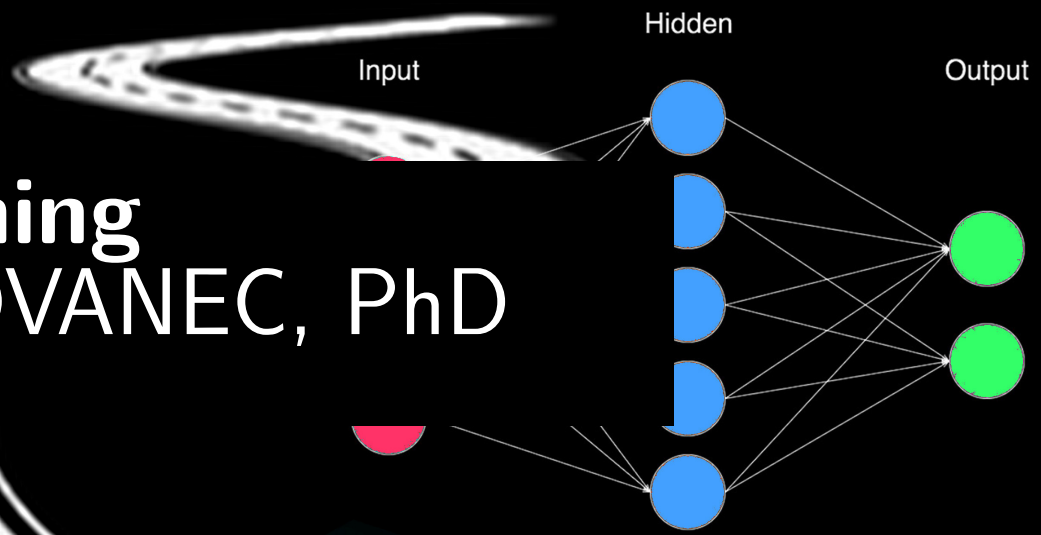
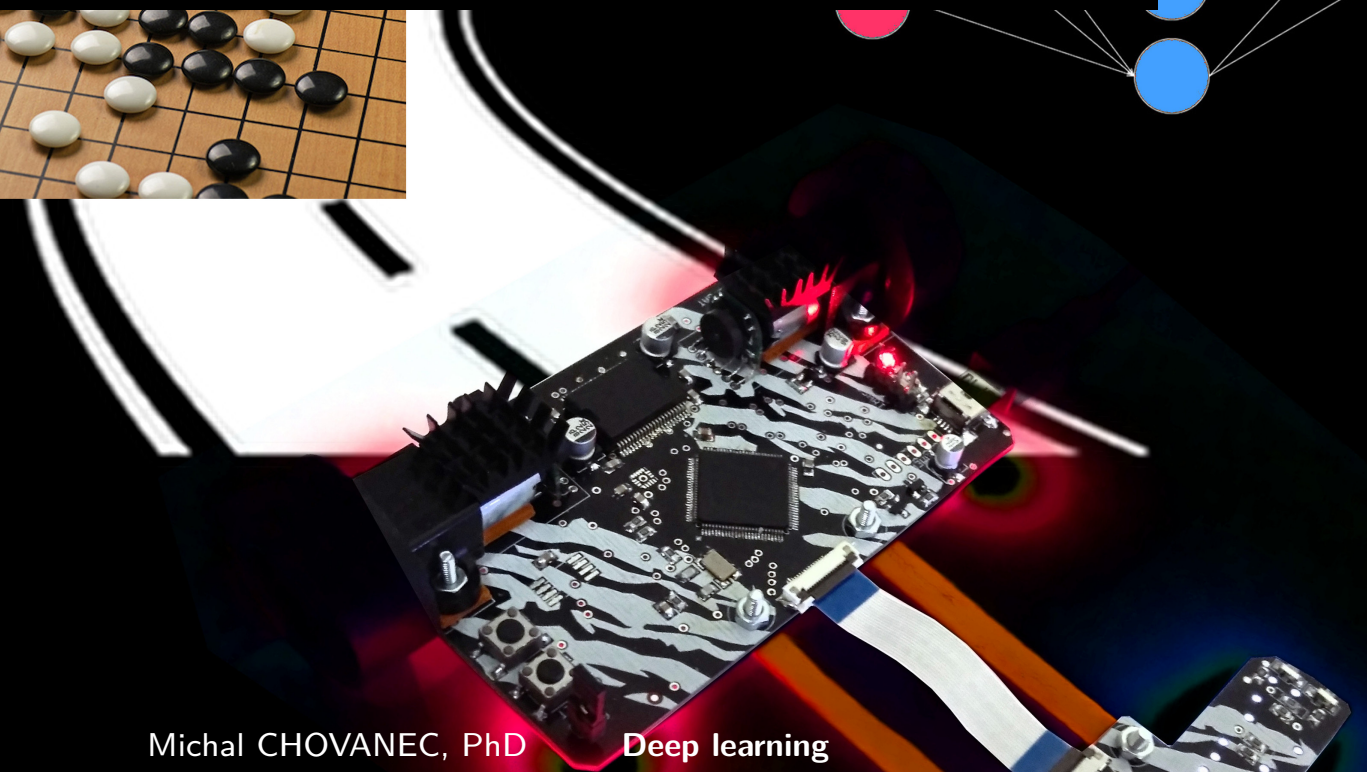
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

