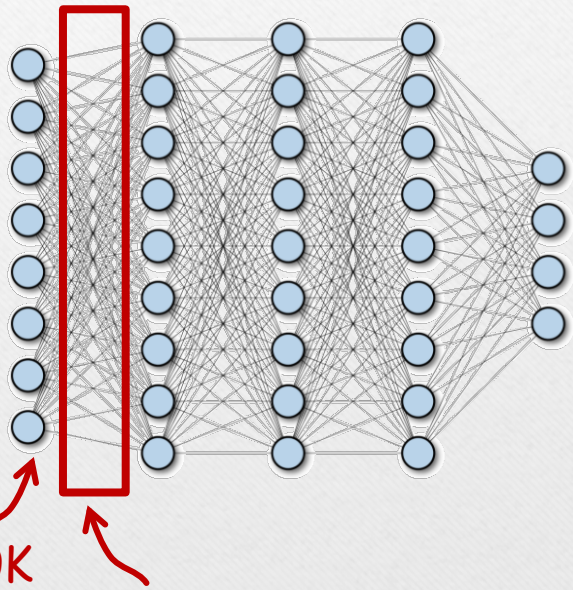
Convolutional Neural Network

ECON 4810

Image Classification

Dog

Image Classification with Neural Network



300K

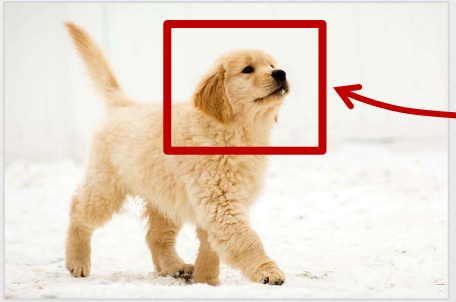
Each hidden neuron has 300K params.
If 100 neurons, 10 mil. params.

250px × 400px = 100K pixels
× RGB = 300K numbers

Dog

Problem with Using Fully Connected Network

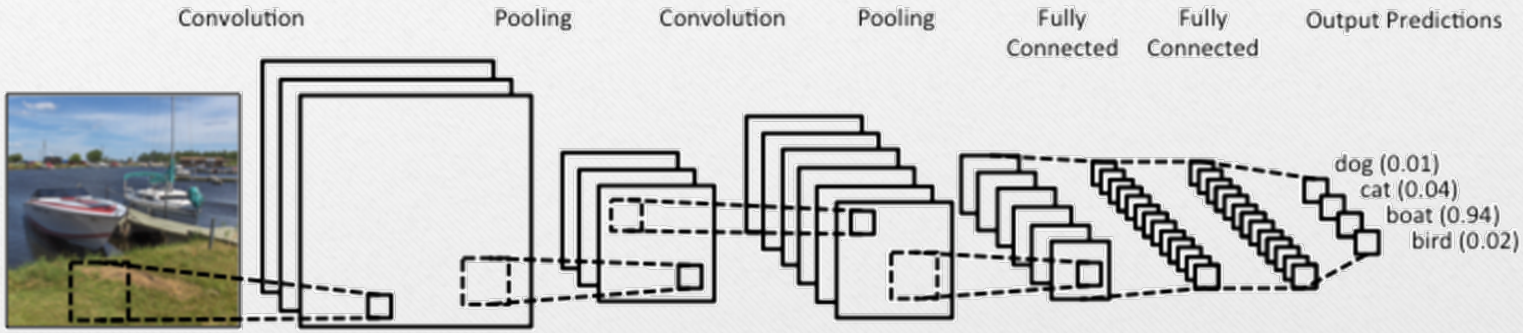
- Using a fully-connected network on image of even moderate size would require an enormous number of parameters.
- Does the understanding of what is in an image really required that many connections?



The model only needs to look at a neighborhood of pixels each time.

Convolutional Neural Networks

Each neuron is only connected to a neighborhood of outputs below it



Source: WILDML

The Competition that Sparked all the Interest

IMAGENET **Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)**

Held in conjunction with PASCAL Visual Object Classes Challenge 2012 (VOC2012)

