip many connections
- Inputs with more complex structures will travel across more connections so all relevant features can be covered
- Side-steps the vanishing gradient problem
  - Effective gradient flow because the backprop algorithm can now use a fast lane using the skip connections

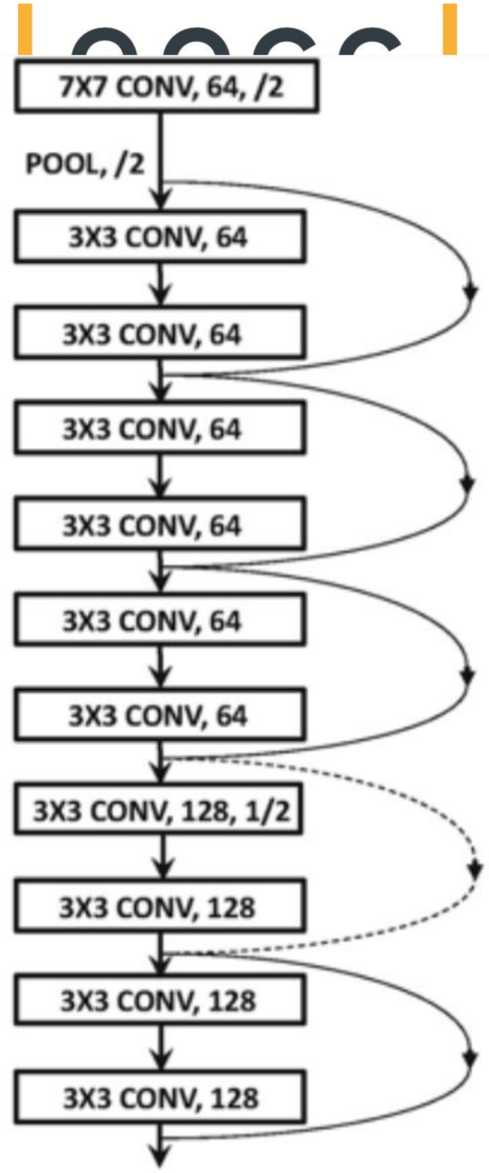
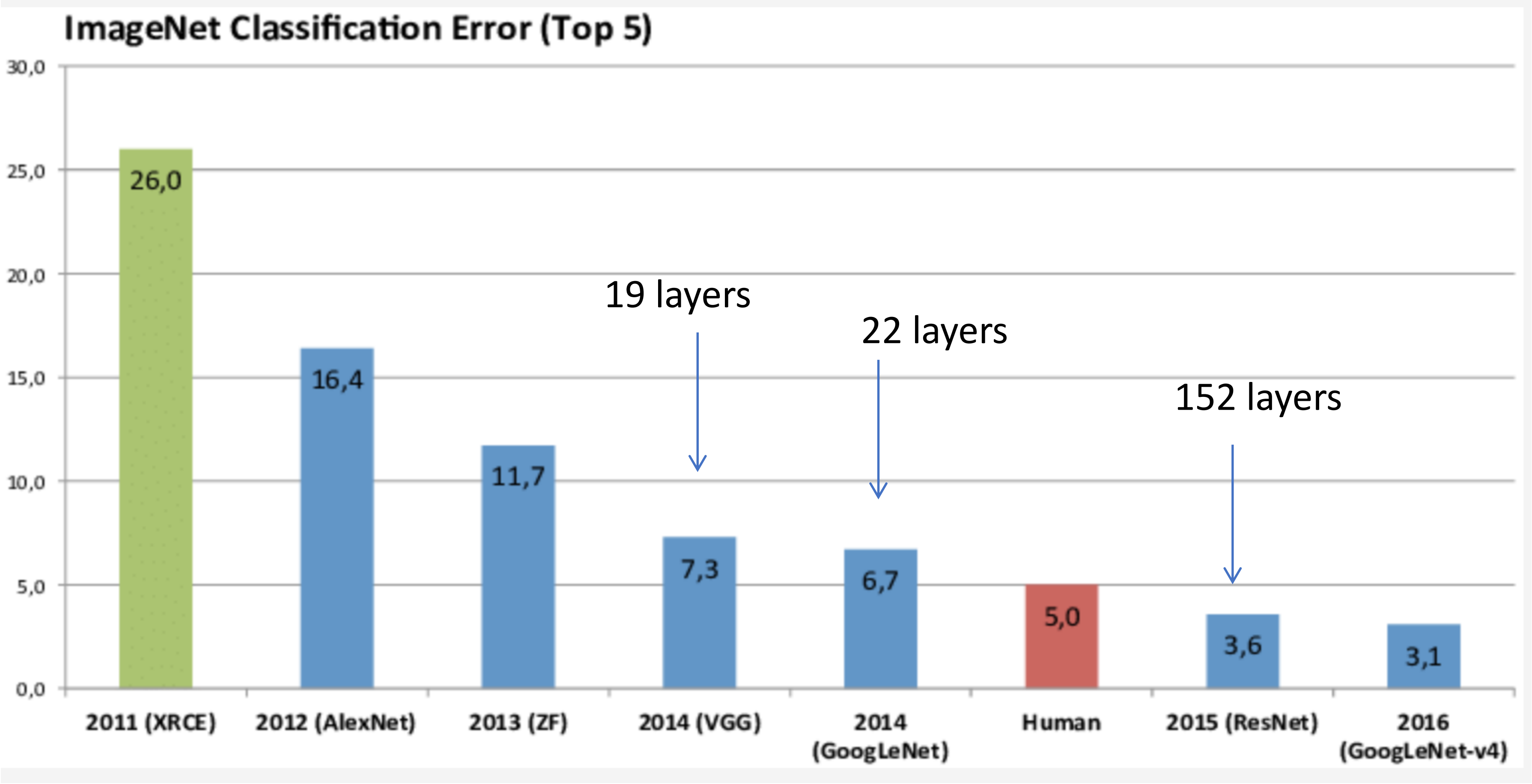


