COMP 3200
Artificial Intelligence

Lecture 22
Deep Neural Networks
Deep Learning

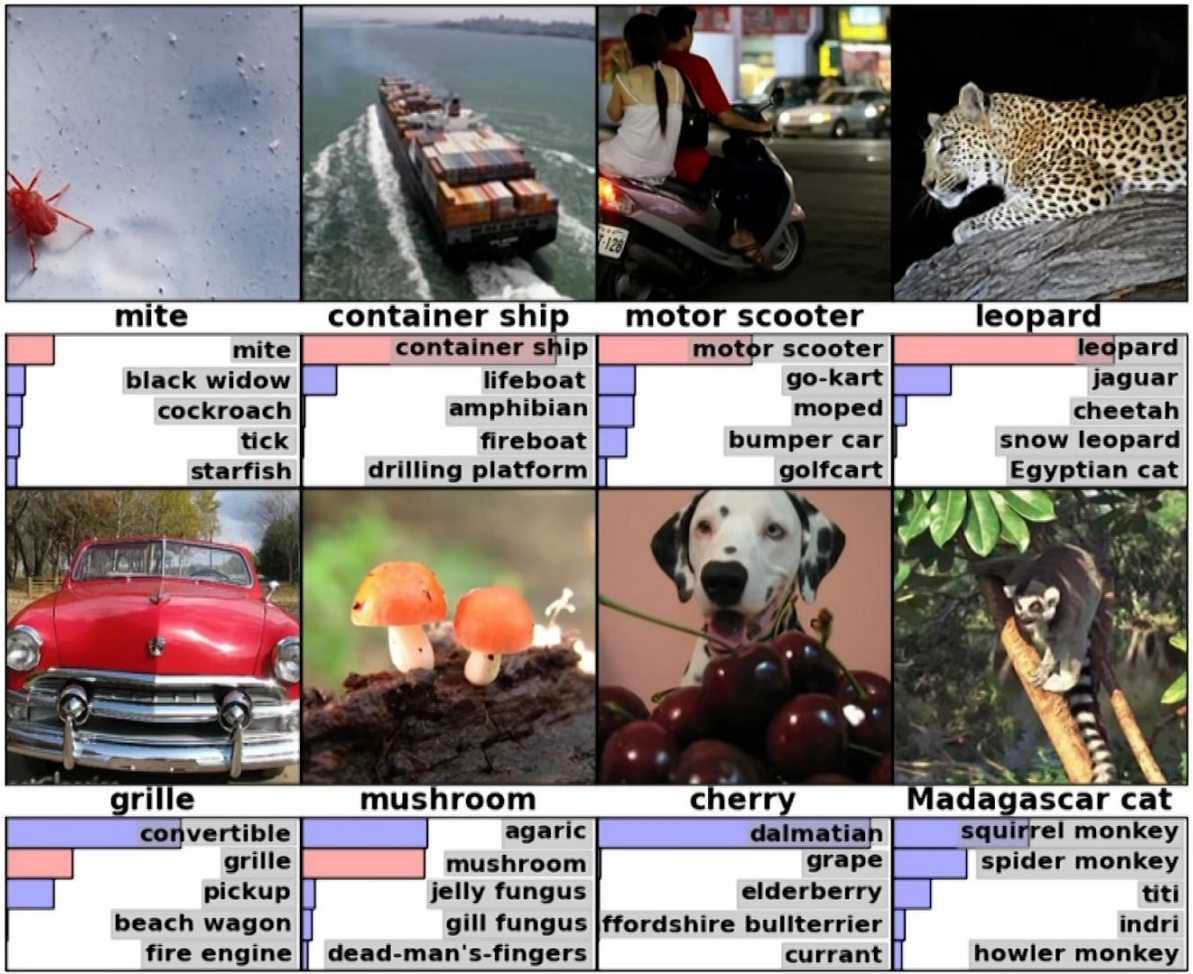
Resources

- The Deep Learning Book
  https://www.deeplearningbook.org/

- Dive into Deep Learning
  https://d2l.ai/

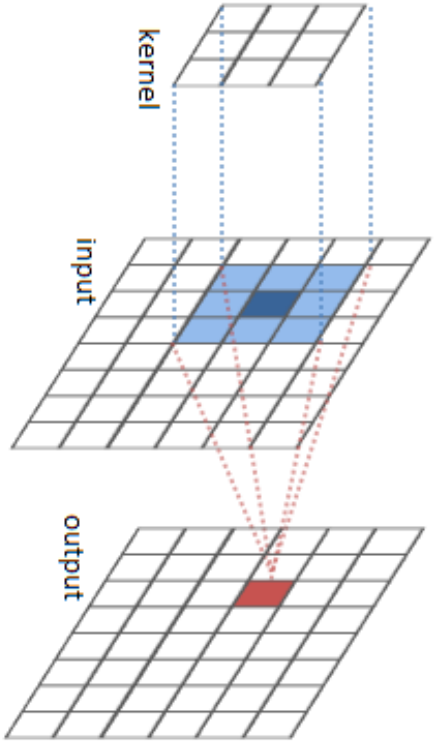
Deep Neural Networks

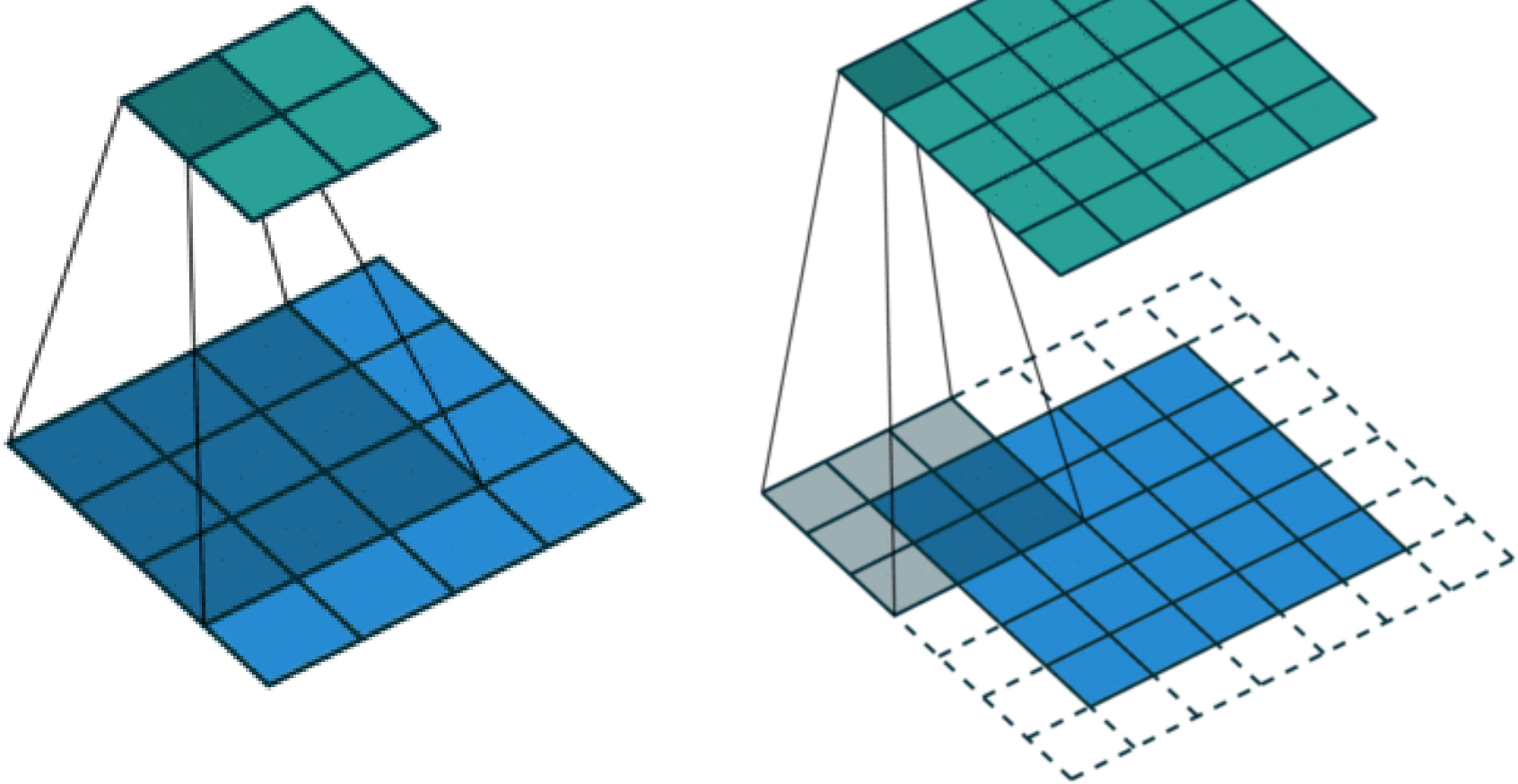
- Neural networks that have many hidden layers, and extra processing
- Have been popular since 2012 (Hinton)
- Rise of GPU computing power has made them feasible (floating point calcs)
- Combine multiple techniques which are effective in specific domains

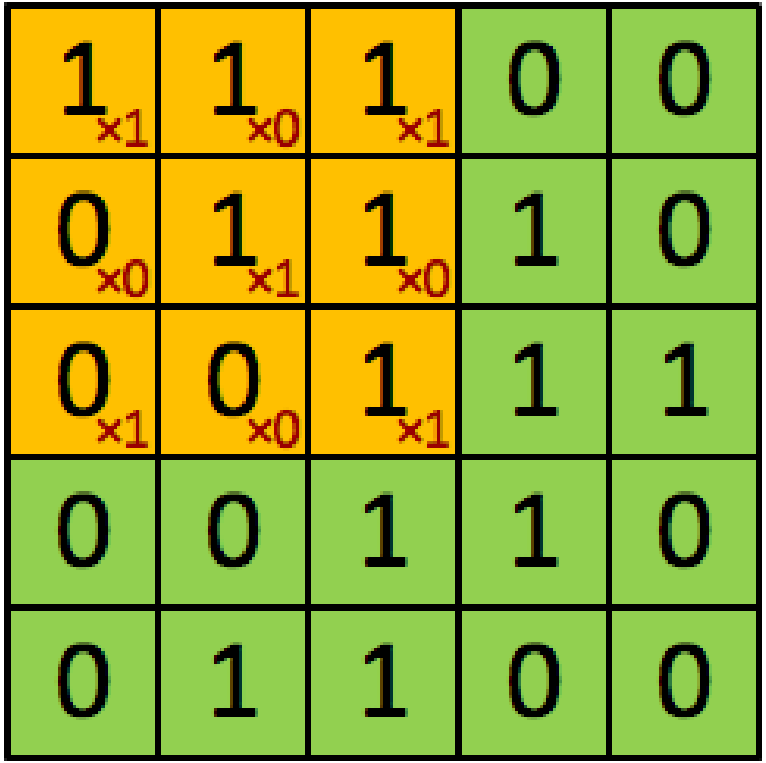

mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

Convolutions

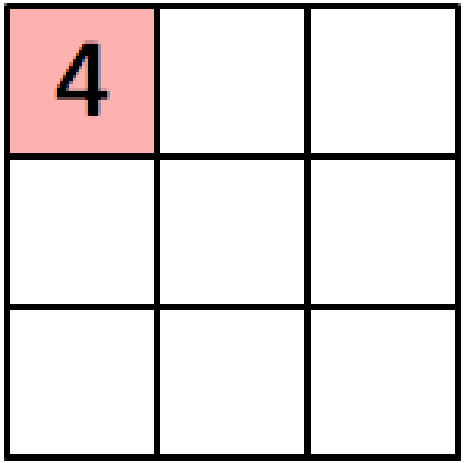
- Have some input image
- Have a process that looks at say, a 3x3 pixel area and produces an output
- That area called a Kernel







Image



Convolved Feature

Edge Detector

- Kernel detects change
- Change = edges
- Apply kernel to img
- Output image is the ‘edge image’
- CNN use this to group and detect features