(The New Action Value = The Old Value) + The Learning Rate × (The New Information − the Old Information)



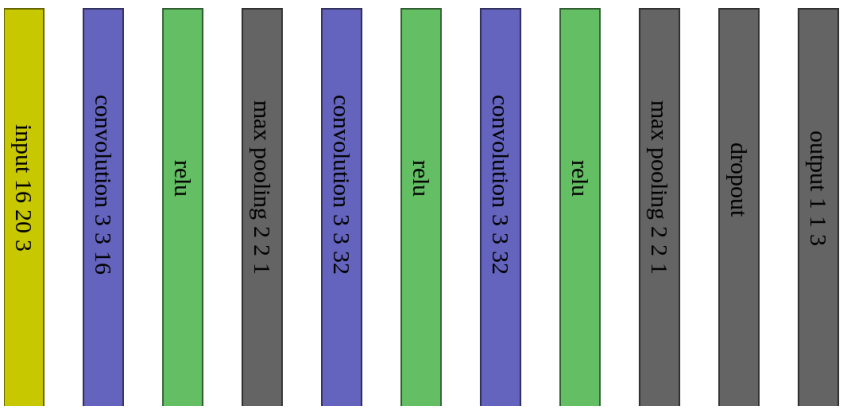
Deep learning

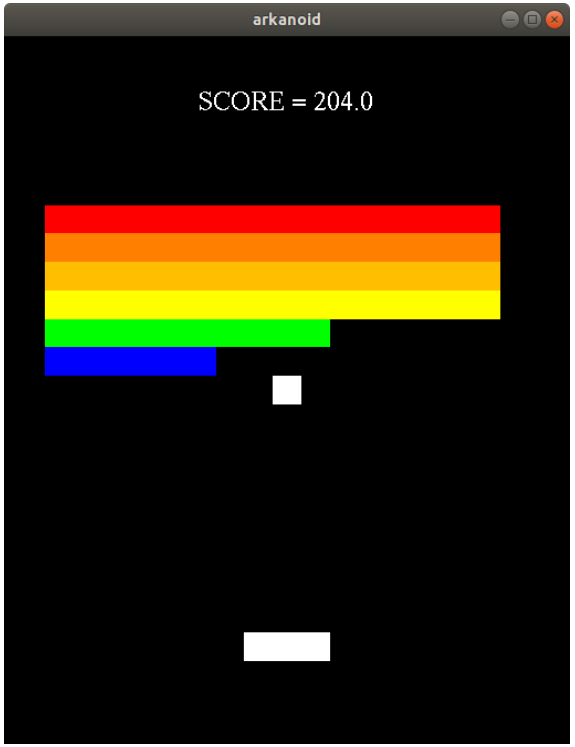
Michal CHOVANEC, PhD



Playing Atari games on super human level

(video)

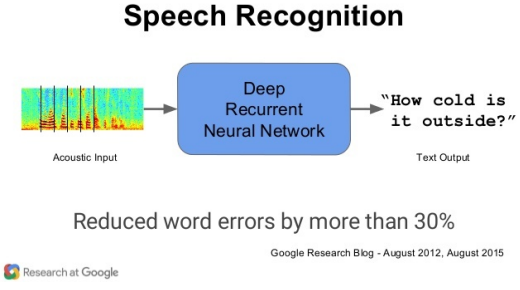


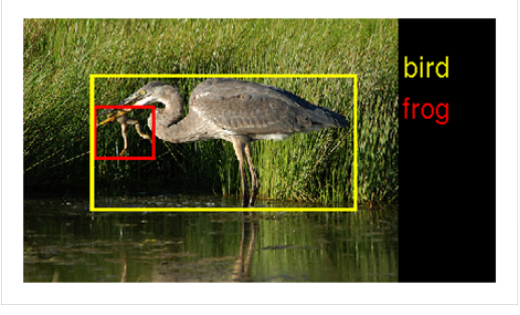


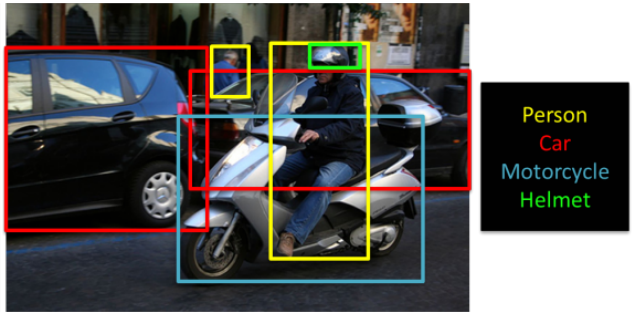
- neurons count **10 000**
  - 500 000 parameters
  - 16 000 parameters
- ant neurons count 250 000
- bee neurons count 960 000