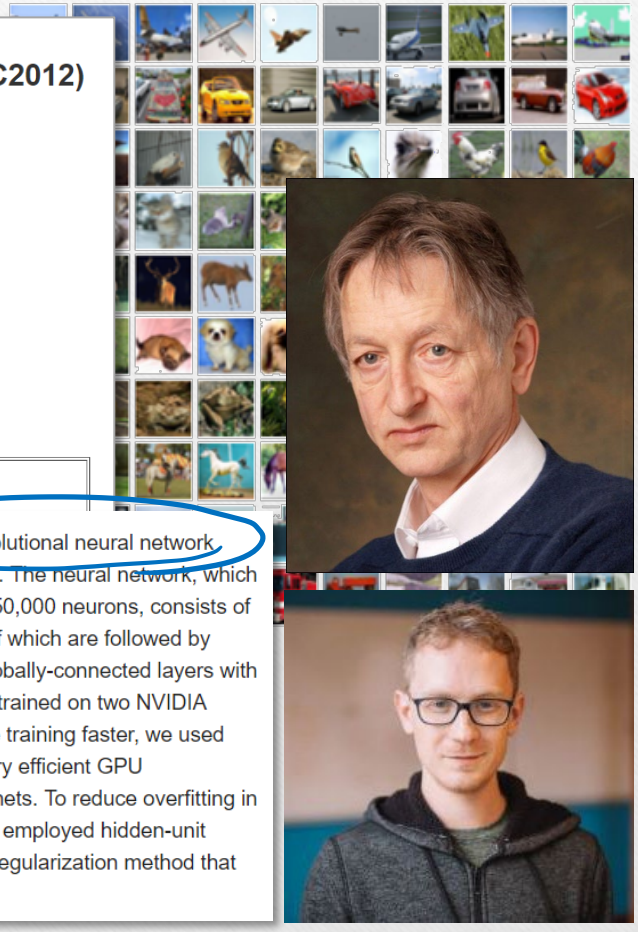
Back to Main page

All results

- Task 1 (classification)
- Task 2 (localization)
- Task 3 (fine-grained classification)
- Team information and abstracts

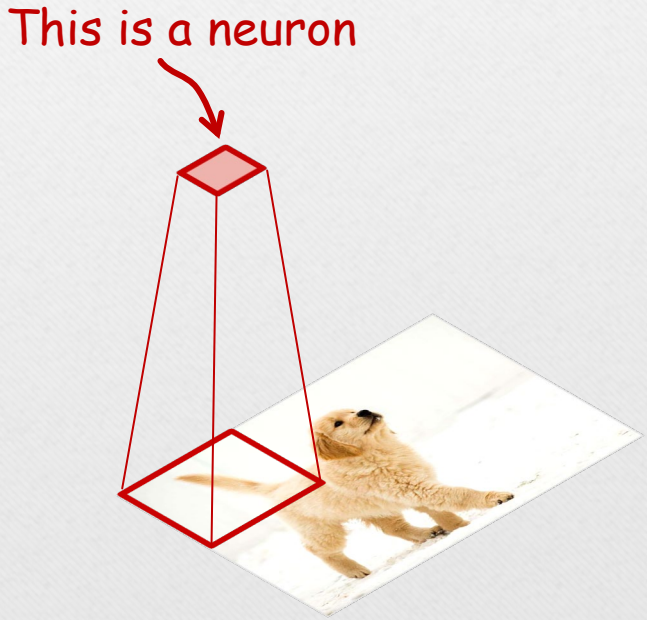
Task 1

Team name	Filename	Error (5 guesses)	Description
SuperVision	Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton University of Toronto		Our model is a large deep convolutional neural network trained on raw RGB pixel values. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three globally-connected layers with a final 1000-way softmax. It was trained on two NVIDIA GPUs for about a week. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of convolutional nets. To reduce overfitting in the globally-connected layers we employed hidden-unit "dropout", a recently-developed regularization method that proved to be very effective.



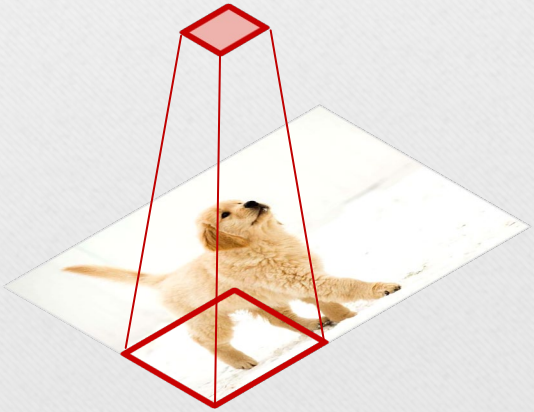
Convolution Layer

In a convolution layer, a **single** neuron goes through the input, processing only a small part at a time.



Convolution Layer

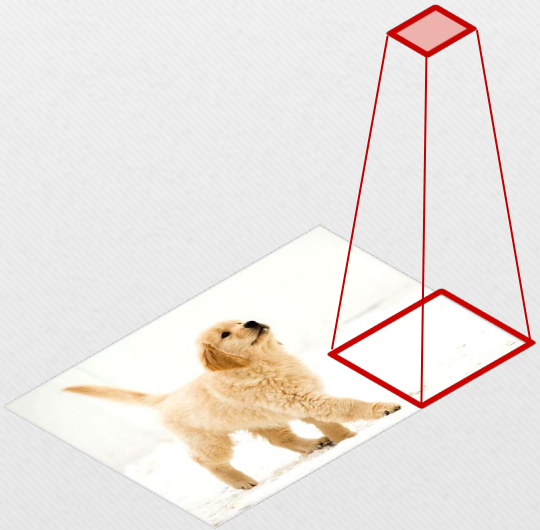
In a convolution layer, a **single** neuron goes through the input, processing only a small part at a time.