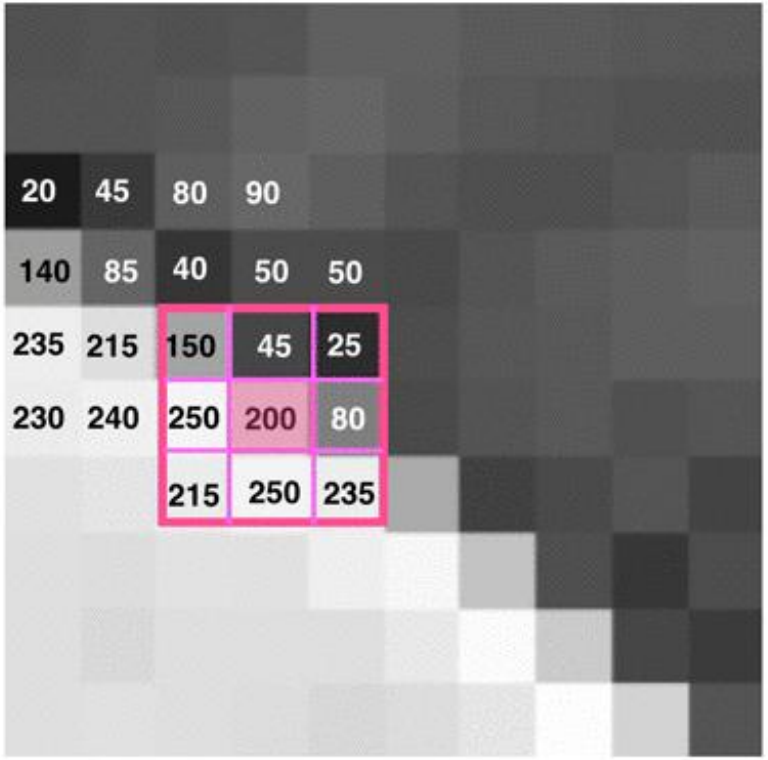
-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

0	-1	0
-1	4	-1
0	-1	0



https://cezannec.github.io/Convolutional_Neural_Networks/

0	-1	0
-1	4	-1
0	-1	0

K

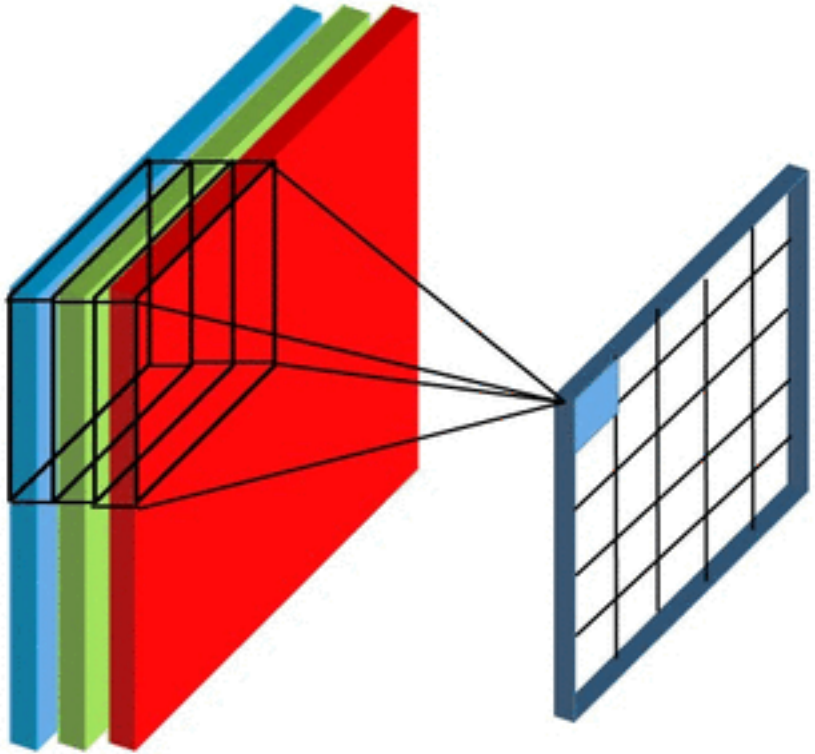
F(x, y)

$$K * F(x,y) = \textit{filtered, output image}$$

filtered image

convolutional kernel
(image filter)

input image



Max Pooling

- Output from convolution produces another matrix
- Now take maximums from areas of that matrix
- This forms another matrix, the process is called pooling

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

20	30
112	37

maxpool

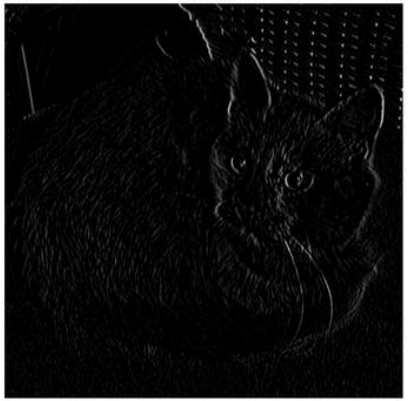
- filter/patch size = 2x2
- stride = 2

Repeating the Process

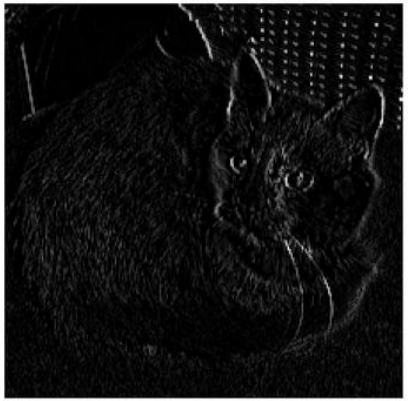
- Convolution -> Max Pooling is repeated some number of times
- We can then optionally run the result of that output through another convolution, and another pooling
- The final output is run through a fully connected neural network

Max Pool Example

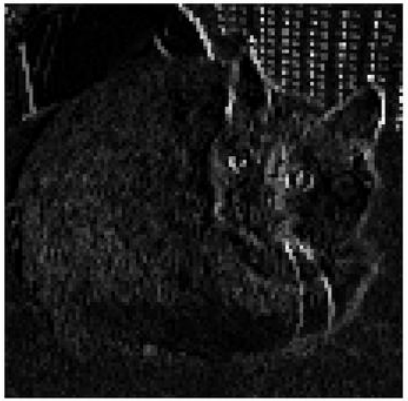
(444, 448)



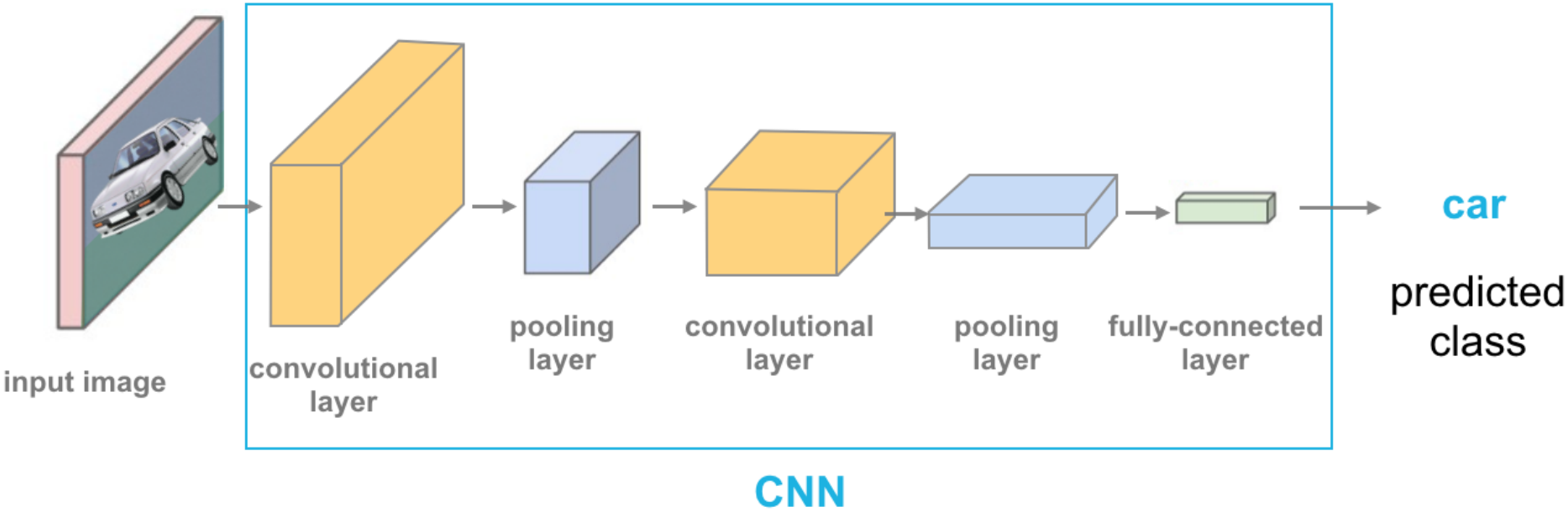
(222, 224)



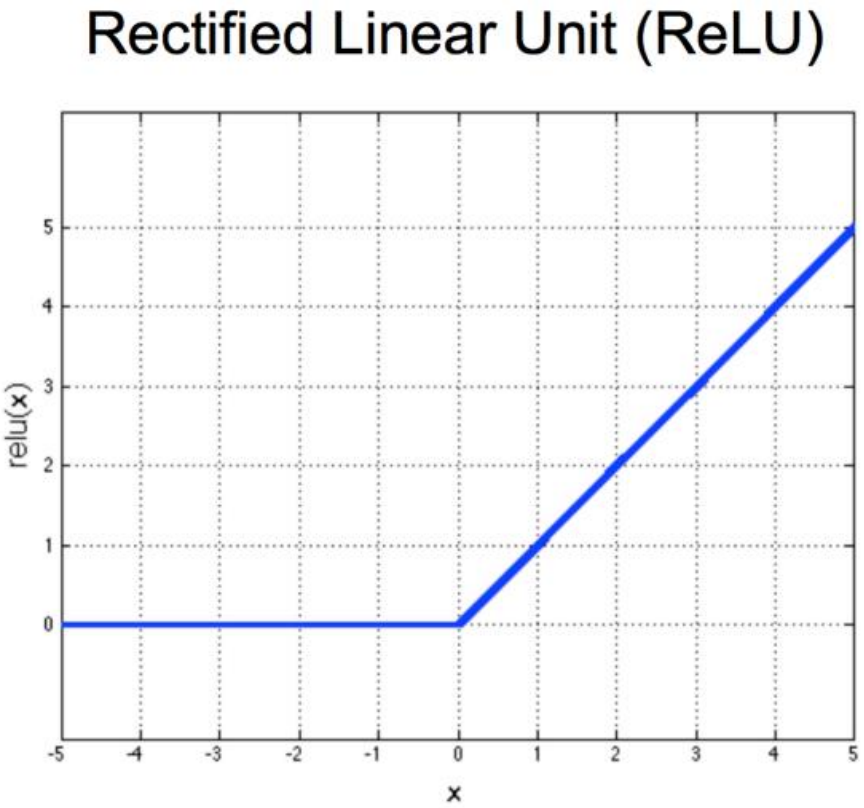
(111, 112)



(55, 56)

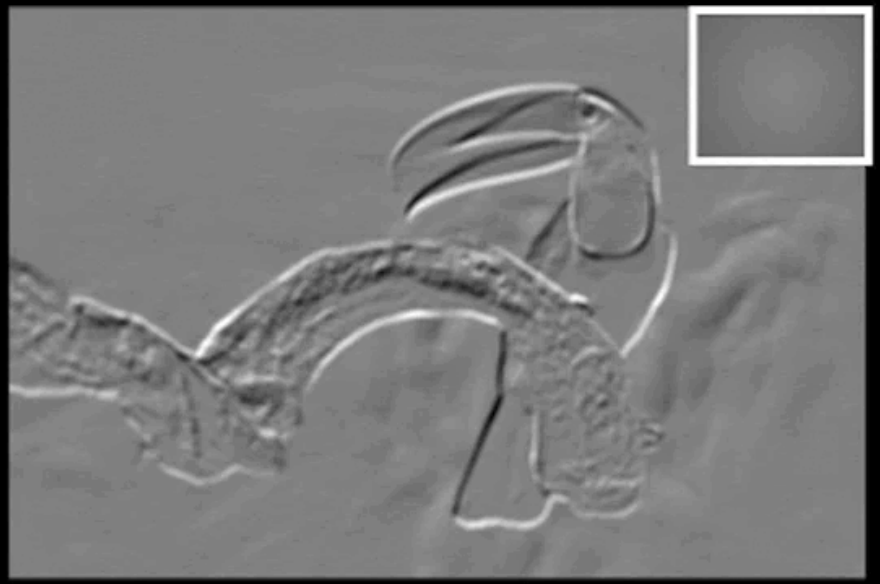

input image
convolutional layer
pooling layer
convolutional layer
pooling layer
fully-connected layer
car
predicted class
CNN

$$\text{relu}(x) \begin{cases} 0, \text{ if } x < 0 \\ x, \text{ if } x >= 0 \end{cases}$$