Image Source: https://link.springer.com/chapter/10.1007/978-3-319-94463-0_8

CNN architectures - depth



Zitzewitz, Gustav. "Survey of neural networks in autonomous driving." (2017)

Some Popular Architectures

- ResNet
  - A ”residual network” : Links to skip layers
- Inception Network
  - Multiple filters combined so that the network can learn the best filter to use
- MobileNet
  - An attempt to lower computation cost using a lower-dimensional filter (“depthwise-seperable convolution”)
- EfficientNet
  - Includes features to trade-off resolution, depth and width to get the best results with the computational resources available

- Earlier networks
  - LeNet, 1989: 7 layers, including convolution, pooling and dense, using sigmoid
  - AlexNet, ~2012: 11 layers, using ReLU
  - GoogLeNet, ~2014: 22 layers
  - VGG-16, ~2018: 21 layers (of which 16 parameterised) aiming to lower number of parameters. More, smaller kernels.

Other Neural Network Architectures

Natural Language Processing
as a key driver in NN architecture design

Bag of Words

- Vector of length *n* to represent occurrence of words from a dictionary of *n* words
- Can be suitable for basic document classification
- Issue: Does not take order into account

RNN

- Has some memory
- Issue: Vanishing & Exploding Gradients

LSTM

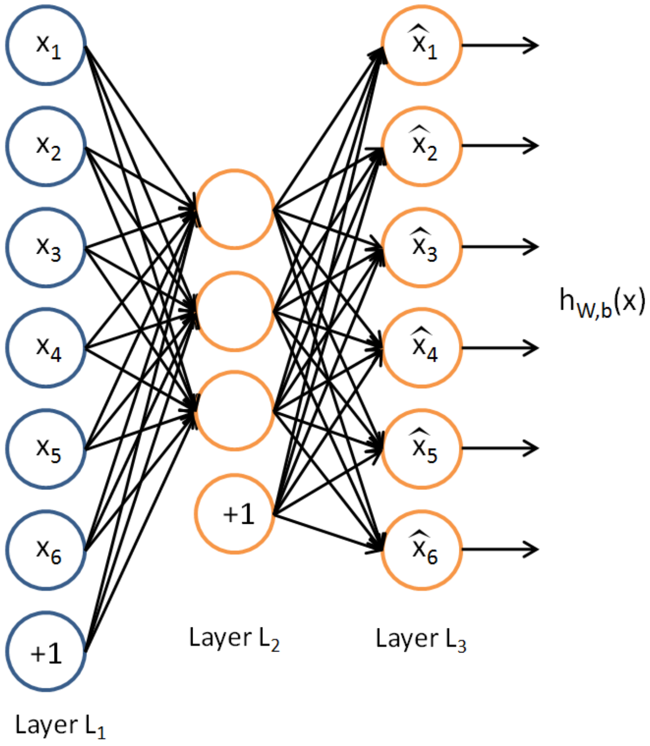
- Proposed 1997
- Introduce a mechanism for long-term memory
- Issue: Difficult to train
- Issue: Transfer Learning doesn't work well
- Issue: Sequential calculation