Applications

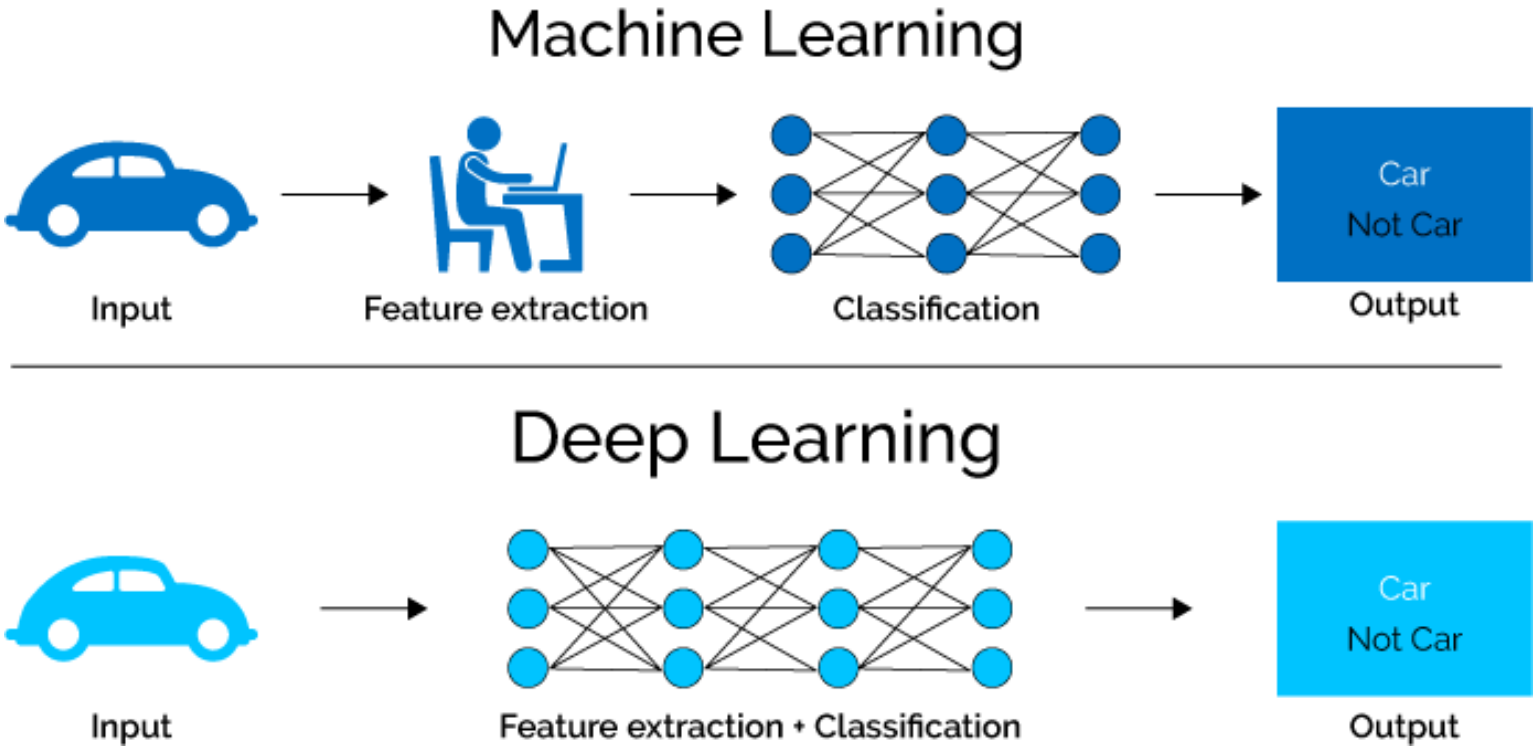
- self driving cars - Tesla model S
- healthcare, biomedical engineering - Cell in fluid
- voice search, control - Google, Amazon Echo
- machine translation - Google translator
- image recognition - Huawei Mate 10
- game bots, robotics - DeepMind, Boston Dynamics