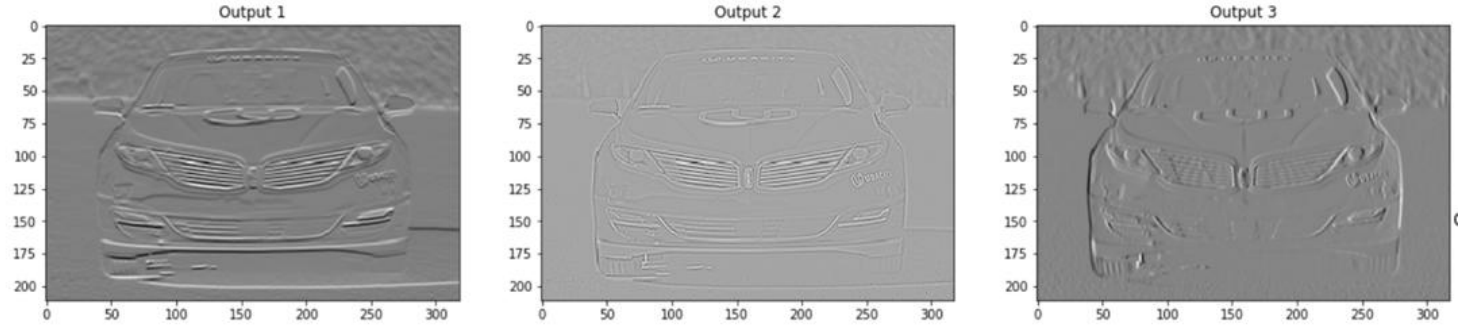


Turns all negative values into 0 (black in image)

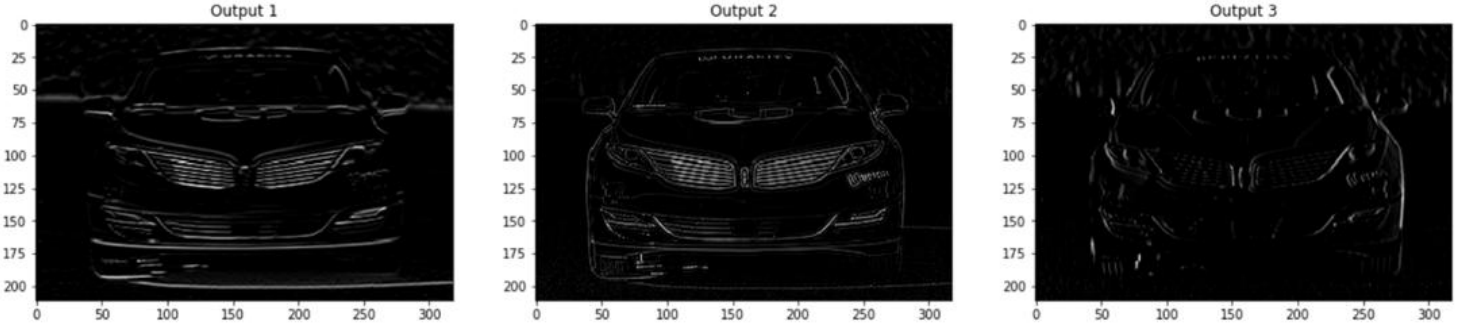
Input Feature Map



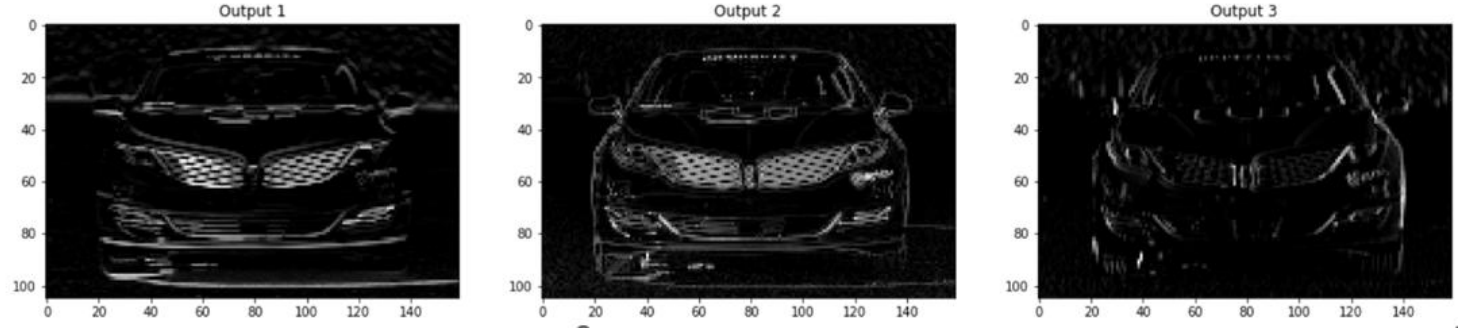
convolutional layer



\+ ReLu (activation)



maxpooling layer



VGG-16 CNN Architecture

Conv-1
Conv-2
Conv-3
Conv-4
Conv-5
FC-6
FC-7
FC-8

224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000

convolution+ReLU
max pooling
fully connected+ReLU

HOW A DEEP NEURAL NETWORK SEES

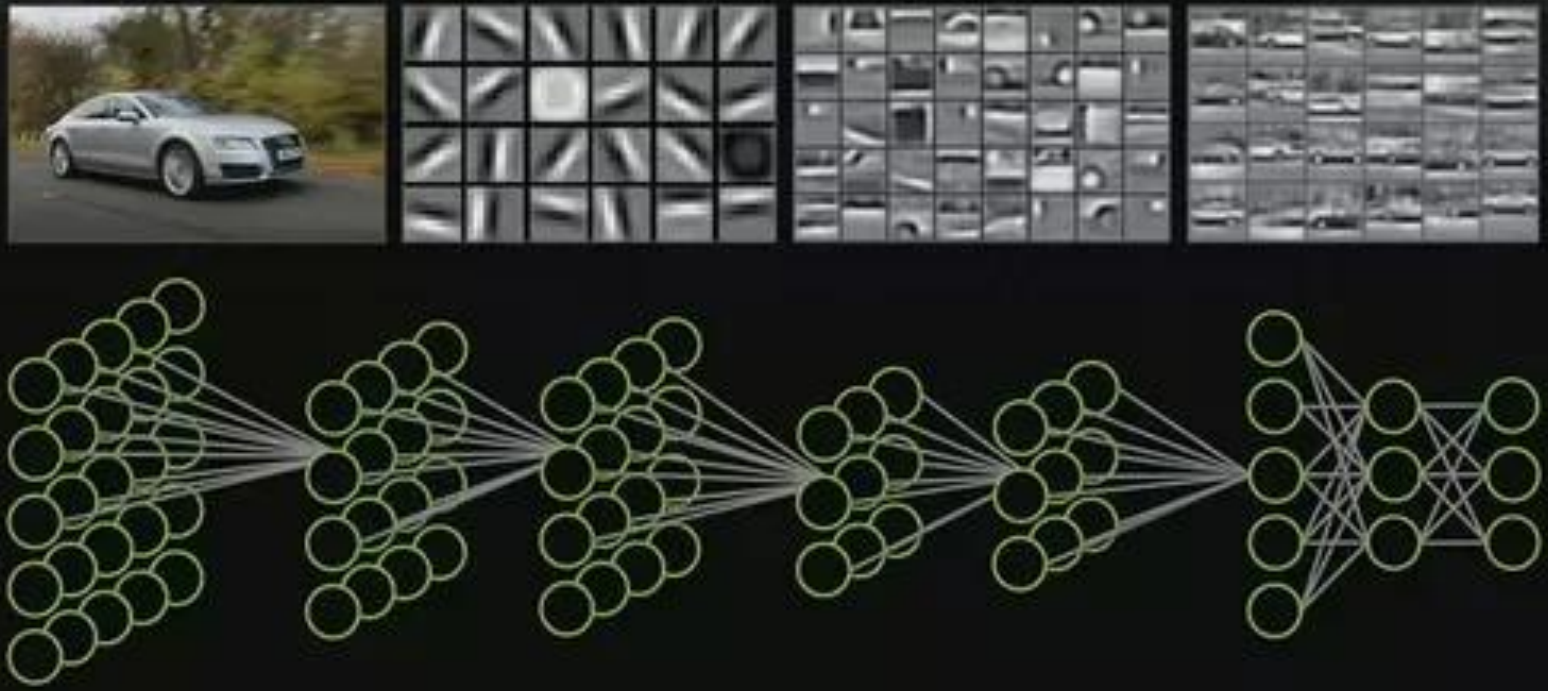
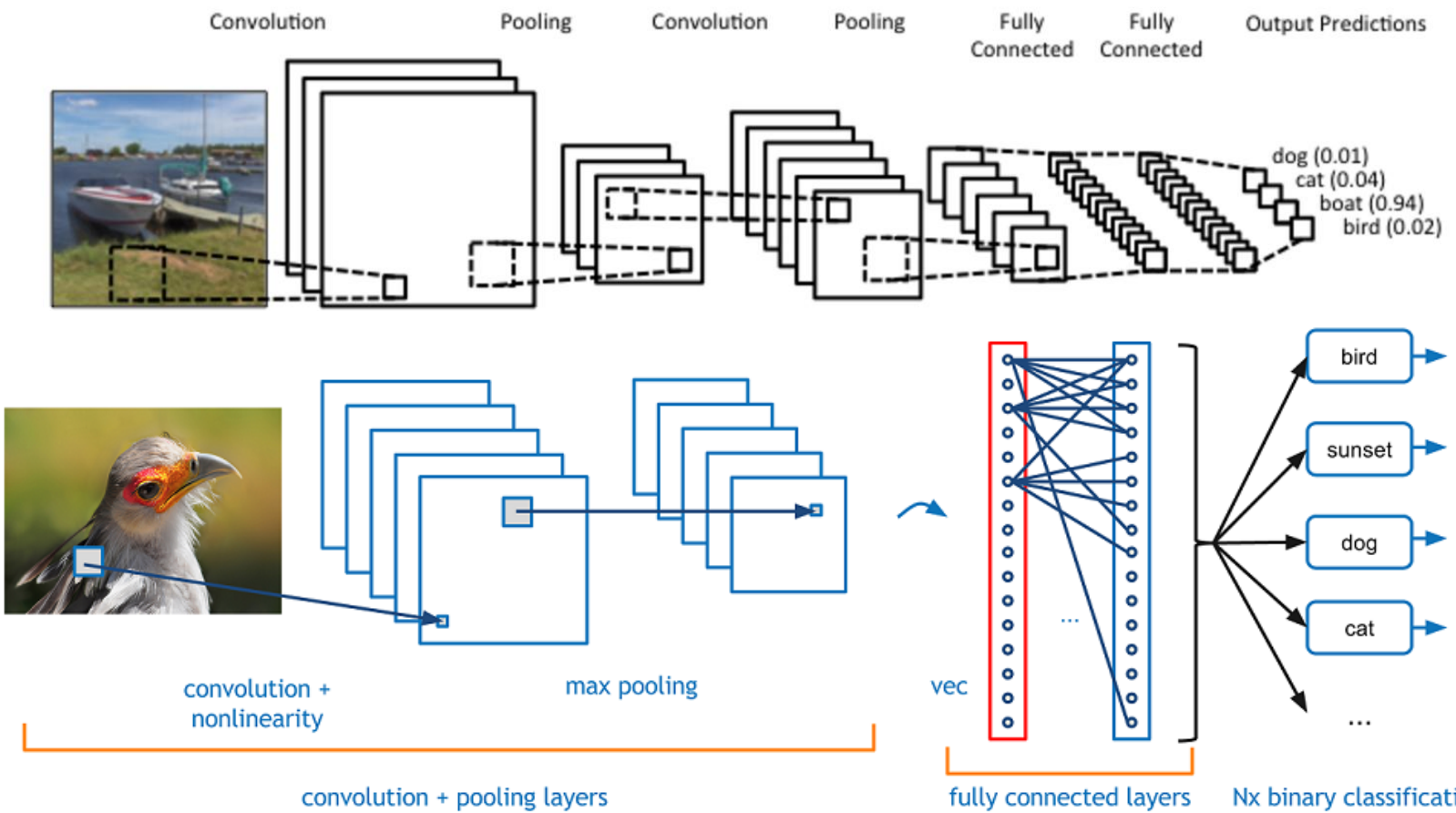
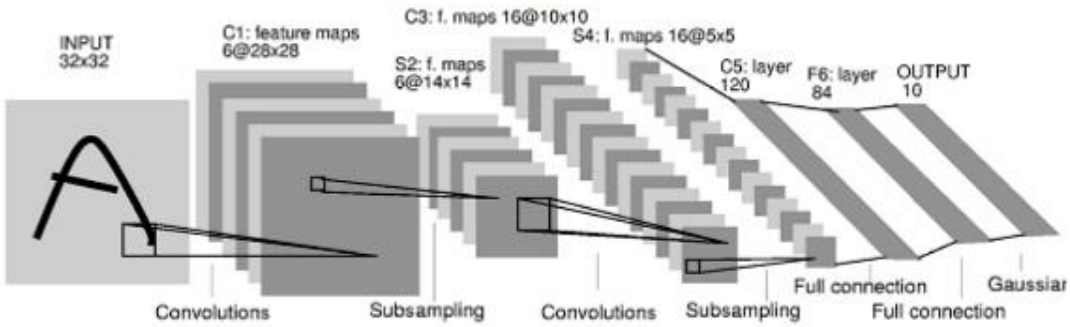


Image source: "Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks" ICML 2009 & Comm. ACM 2011. Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Ng.