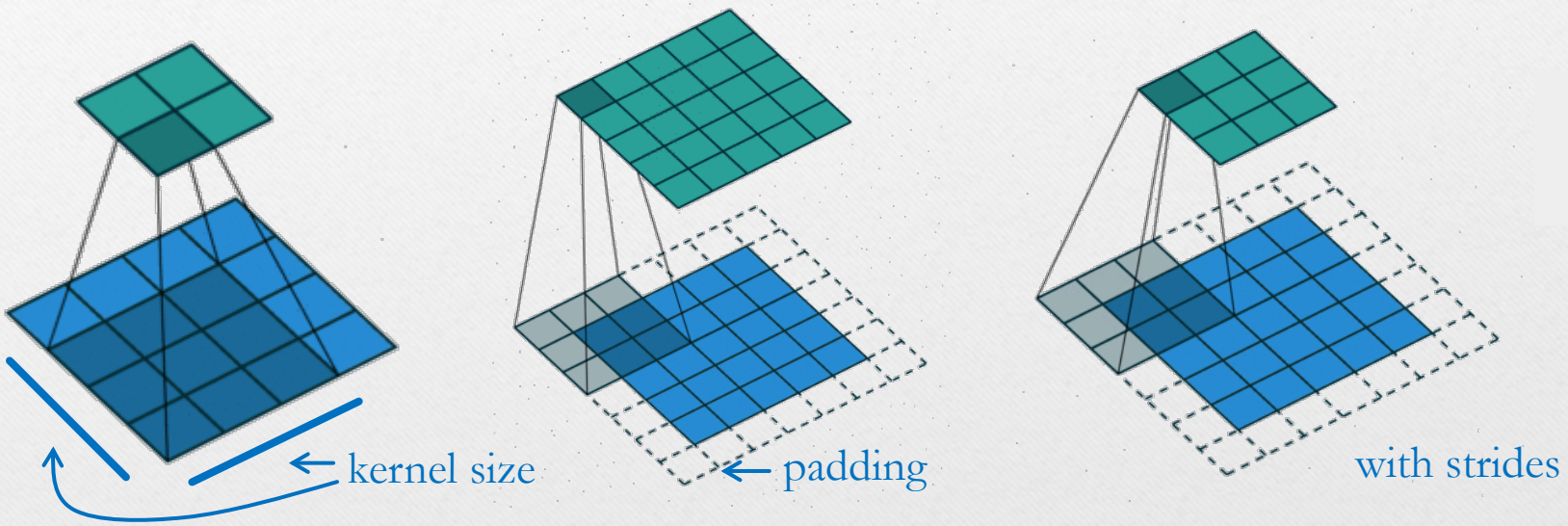
Convolution Layer

Each pixel in the input window has its own weight. E.g., if the neuron takes 2×2 pixels at a time, there are four weights + bias.

Because there is only one neuron, very few parameters are needed to cover a whole image.



Convolution Layer Settings

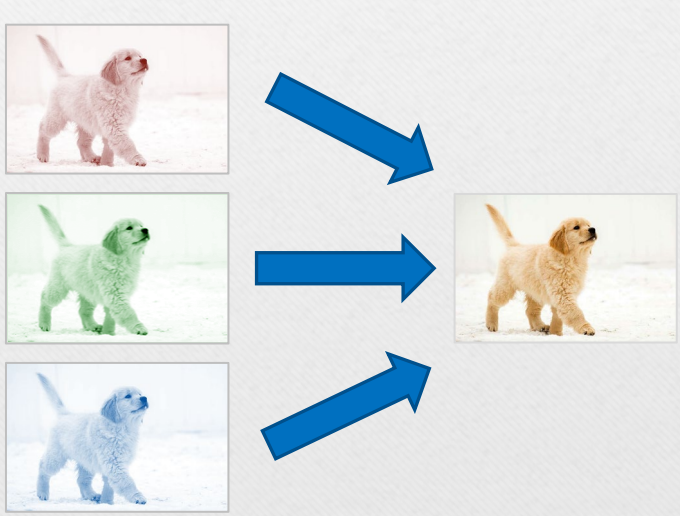


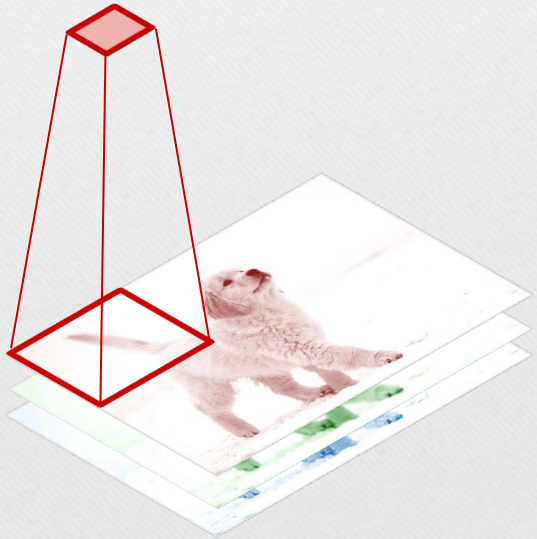
https://github.com/vdumoulin/conv_arithmetic

Color Channels

Color images are typically stored in RGB format, so there are three numbers per pixel.