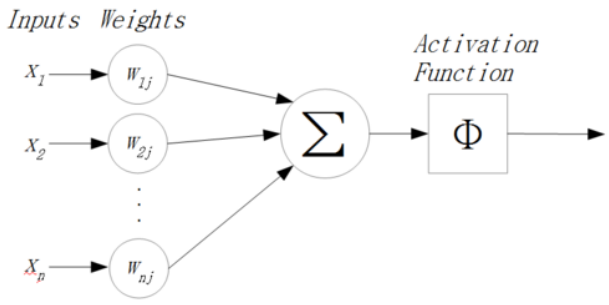


Deep learning [1]

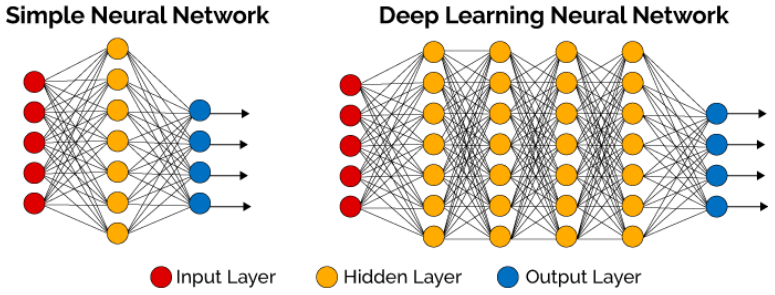


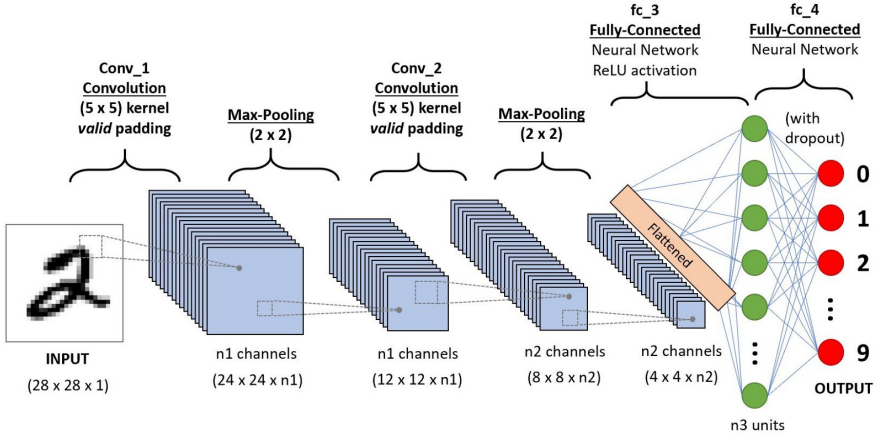
[1] https://medium.com/datamob/clearing-the-buzzwords-in-machine-learning-e395ad73178b

Neural network



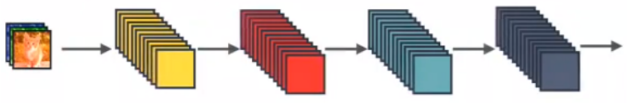
$$y(X) = f(\sum_i X_i W_i + b)$$



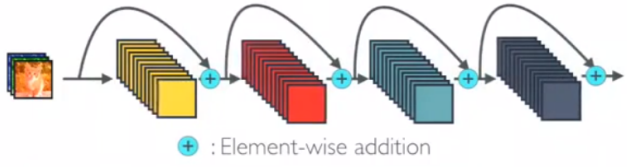


Deep neural network

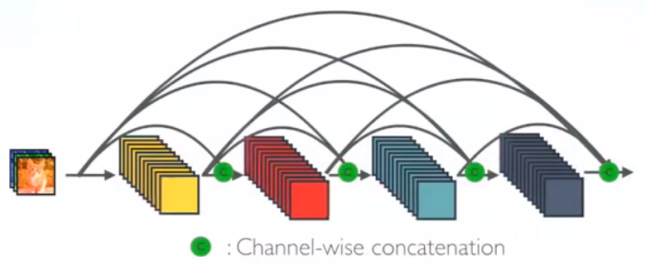
- Old methods (before NN) - 2011, 25.8%
- Convolutional, AlexNet - 2012, 16.4%
- Google inception - 2013, 6.7%
- Microsoft ResNet - 2015, 6.1%
- DenseNet - 2018, 5.17%
- CNN