Convolution
Pooling
Convolution
Pooling
Fully Connected
Fully Connected
Output Predictions
dog (0.01)
cat (0.04)
boat (0.94)
bird (0.02)
convolution + nonlinearity
max pooling
vec
bird
sunset
dog
cat
...
convolution + pooling layers
fully connected layers
Nx binary classification

1998

LeCun et al.



# of transistors

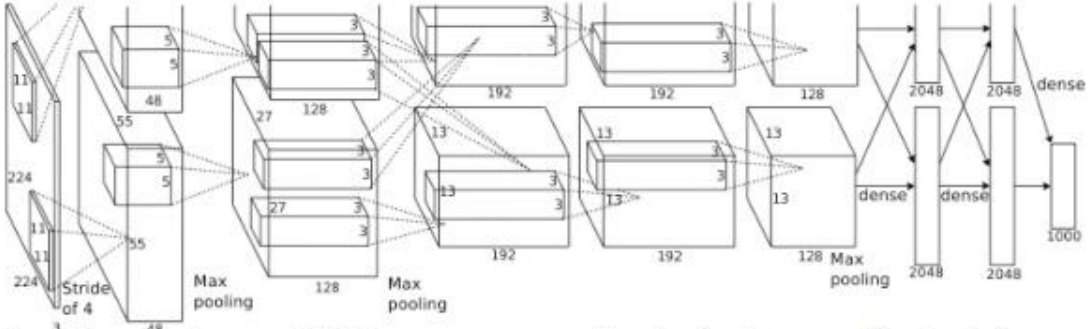
10^6

# of pixels used in training

10^7 NIST

2012

Krizhevsky et al.