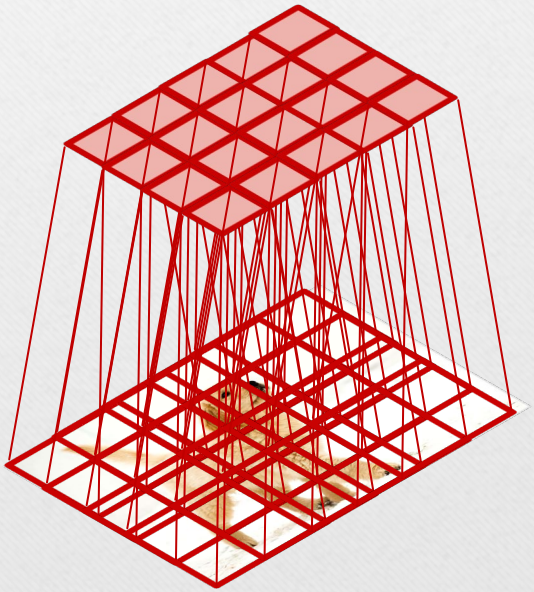
The neuron takes in all three numbers, each with their own weight. E.g., $2\text{ px} \times 2\text{ px} \times 3\text{ colors} = 12$ weights.





Parallel Processing

While conceptually we can think there is one neuron going through the input, in practice this process is done in parallel, all in one go.

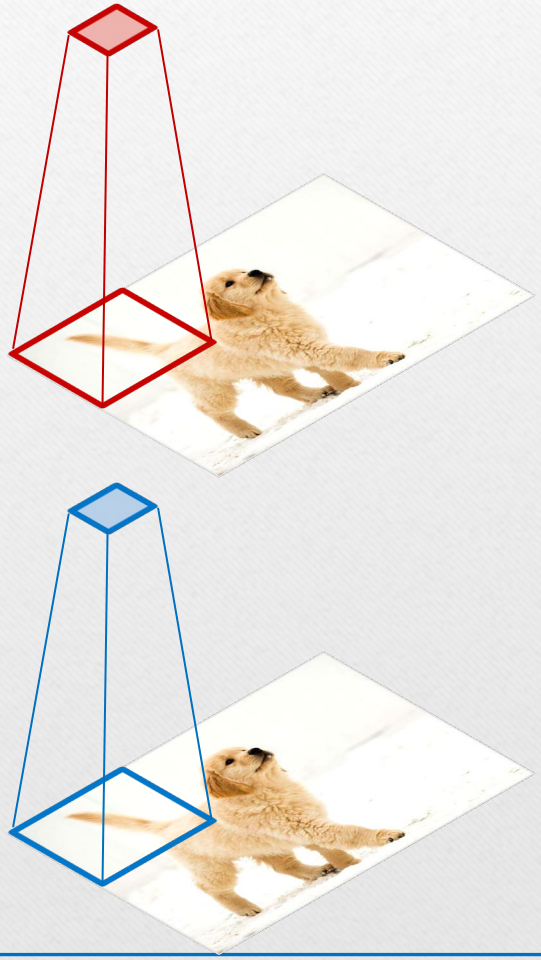


Filter

A single neuron learns to detect a specific feature in the image.

For the layer to be able to detect multiple feature, it needs multiple neurons.

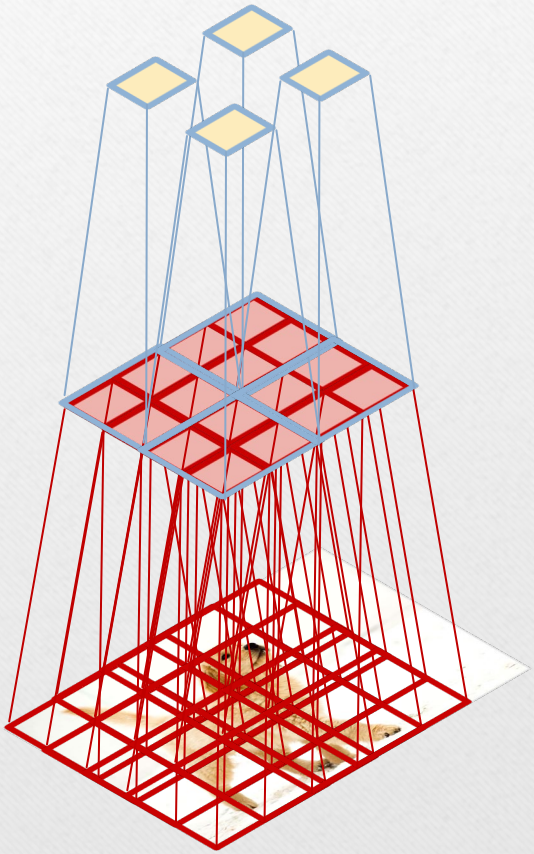
Additional neurons are called **filters**.