- ResNET



- DenseNet



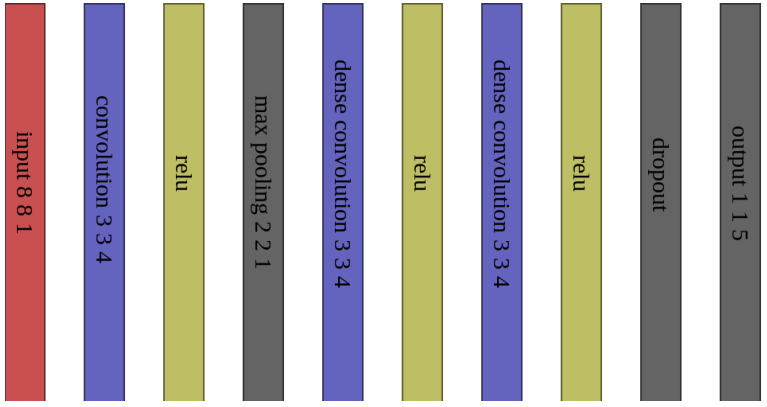
My research

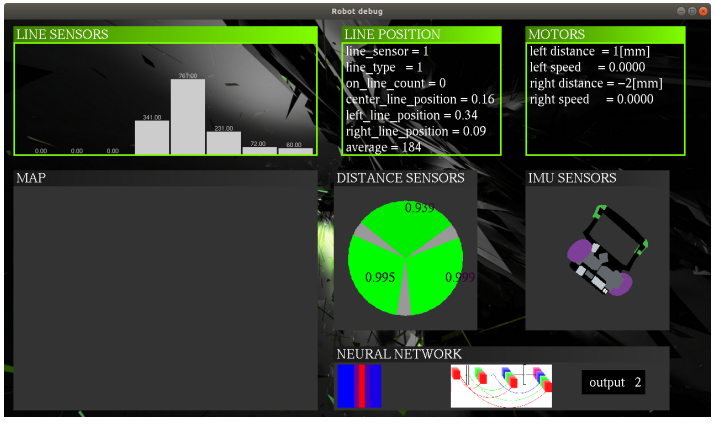
- Robotics - hobby
- Red blood cells trajectory prediction
- Deep reinforcement learning

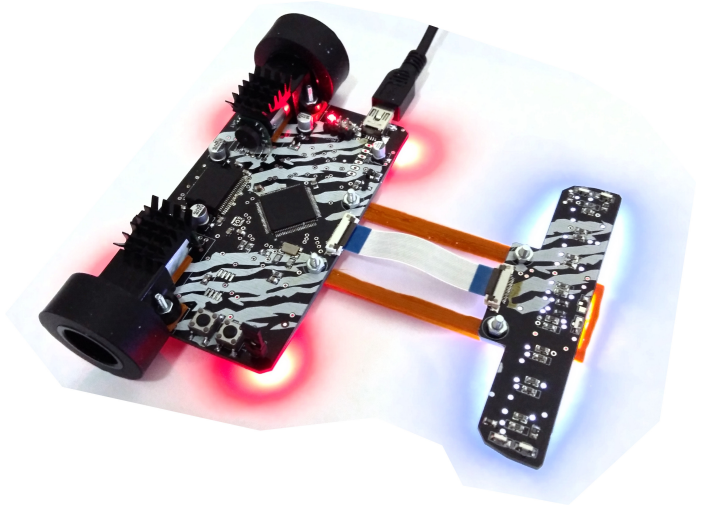
Robotics - line follower

Curve shape classification

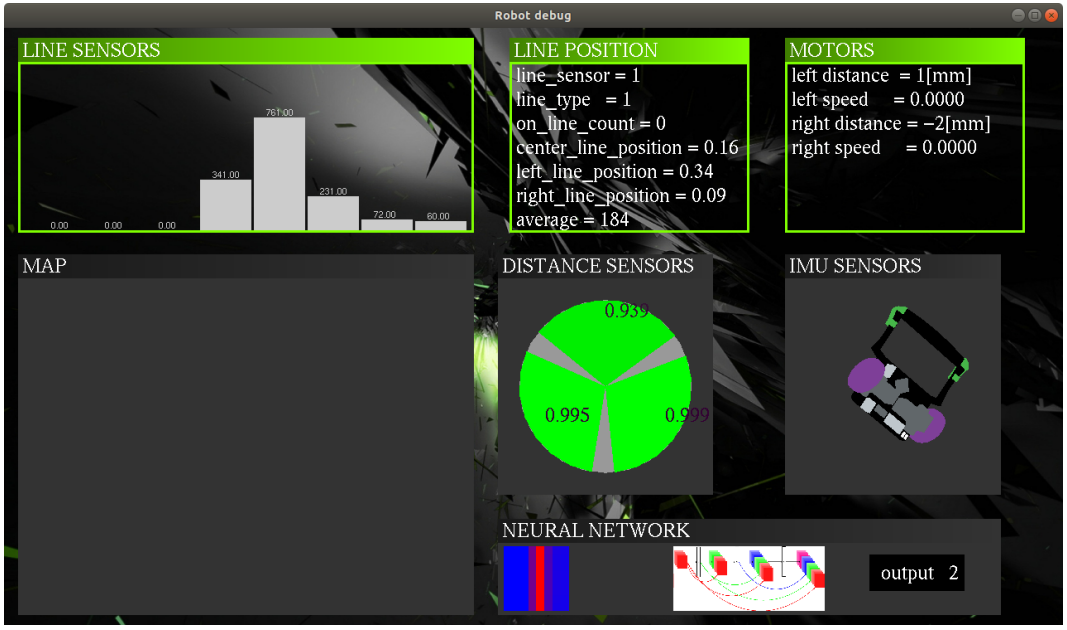
- stm32f303 (72MHz), stm32f746 (216MHz)
- 8x 500nm line sensors
- pololu motors, 1:30
- network input : 8 last line sensors results (8x8 matrix)
- response 4..5ms