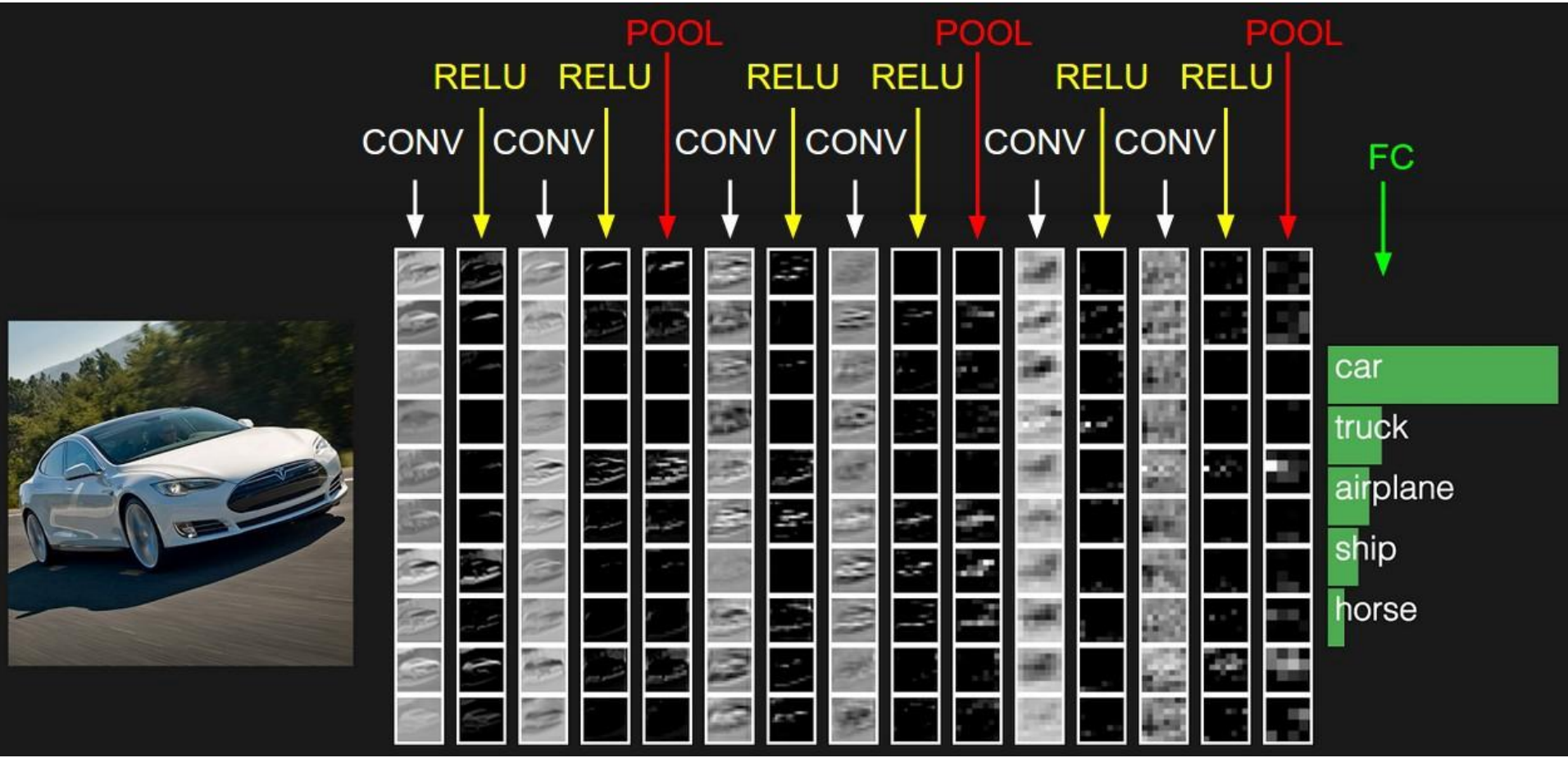
# of transistors

10^9

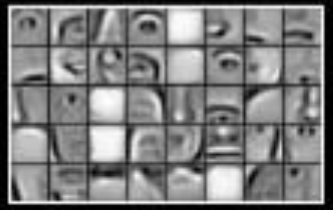
GPUs

# of pixels used in training

10^{14} IMAGENET

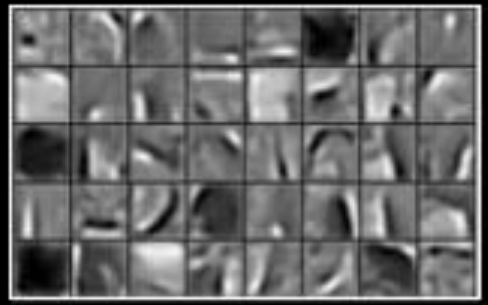

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

Feature Hierarchies: And so on…

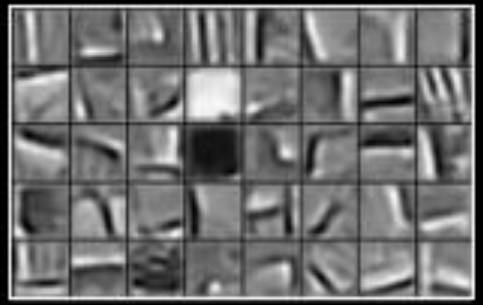


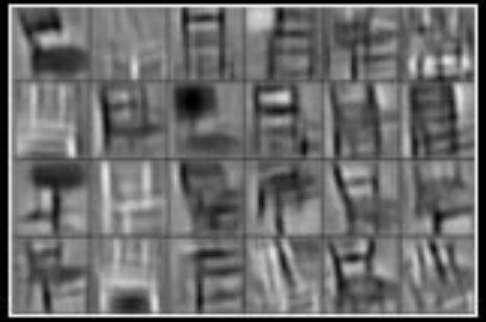
cars

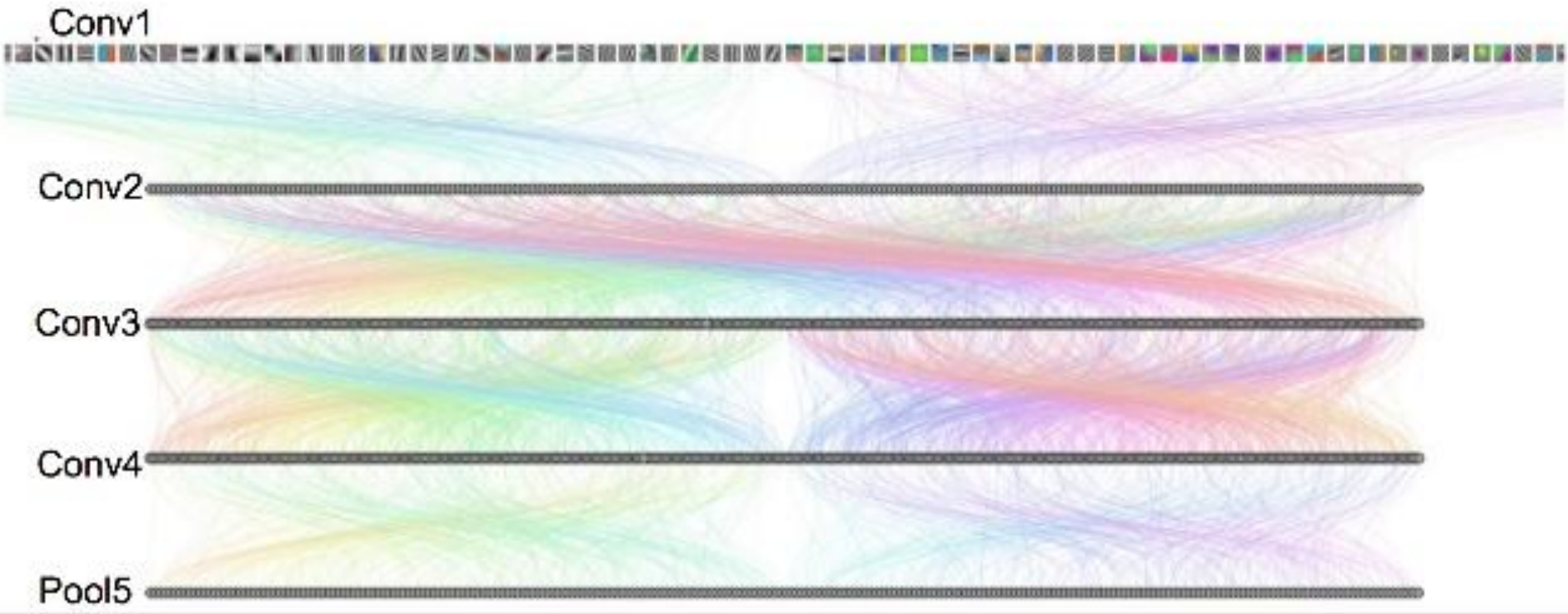
elephants



chairs





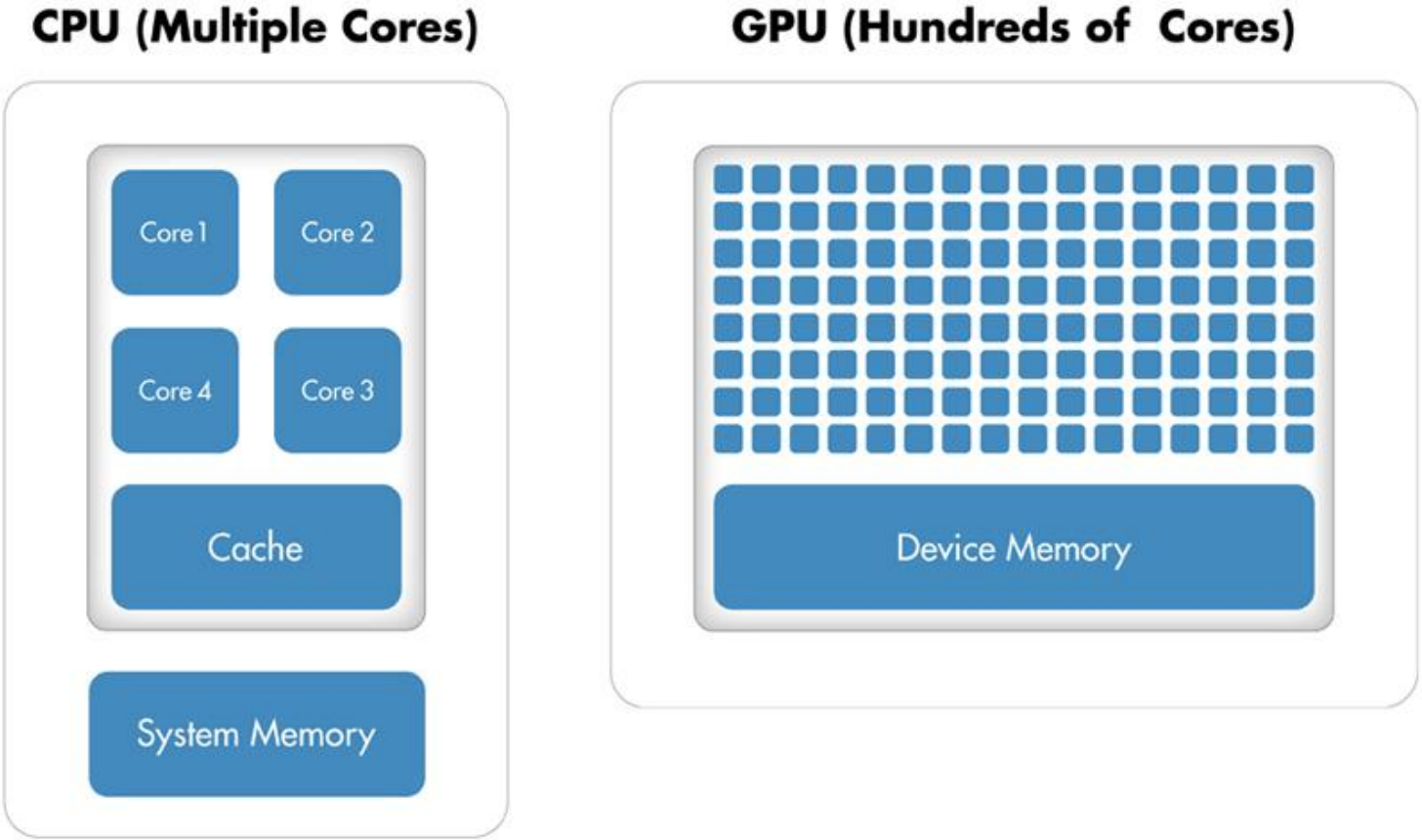


DrawCNN: visualizing the units' connections

Agrawal, et al. Analyzing the performance of multilayer neural networks for object recognition. ECCV, 2014
Szegedy, et al. Intriguing properties of neural networks.arXiv preprint arXiv:1312.6199, 2013
Zeiler, M. et al. Visualizing and Understanding Convolutional Networks, ECCV 2014

Computing Required

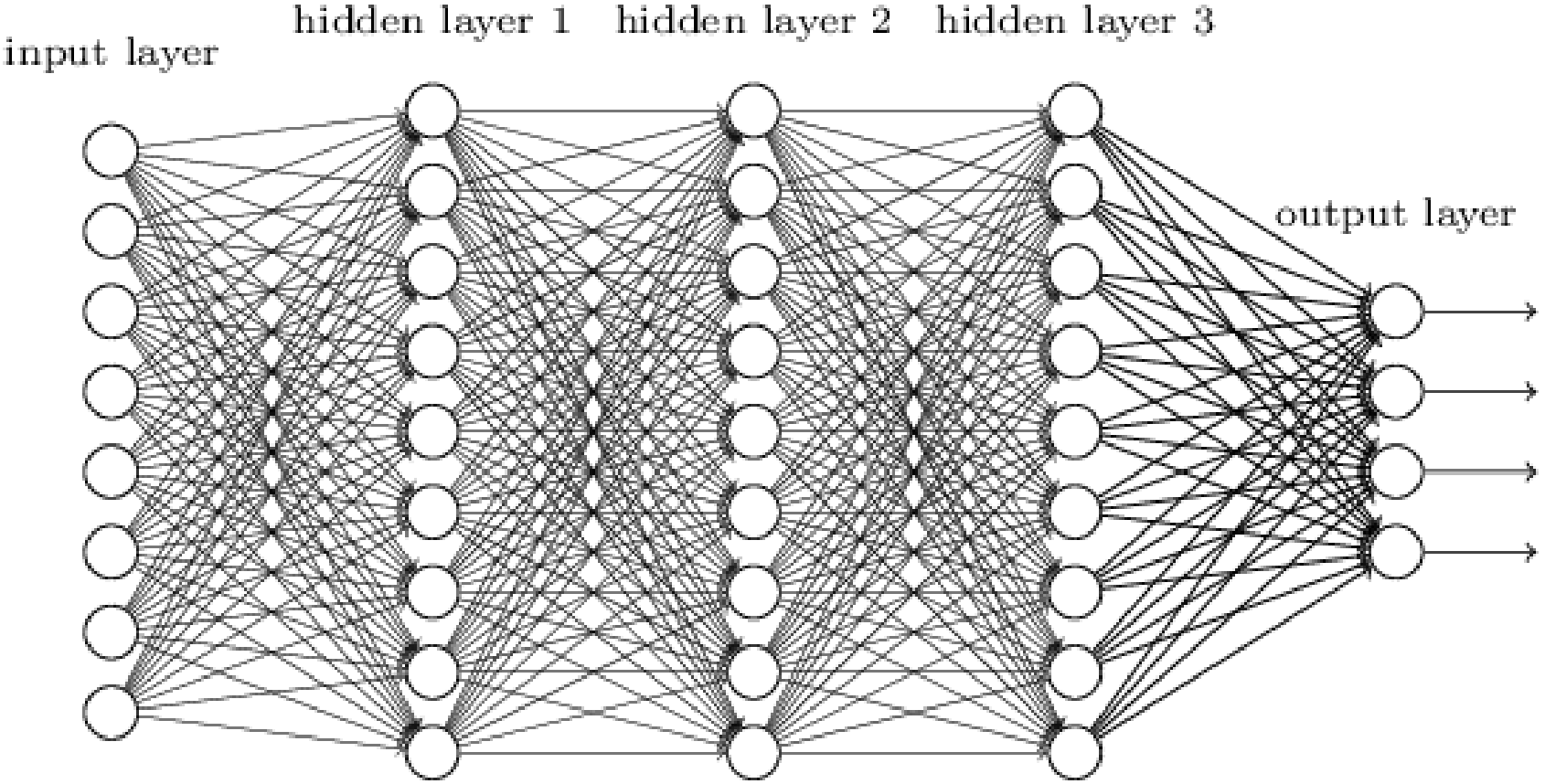
- These techniques are only recently possible due to massive computing power
- GPUs have become ubiquitous and very cheap to manufacture
- Trade single processor speed for massive parallelism
- Neural nets are inherently parallel


CPU (Multiple Cores)
Core 1
Core 2
Core 4
Core 3
Cache
System Memory
GPU (Hundreds of Cores)
Device Memory

GPU	VRAM	CUDA	SU's	Tensor	TMU	RT Cores	SM's	ROP	Mem. Band.
5090	32 GB	21,760	21,760	680	680	170	170	176	1,792 GB/s
4090	24 GB	16,384	16,384	512	512	128	128	176	1,008 GB/s
3090 TI	24 GB	10,752	10,752	336	336	84	84	112	936 GB/s
3090	24 GB	10,496	10,496	328	328	82	82	112	936 GB/s
5090 Laptop	24 GB	10,496	10,496	328	328	84	84	128	936 GB/s
5080	16 GB	10,752	10,752	336	336	84	84	112	960 GB/s
4080 Super	16 GB	10,240	10,240	320	320	80	80	112	716 GB/s
4090 Laptop	16 GB	9,728	9,728	304	304	76	76	112	768 GB/s
4080	16 GB	9,728	9,728	304	304	76	76	112	716 GB/s
5070 TI	16 GB	8,960	8,960	280	280	70	70	128	896 GB/s
4070 TI Super	16 GB	8,448	8,448	264	264	66	66	96	672 GB/s
4070 TI	16 GB	7,680	7,680	240	240	60	60	80	504 GB/s
5080 Laptop	16 GB	7,680	7,680	240	240	60	60	80	811 GB/s
3080 TI Laptop	16 GB	7,424	7,424	232	232	58	58	96	512 GB/s
3080 Laptop	16 GB	6,144	6,144	192	192	48	48	96	448 GB/s