Transformers

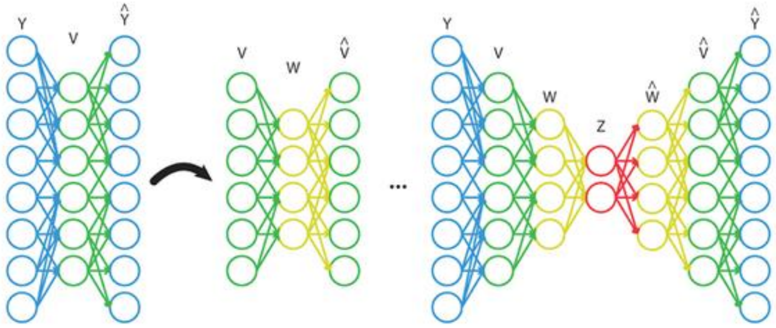
- Use self attention
- Can process data in parallel
  - Although still computationally expensive
- Appear to give better results than LSTM
- Better at keeping track of context

Types of DNN

- Autoencoder
  - Neural network used for dimensionality reduction and feature learning
  - Comprises an encoder to map data to a lower-dimensional representation and a decoder for reconstruction
  - Layers trained separately with input == output
  - Data compression, image denoising, anomaly detection.
  - Feature learning for subsequent tasks
- Can also use multiple DNN together, i.e. Generative Adversarial Network (GAN)
  - Generator network to create data
  - Discriminator network to try and identify genuine data
  - Generator aims to produce data like the real data
  - Discriminator distinguishes between real and fake data
  - Loss functions to optimise real data production and identification

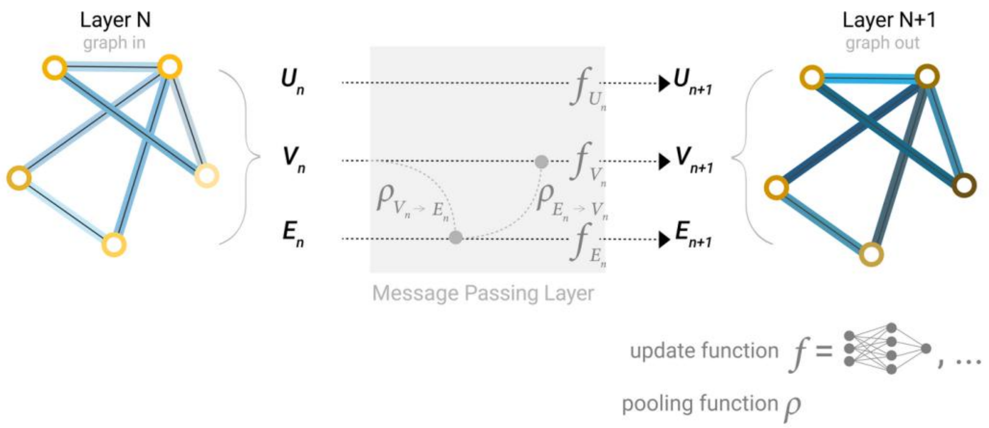


http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/



Berniker M, Kording KP. Deep networks for motor control functions. *Frontiers in Computational Neuroscience*. 2015;9(32). doi:10.3389/fncom.2015.00032.

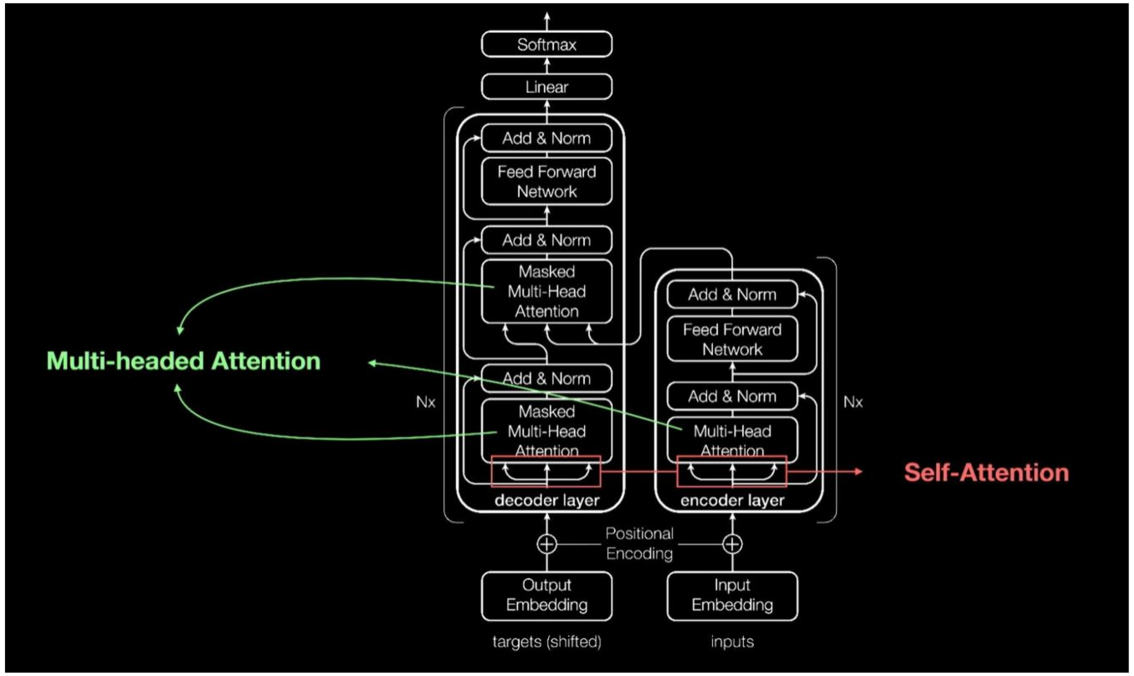
Graph Neural Networks (GNN)