Robotics - line follower



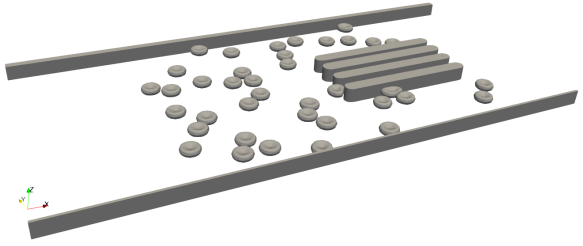
(video)

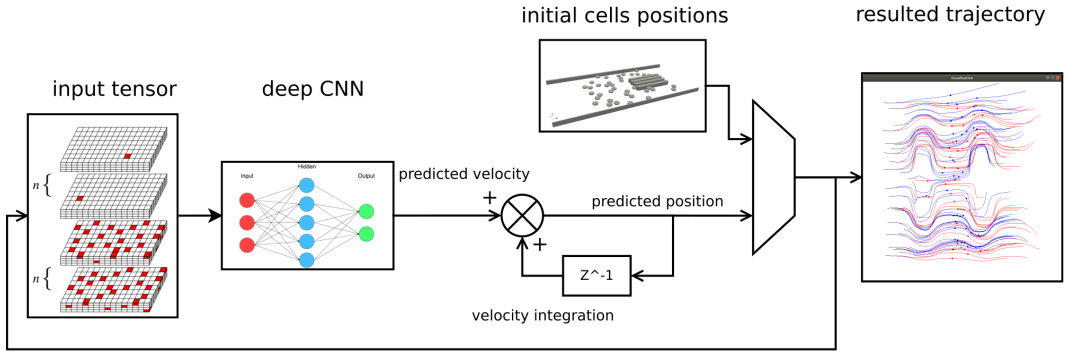
Red blood cells trajectory prediction

Research group **Cell in fluid**

Mgr. Katarína Jasenčáková, PhD thesis

- train DNN to predict RBC trajectory from past
- 15 conv layers network (6hours training on GTX1080ti)
- input : RBC position + 7 past frames + other cells position
- output: RBC predicted velocity