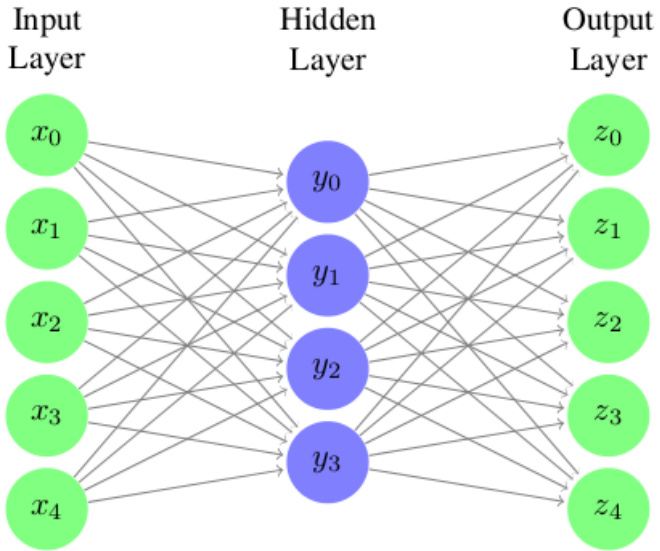
CPU vs GPU

https://www.youtube.com/watch?v=-P28LKWTzrI


input layer
hidden layer 1
hidden layer 2
hidden layer 3
output layer

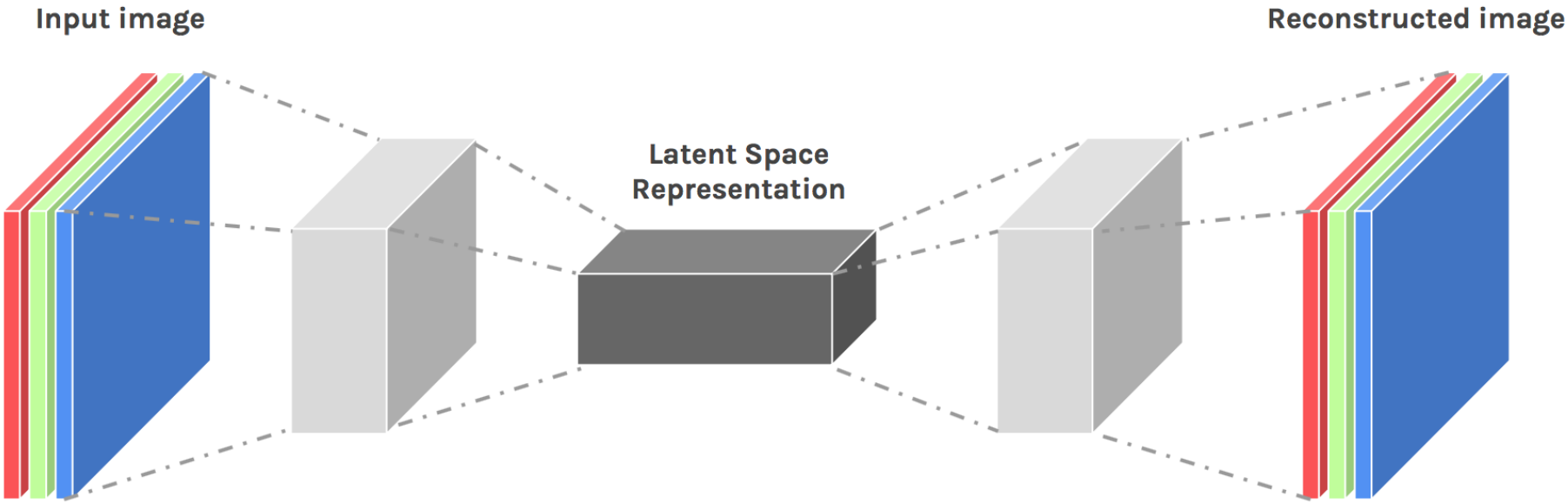
Autoencoding

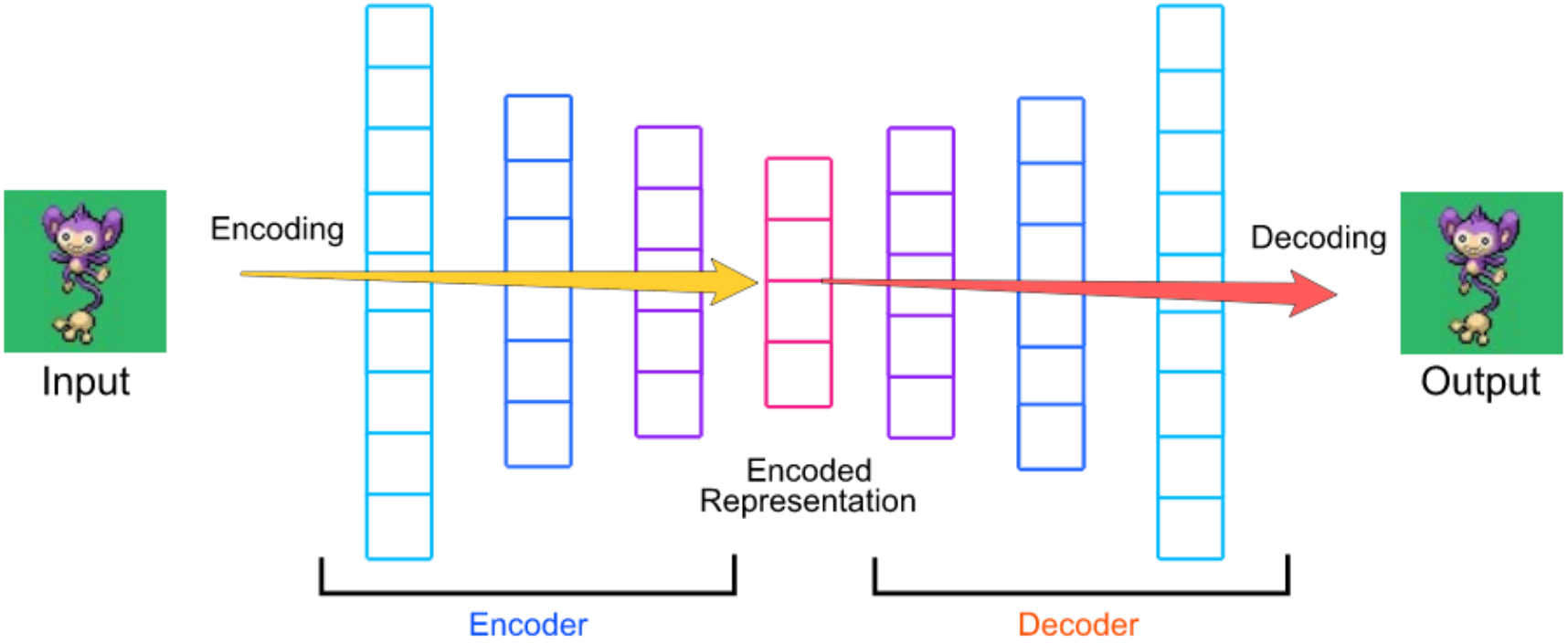
- Inputs x_1 … x_n
- Hidden layer which is smaller than input
- Output layer size of input
- Our target values are x_i
- Finding generalizations of patterns on input

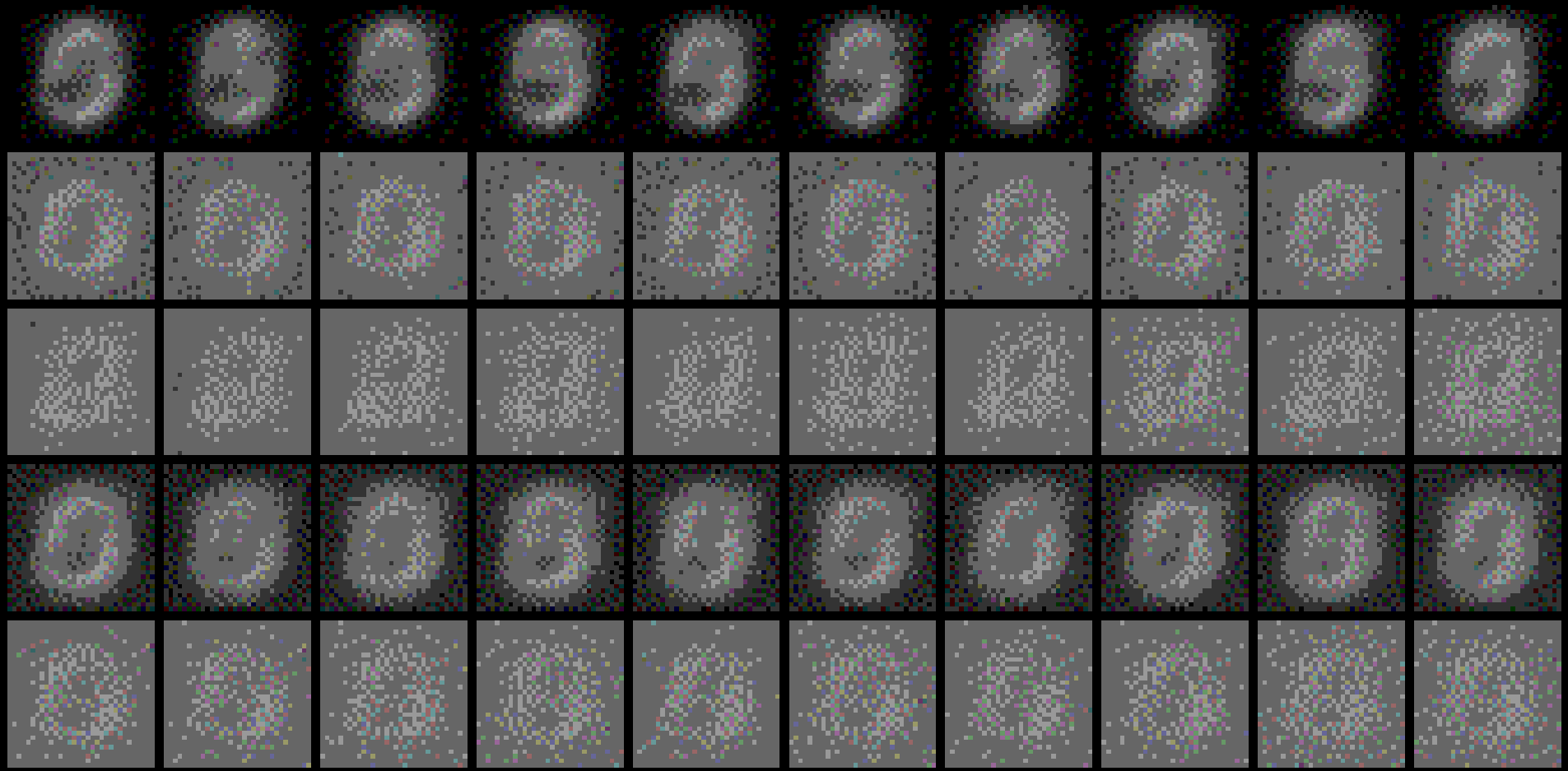


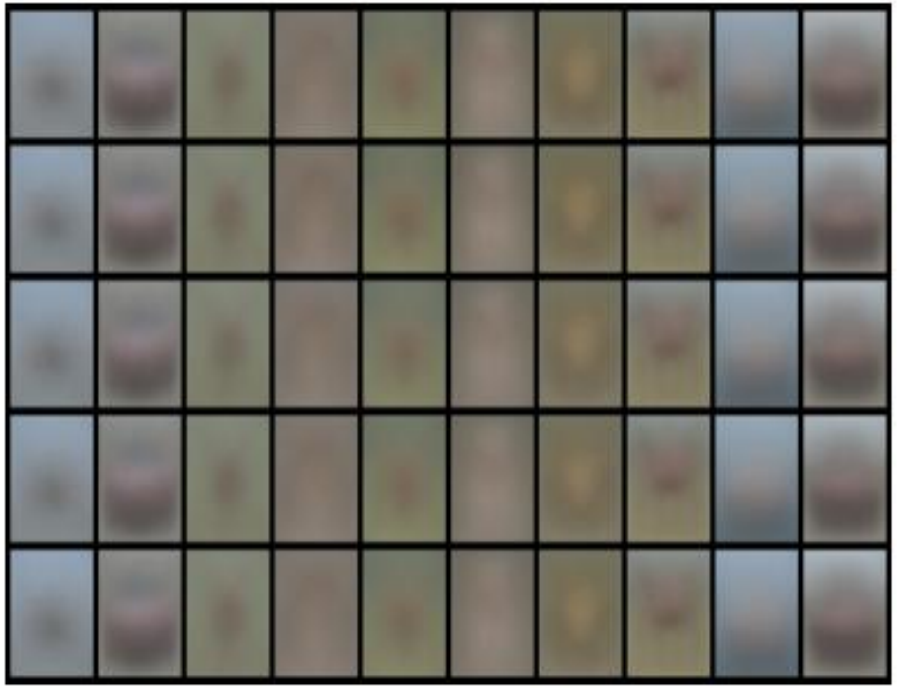
Autoencoding

- Autoencoding “generalizes” inputs
- Can be seen as form of compression
- DNN use autoencoders frequently