Pooling

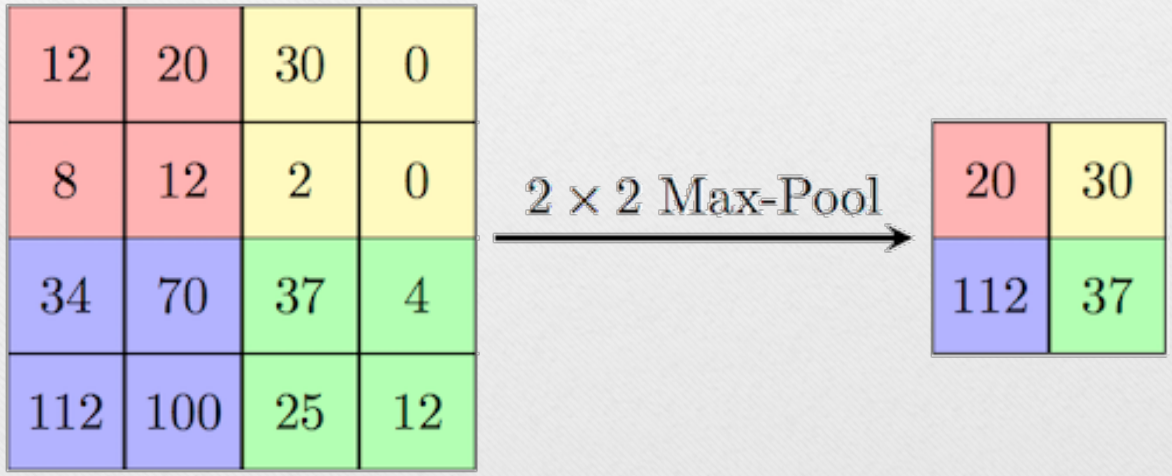
With features detected, we do not need as much resolution.

We scale down the output of the convolution layer. This is called **pooling**.



Pooling

The most common type of pooling is **max-pooling**, which means keeping only the biggest value over an input window.