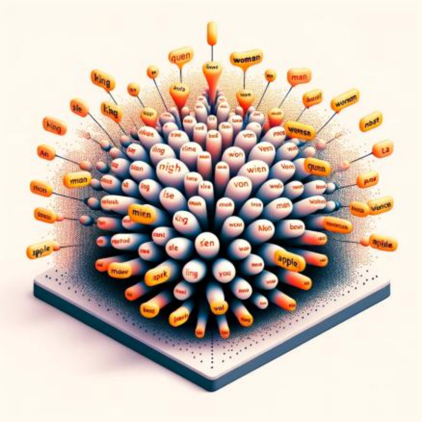
https://distill.pub/2021/gnn-intro/

- Different approaches to GNN
  - Graph: Whole structure categorisation
  - Node: Classification of elements
  - Edge: Relationship between elements
  - Collapsing graphs to matrices for computation so care about adjacency
- Typically have a multi-level perceptron (MLP) for each aspect of the graph (nodes, edges, etc…)
  - Plus pooling layers to couple these together (nodes influence edges, etc…)
- Nearest neighbour message passing to maintain locality
  - Variable form of convolution (no fixed kernels)

Transformers

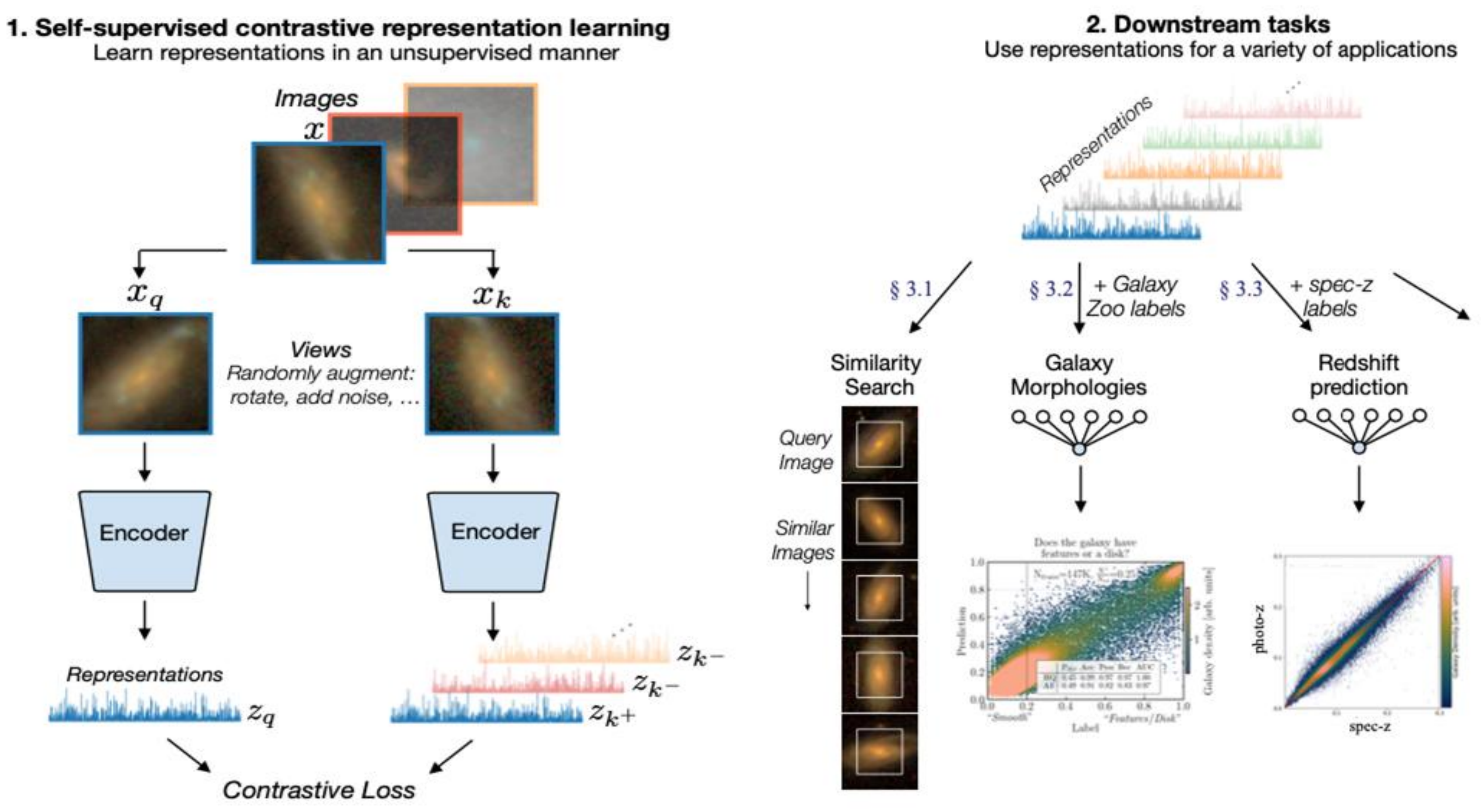


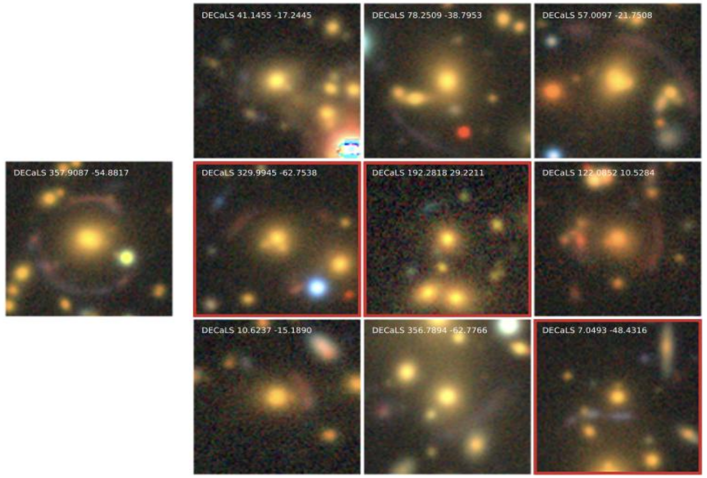
https://blogs.nvidia.com/blog/what-is-a-transformer-model/



- LLM built on transformer models
  - The feed forward network in the encoder and decoder is still a standard DNN (if often very large)
  - GPT-3 has something like 95 layers, 175 billion parameters (proportional to neurons)
  - Attention calculations are small matrix operations on word placement
    - Attention scores
- Word embeddings key part of transformers
  - Mapping words into high dimensional space
  - Embedding is learned as the model is trained

Example use: Identifying galaxy features





Initial approach: Hayat et. al. (2020) arXiv:2012.13083
Strong-lens analysis: Stein et. al. (2021) arXiv:2110.00023

DNNs

- Neural Networks are **highly-parameterised**, **non-linear** models where the parameters (the **weights** and **biases**) are calculated using **back propagation** to calculate **gradients** which are, in-turn, used to **minimise** a loss function.
- Increasing depth gives more parameters:
  - Tends to allow for a more accurate model
  - Run the risk of overfitting
  - Computationally costly
- NN architectures are often selected for the problem at hand but often this boils down to finding different ways to try to optimally trade off
  - Accuracy
  - Training time & convergence
  - Prediction time

DNNs

- Key to DNNs usefulness are
  - Growth in computing to enable large linear algebra operations to be undertaken quickly
  - Depth and breadth of the network has improved performance
- Still many challenges to functionality
  - Over fitting
  - Exploding or vanishing gradients
  - Reliability/Generalisability
  - Network design
  - Data availability
- HPC challenges are common
  - Parallelisation
  - Memory capacity
  - Communication
  - Load balance
- Try out some network features for yourselves
  - https://playground.tensorflow.org