Input image
Latent Space Representation
Reconstructed image


Input
Encoding
Encoded
Representation
Decoding
Output
Encoder
Decoder

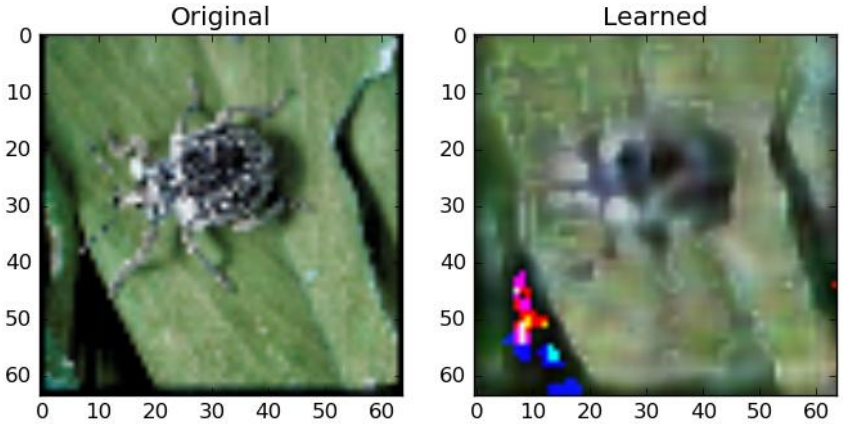


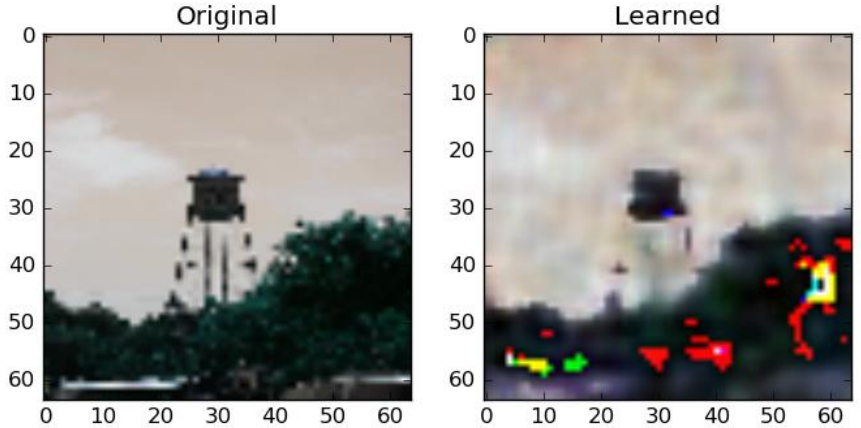


failure on samples of CIFAR 10

failure on reconstruction of pokemon dataset

Autoencoding Examples

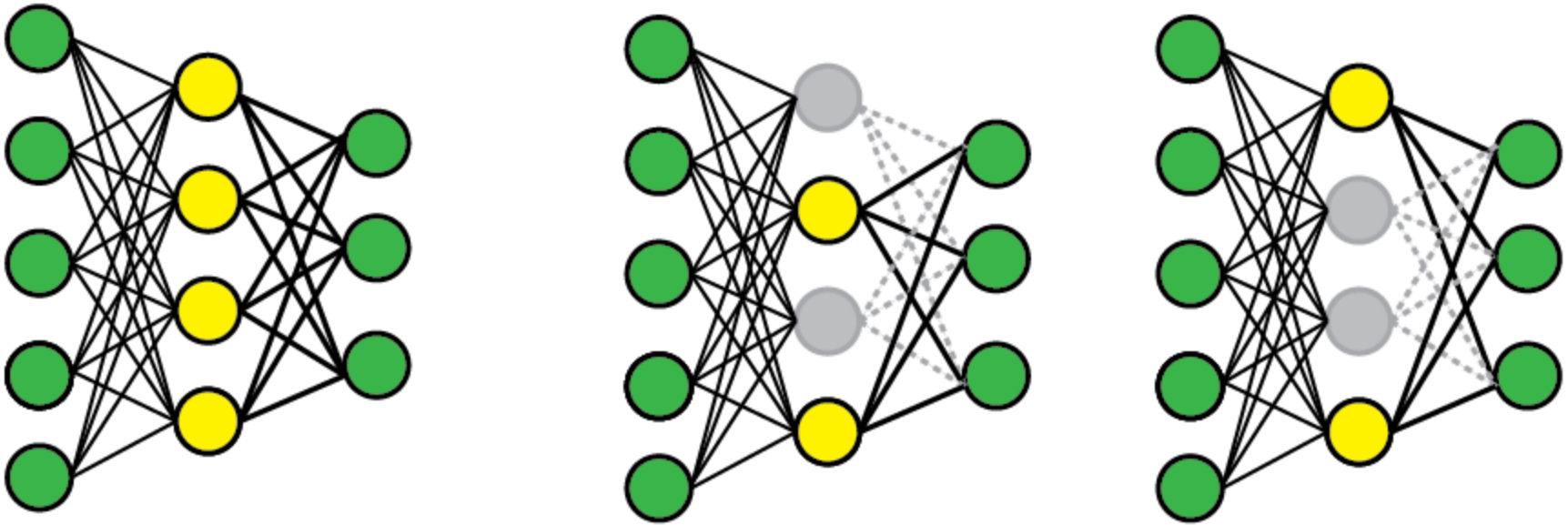




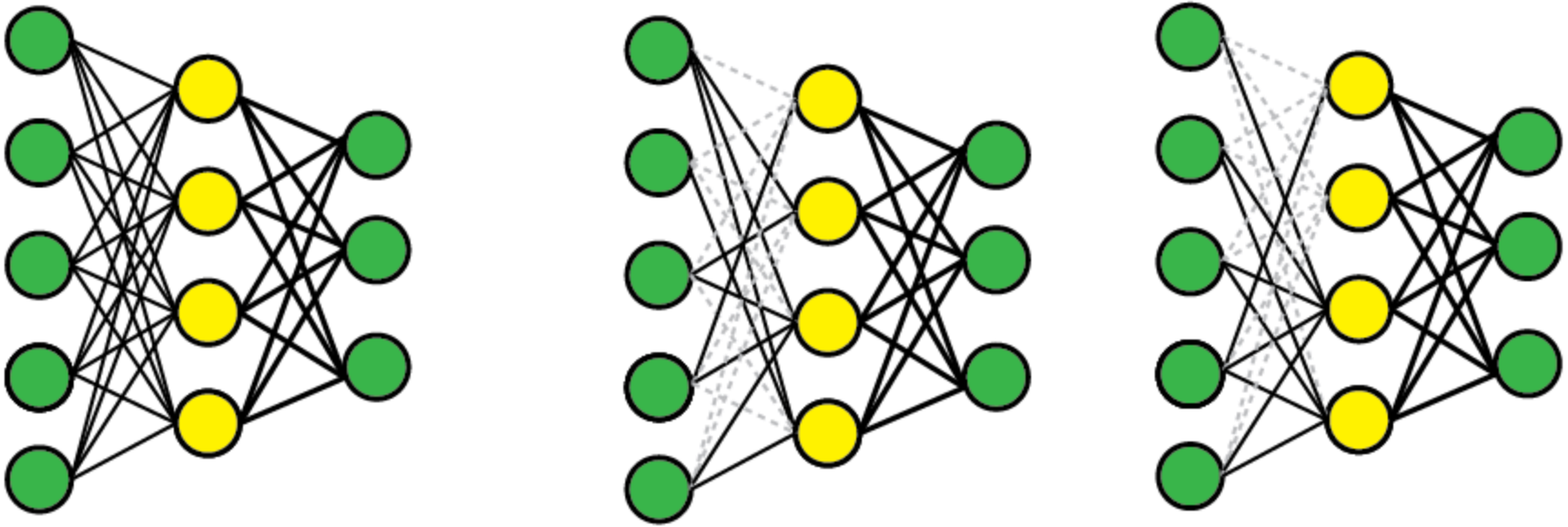
Dropout

- The curse of the neural net is that it can get stuck in a local maxima
- Dropout: If on every iteration we flip a coin for each neuron, and depending on the output we disable the neuron
- Next iteration, drop out a different set
- Prevents getting stuck in local maxima as well as overfitting to training data

Dropout

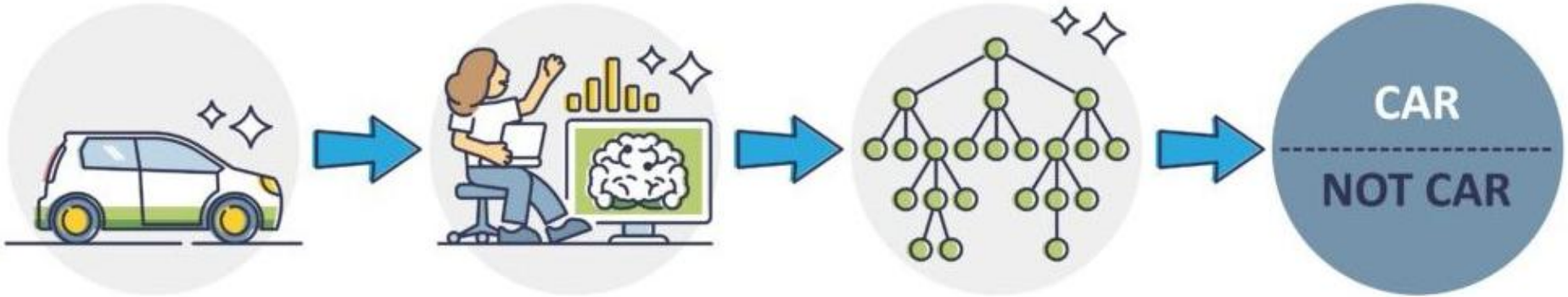


DropConnect (Dropout Generalized)