http://deep-learning.co.uk/

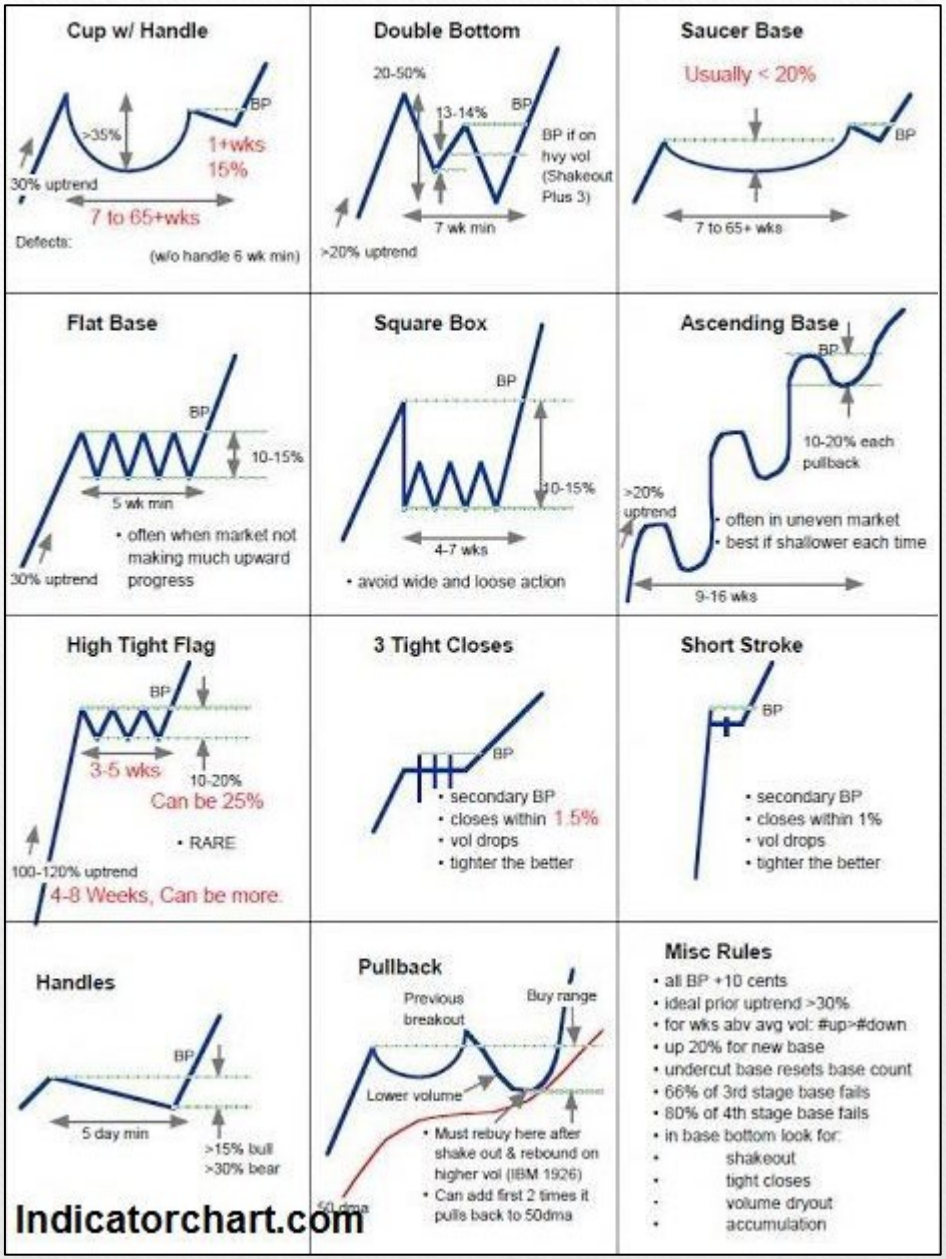
Visualization

- https://adamharley.com/nn_vis/mlp/3d.html
- https://poloclub.github.io/cnn-explainer/

Using CNN for Time Series

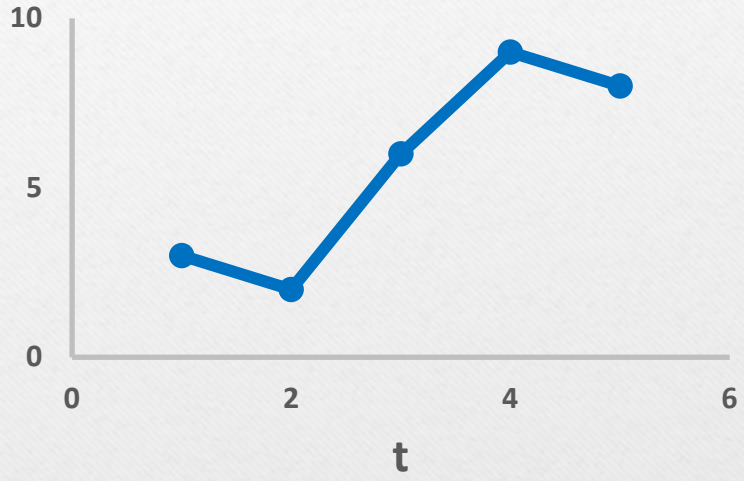
Take technical analysis as an example: the analysis is highly graphical in nature.

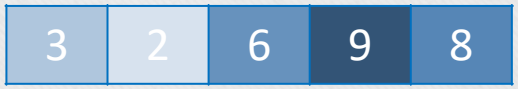
If CNN is good for image recognition, it should also be good for time-series pattern recognition.

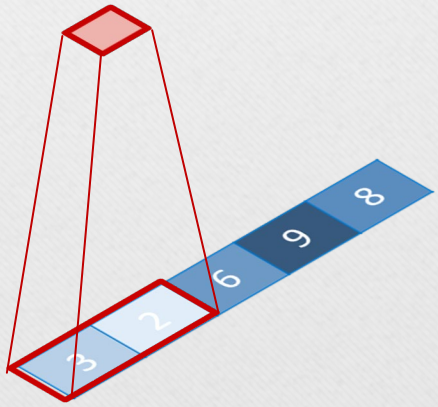


CNN for Time Series

We can treat time series as 1D-images.

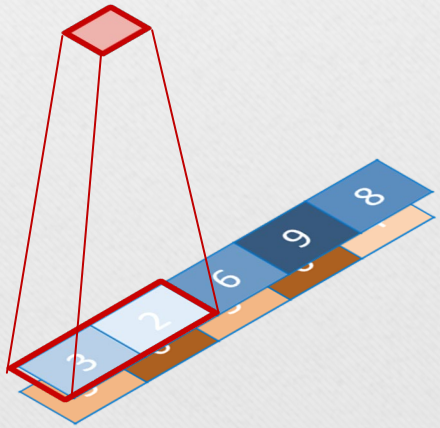






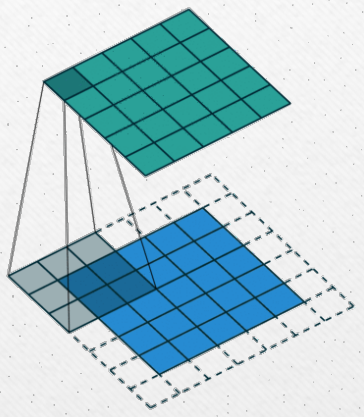
CNN for Time Series

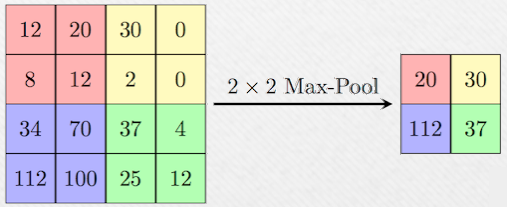
Multiple time series can be treated just like multiple colors.