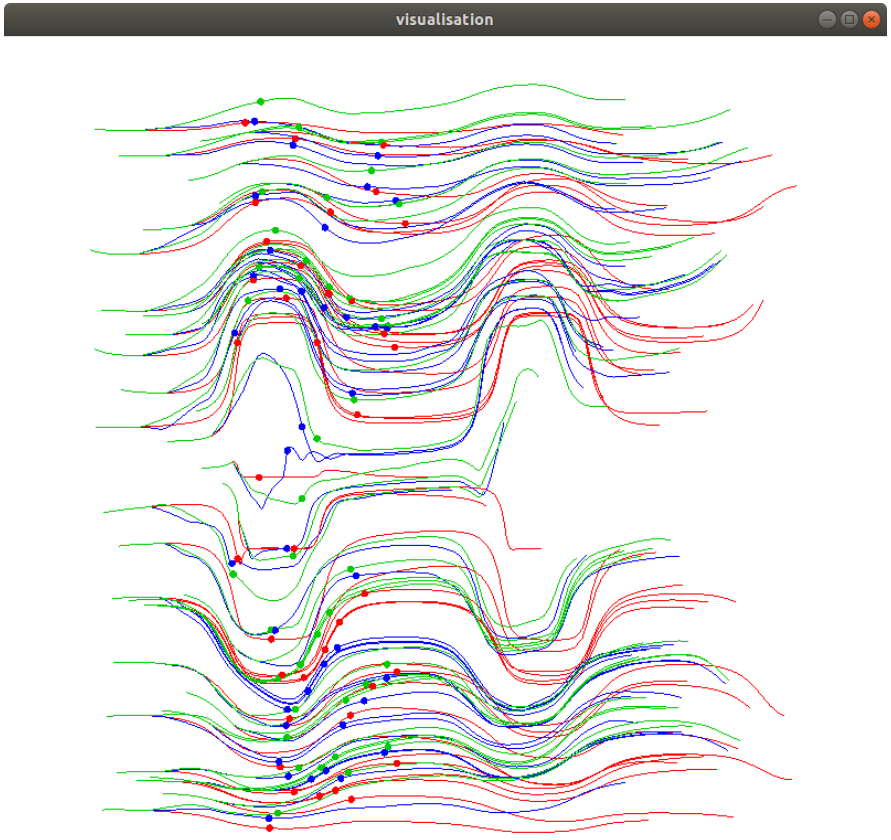


Networks

layer	net 0	net 1	net 2	net 3	net 4	net 5	net 6	net 7
0	fc 256	conv 3x3x32	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
1	fc 64	fc 64	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
2	fc 32	fc 32	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
3	fc 3	fc 3	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
4			conv 1x1x32	conv 1x1x16	conv 1x1x32	conv 1x1x16	conv 1x1x16	conv 1x1x32
5			fc 3	dense conv 3x3x8	fc 3	dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
6				dense conv 3x3x8		dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
7				dense conv 3x3x8		dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
8				dense conv 3x3x8		dense conv 3x3x8	dense conv 3x3x8	dense conv 3x3x8
9				conv 1x1x32		conv 1x1x32	conv 1x1x16	conv 1x1x32
10				fc 3		fc 3	dense conv 3x3x8	dense conv 3x3x8
11							dense conv 3x3x8	dense conv 3x3x8
12							dense conv 3x3x8	dense conv 3x3x8
13							dense conv 3x3x8	dense conv 3x3x8
14							conv 1x1x32	conv 1x1x64
15							fc 3	fc 3

Results

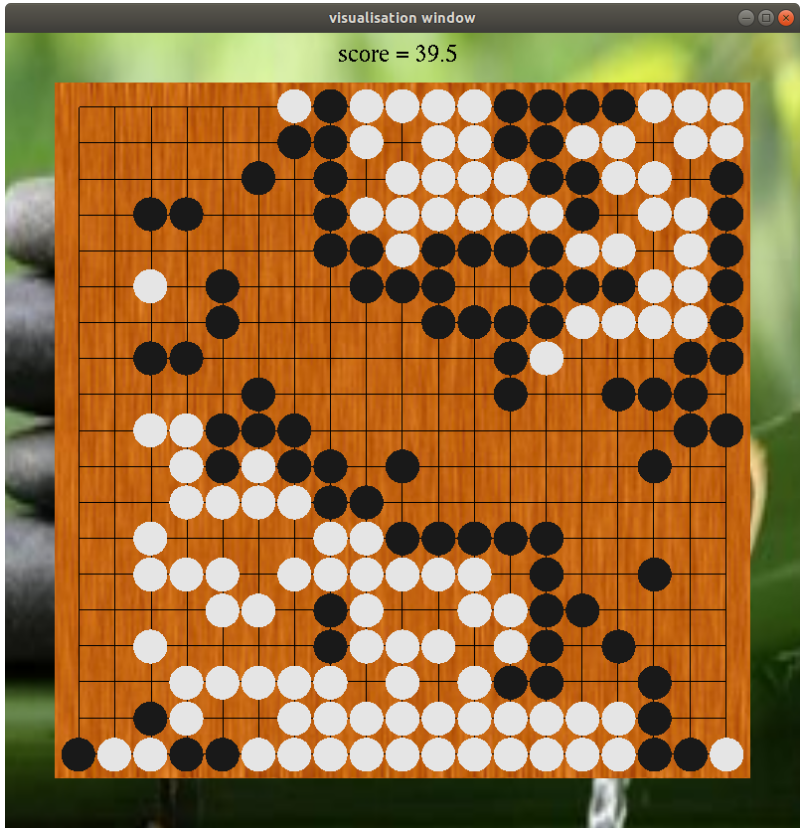
(video)



Results

ID	total error			
	mean [um]	sigma [um]	rms [um]	rms relative [%]
0	-9.8	38.88	40.1	7.23
1	-36.99	215.02	218.18	39.32
2	-3.85	23.11	23.43	4.22
3	-0.32	16.09	16.09	2.9
4	-1.26	11.44	11.51	2.07
5	-0.25	11.27	11.28	2.03
6	-0.85	12.26	12.29	2.22
7	-1.67	11.9	12.02	2.17