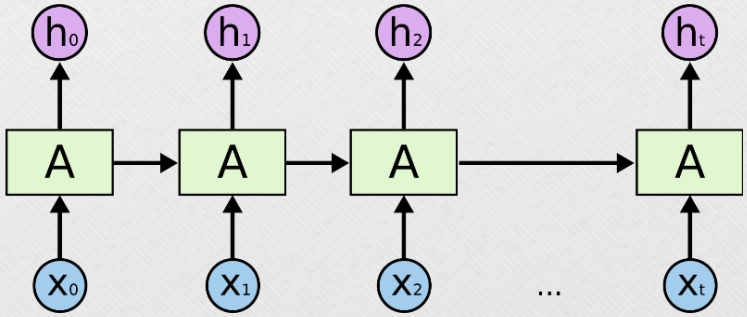


Speed: CNN vs RNN

- CNN is very fast because a small number of neurons—with a small number of weights—working over the whole image, in parallel.
- RNN, due to its sequential nature, is much slower.
  - Different weights need to be loaded into the GPU/CPU cores when going through different time steps.
  - Being autoregressive means it is more likely to suffer from vanishing or exploding gradient.

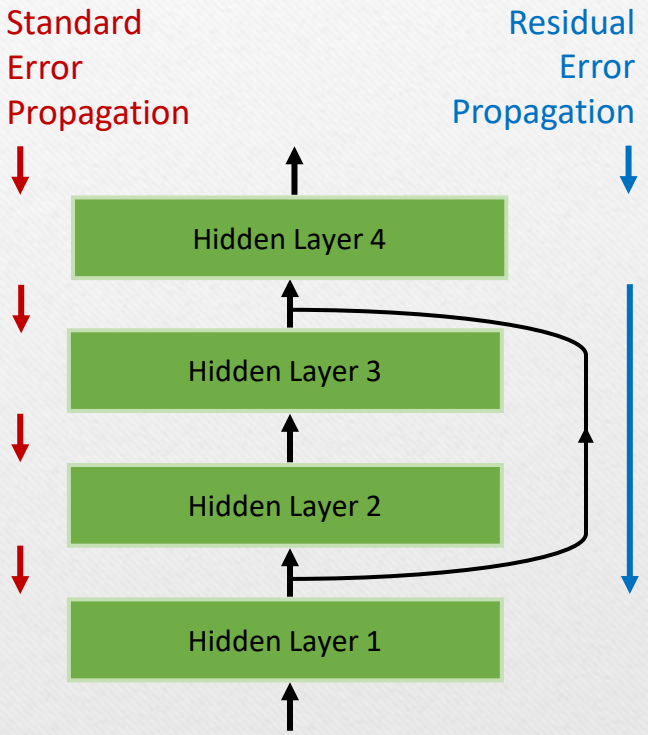






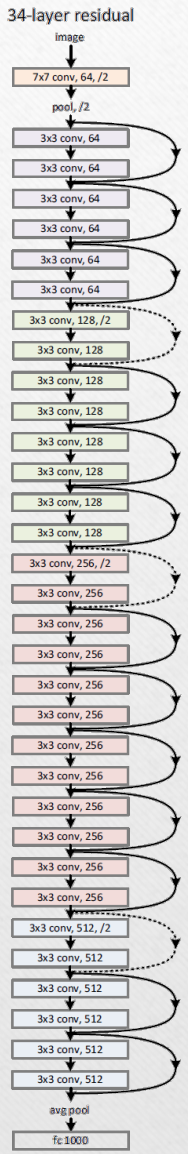
Residual Learning

- **Residual Learning** passes the activation of a layer to a higher, non-adjacent layer directly, in additional to the usual activation pathway between the two layers.
- In the example here, the activation of Layer 1 is added to the activation of Layer 3.
- This helps mitigate the vanishing gradient problem because you can propagate errors down to lower layers with fewer steps in between.



Residual Learning

- Residual Learning allows the training of very deep networks.
- Original paper: Deep Residual Learning for Image Recognition
- A 152-layer CNN with residual learning achieved the highest performance on the ImageNet dataset in 2015.
- Nowadays, the network design used in the paper is simply referred to as **ResNet**.
- AlphaGo Zero also uses residual learning.



ALPHAGO ZERO CHEAT SHEET

The training pipeline for AlphaGo Zero consists of three stages, executed in parallel

SELF PLAY

Create a 'training set'

The best current player plays 25,000 games against itself

See MCTS section to understand how AlphaGo Zero selects each move

At each move, the following information is stored

The game state
(see 'What is a Game State section')

The search probabilities
(from the MCTS)

The winner
(+1 if this player won, -1 if this player lost - added once the game has finished)

RETRAIN NETWORK

Optimise the network weights

A TRAINING LOOP

Sample a mini-batch of 2048 positions from the last 500,000 games

Retrain the current neural network on these positions
– The game states are the input (see 'Deep Neural Network Architecture')

Loss function
Compares predictions from the neural network with the search probabilities and actual winner

PREDICTIONS **p** **v** Cross-entropy + Mean-squared error + Regularisation **π** ACTUAL

After every 1,000 training loops, evaluate the network

EVALUATE NETWORK

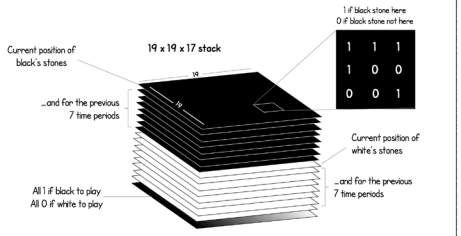
Test to see if the new network is stronger

Play 400 games between the latest neural network and the current best neural network

Both players use MCTS to select their moves, with their respective neural networks to evaluate leaf nodes

Latest player must win 55% of games to be declared the new best player

WHAT IS A 'GAME STATE'



This stack is the input to the deep neural network

THE DEEP NEURAL NETWORK ARCHITECTURE

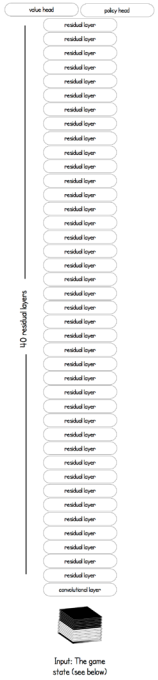
How AlphaGo Zero assesses new positions

The network learns 'tabula rasa' (from a blank slate)

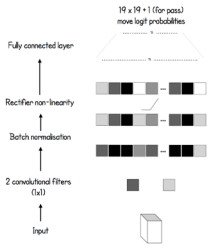
At no point is the network trained using human knowledge or expert moves

The value head

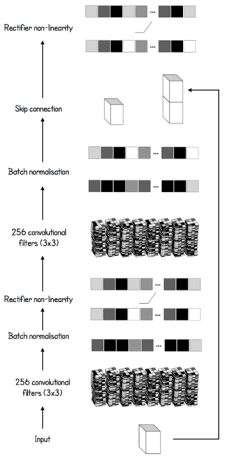
The network



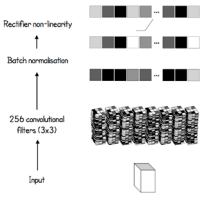
The policy head



A residual layer

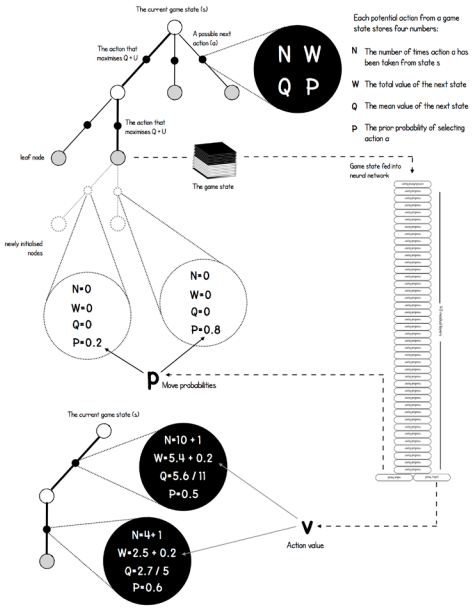


A convolutional layer



MONTE CARLO TREE SEARCH (MCTS)

How AlphaGo Zero chooses its next move



First, run the following simulation 1,600 times...

Start at the root node of the tree (the current game state)

1. Choose the action that maximises...

$$Q + U$$

The mean value of the next state

A function of P and N that increases if an action hasn't been explored much, relative to the other actions, or if the prior probability of the action is high

Early on in the simulation, U dominates (more exploration), but later Q is more important (less exploration)

2. Continue until a leaf node is reached

The game state of the leaf node is passed into the neural network, which outputs predictions about two things:

p Move probabilities

v Value of the state (for the current player)

The move probabilities p are attached to the new feasible actions from the leaf node

3. Backup previous edges

Each edge that was traversed to get to the leaf node is updated as follows:

$N \rightarrow N + 1$

$W \rightarrow W + v$

$Q = W / N$

...then select a move

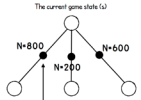
After 1,600 simulations, the move can either be chosen:

Deterministically (for competitive play)
Choose the action from the current state with greatest N

Stochastically (for exploratory play)
Choose the action from the current state from the distribution

$$\pi \sim N^{\frac{1}{\tau}}$$

where τ is a temperature parameter controlling exploration



Choose this move if deterministic
If stochastic, sample from categorical distribution
π with probabilities (0.5, 0.125, 0.375)

Other points

- The sub-tree from the chosen move is retained for calculating subsequent moves
- The rest of the tree is discarded

https://medium.com/applied-data-science/alphago-zero-explained-in-one-diagram-365f5abf67e0

Chihuahua or Muffin?