Playing GO

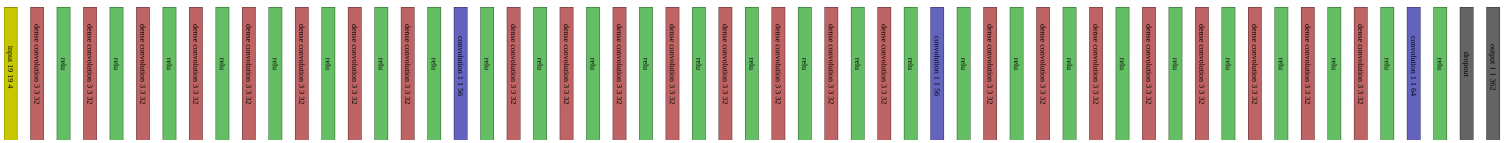


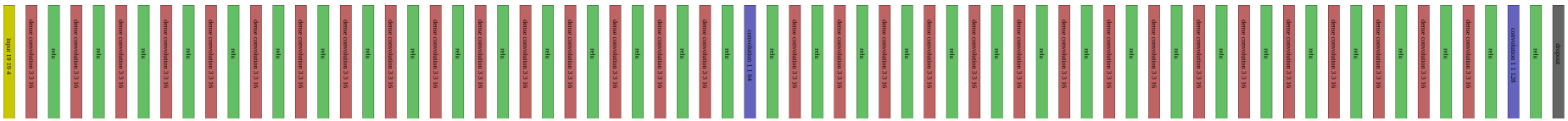
- **supervised training** - train game using Masters games
- **reinforcement learning** - let play two networks against each other

Network architecture

we need to go much deeper for GO

- **28, 35 layers**
  dense blocks + feature pooling layer
- **input**
  4 matrices 19x19: black stones, white stones, empty fields, active player
- **output**
  recommended moves 19x19 + 1 for pass = 362 outputs

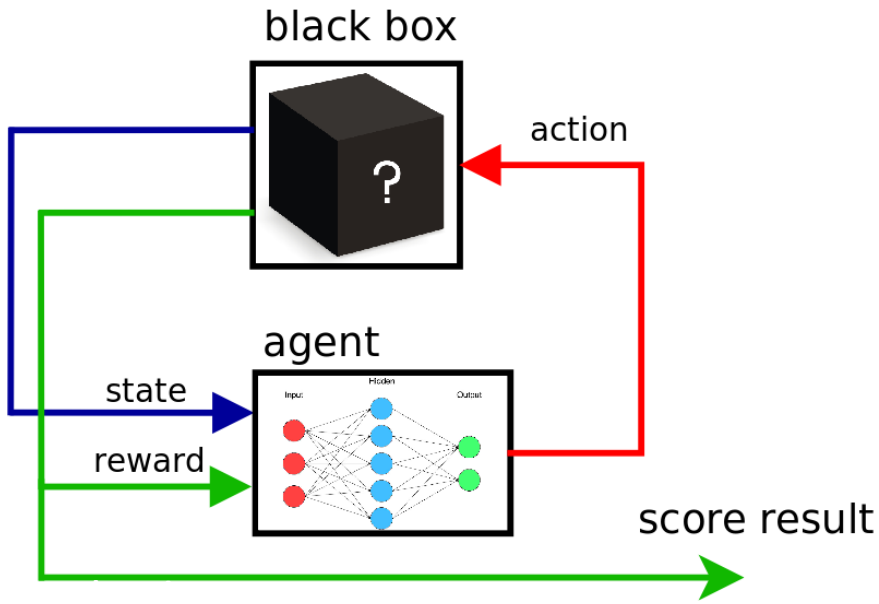




Reinforcement learning

- learn from punishment and rewards
- learn to play a game with unknown rules
- obtain **state**, s
- choose **action**, a
- **execute** action
- obtain **reward**, R
- learn from **experiences**, $Q(s,a) = R + \gamma \max_{\alpha'} Q(s', \alpha')$

$$Q(s,a) \approx \hat{Q}(s,a;w)$$



Usefull links

CHRISTOPHER J.C.H. WATKINS : Q-learning
http://www.gatsby.ucl.ac.uk/~dayan/papers/cjch.pdf

Richard S. Sutton : Reinforcement Learning: An Introduction
https:
//www.amazon.com/Reinforcement-Learning-Introduction-Adaptive-Computation/dp/0262193981

Google DeepMind : Playing Atari with Deep Reinforcement Learning
https://arxiv.org/pdf/1312.5602.pdf

Google DeepMind : Dueling Network Architectures for Deep Reinforcement Learning
https://arxiv.org/pdf/1511.06581.pdf

Google DeepMind :Mastering the Game of Go without Human Knowledge
https://deepmind.com/documents/119/agz_unformatted_nature.pdf

Andrej Karpathy : Pong from pixels
http://karpathy.github.io/2016/05/31/rl/

Maxim Lapan : Deep reinforcement learning
https://www.amazon.com/Practical-Reinforcement-Learning-Maxim-Lapan/dp/1788834240

Mohit Sewak : Practical Convolutional Neural
Networks
https://www.amazon.com/Practical-Convolutional-Neural-Networks-Implement/dp/1788392302

Densely Connected Convolutional Networks
https://arxiv.org/pdf/1608.06993.pdf

Q&A

michal chovanec (michal.nand@gmail.com)

www.youtube.com/channel/UCzVvP2ou8v3afNiVrPAHQGg

github https://github.com/michalnand