Deep Learning

MACHINE LEARNING



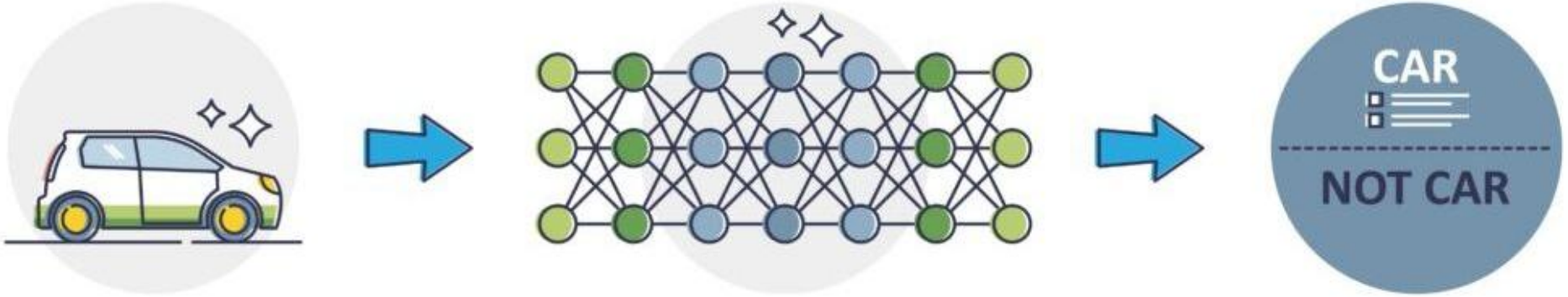
INPUT

FEATURE EXTRACTION

CLASSIFICATION

OUTPUT

DEEP LEARNING



INPUT

FEATURE EXTRACTION + CLASSIFICATION

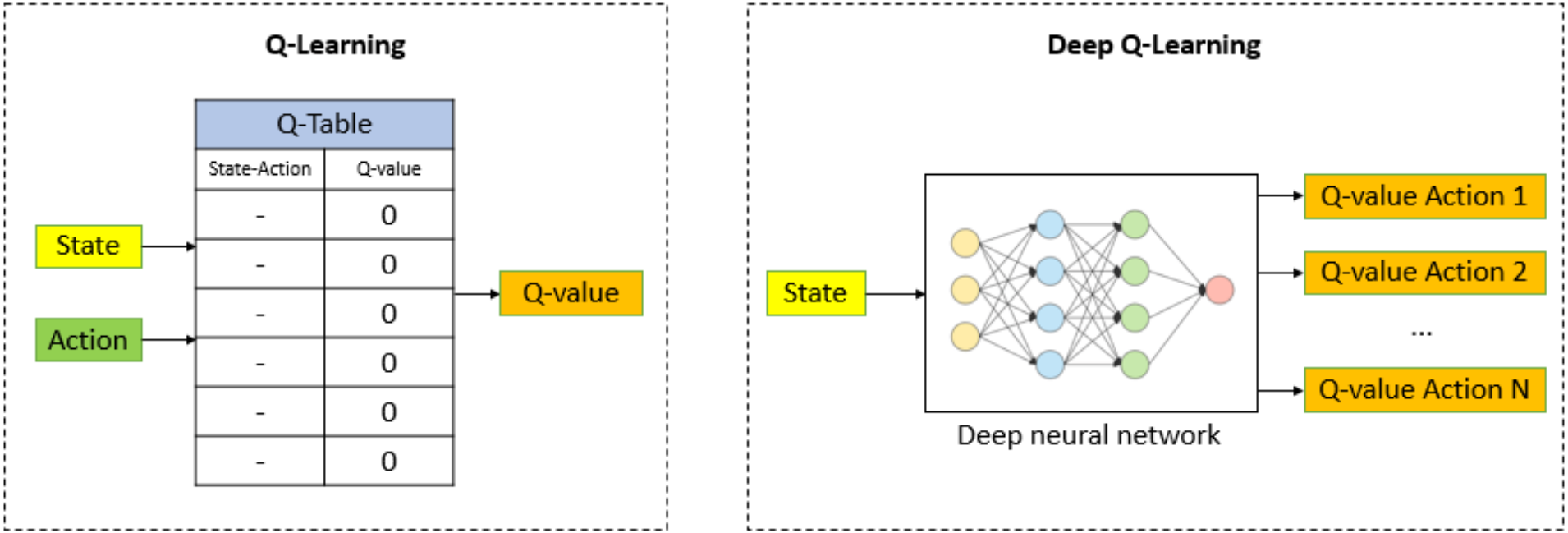
OUTPUT

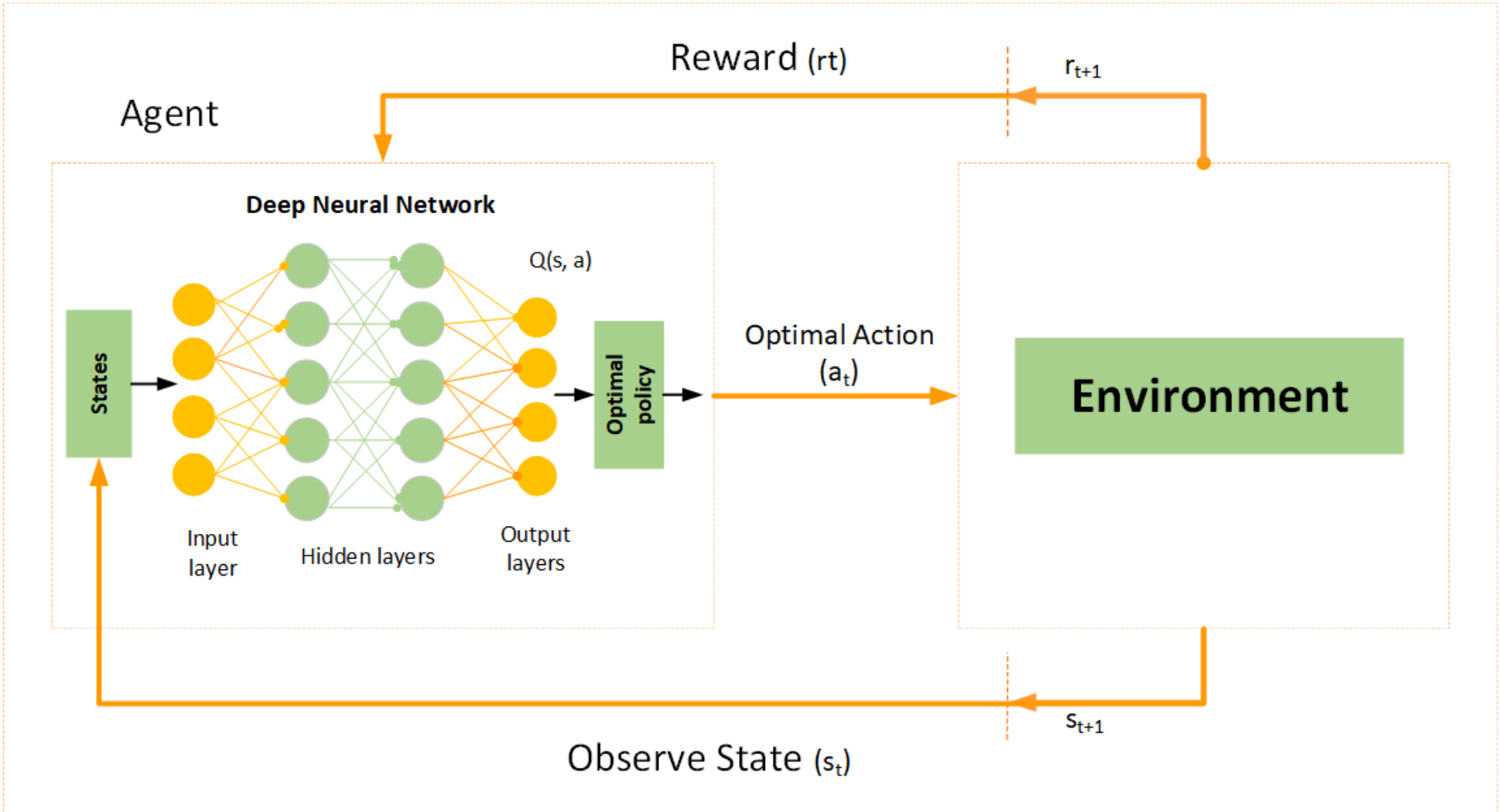
Deep Reinforcement Learning

- Tabular RL only goes so far
  - Discrete state representation
  - Low dimensional data
- We require function approximation (FA)
  - Q(s,a) = result of function, not table lookup
  - Deep neural nets are good at FA
- Enter Deep Q-Networks

Deep Q-Networks (DQN)

- Uses deep CNN for function approximation to approximate optimal Q*(s, a)
- Uses a modified version of Q-learning that changes when some updates are performed, to encourage convergence
- Paper: *Human-level control through deep reinforcement learning*
  - https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf


Q-Learning
Q-Table
State-Action
Q-value
-
0
-
0
-
0
-
0
-
0
-
0
State
Action
Q-value
Deep Q-Learning
State
Deep neural network
Q-value Action 1
Q-value Action 2
...
Q-value Action N


Reward (rt)
r_{t+1}
Agent
Deep Neural Network
Q(s, a)
States
Optimal policy
Optimal Action (a_t)
Environment
Input layer
Hidden layers
Output layers
s_{t+1}
Observe State (s_t)

parameterize an approximate value function $Q(s,a;\theta_i)$ using the deep convolutional neural network shown in Fig. 1, in which θ_i are the parameters (that is, weights) of the Q-network at iteration i. To perform experience replay we store the agent's experiences $e_t = (s_t,a_t,r_t,s_{t+1})$ at each time-step t in a data set $D_t = \{e_1,\ldots,e_t\}$. During learning, we apply Q-learning updates, on samples (or minibatches) of experience $(s,a,r,s') \sim U(D)$, drawn uniformly at random from the pool of stored samples. The Q-learning update at iteration i uses the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r+\gamma \max_{a'} Q(s',a';\theta_i^-) - Q(s,a;\theta_i)\right)^2\right]$$

in which γ is the discount factor determining the agent's horizon, θ_i are the parameters of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i. The target network parameters θ_i^- are only updated with the Q-network parameters (θ_i) every C steps and are held fixed between individual updates (see

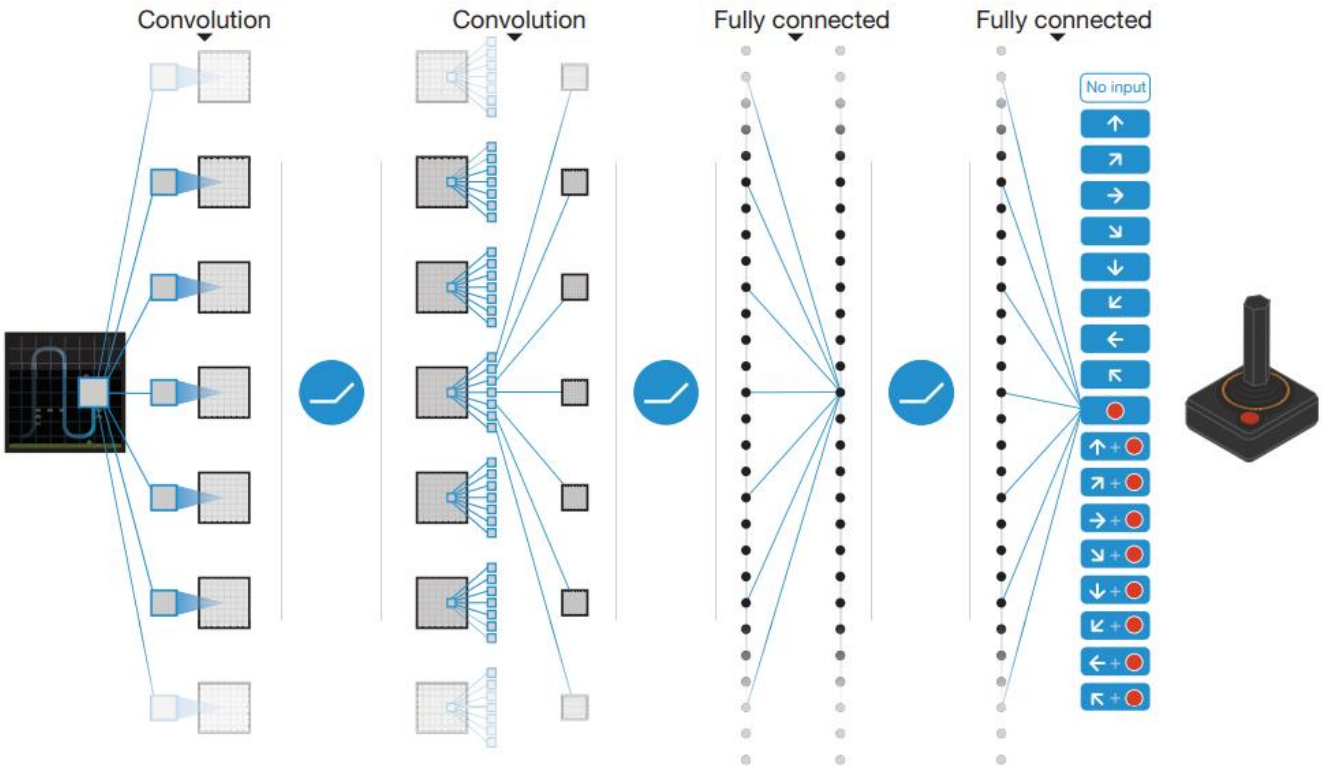
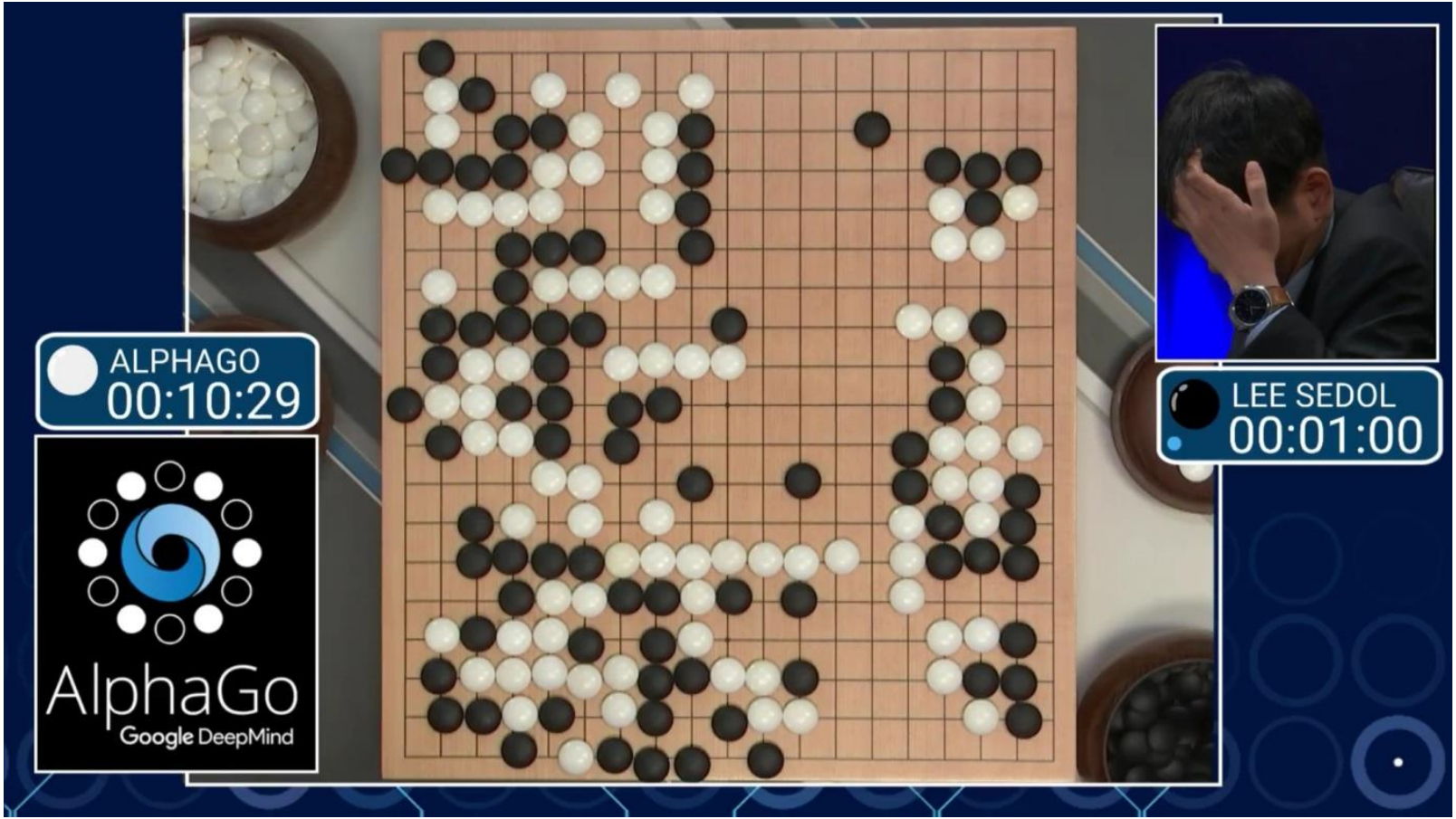


Figure 1 | Schematic illustration of the convolutional neural network. The details of the architecture are explained in the Methods. The input to the neural network consists of an $84 \times 84 \times 4$ image produced by the preprocessing map ϕ, followed by three convolutional layers (note: snaking blue line symbolizes sliding of each filter across input image) and two fully connected layers with a single output for each valid action. Each hidden layer is followed by a rectifier nonlinearity (that is, $\max(0,x)$).


ALPHAGO
00:10:29
AlphaGo
Google DeepMind
LEE SEDOL
00:01:00

AlphaGo

- Combined Deep Learning with Heuristic Search (MCTS)

- Two Networks were trained:
  - Value Network (how good is state S)
  - Policy Network (what move to do at state S)

Learning from Replays

- DeepMind had thousands of professional human games to learn from
- Policy Network
  - Input = State
  - Target = Move the expert human performed
- Value Network
  - Input = State
  - Target = Who won the game from that state

Learning from Self Play

- Once it had learned all it could from humans, it played against itself
- Continued to learn, and get stronger
- Now it learns from a stronger player each time it plays a new game
- Also key: throw in some random moves to learn about states you may not visit

AlphaGo Zero

- Learned entirely from self play
- Use MCTS to play games against itself
- Learned which moves were good from self play by using Deep RL
- Converged faster, used less computation than the original AlphaGo
- Worked for Chess as well

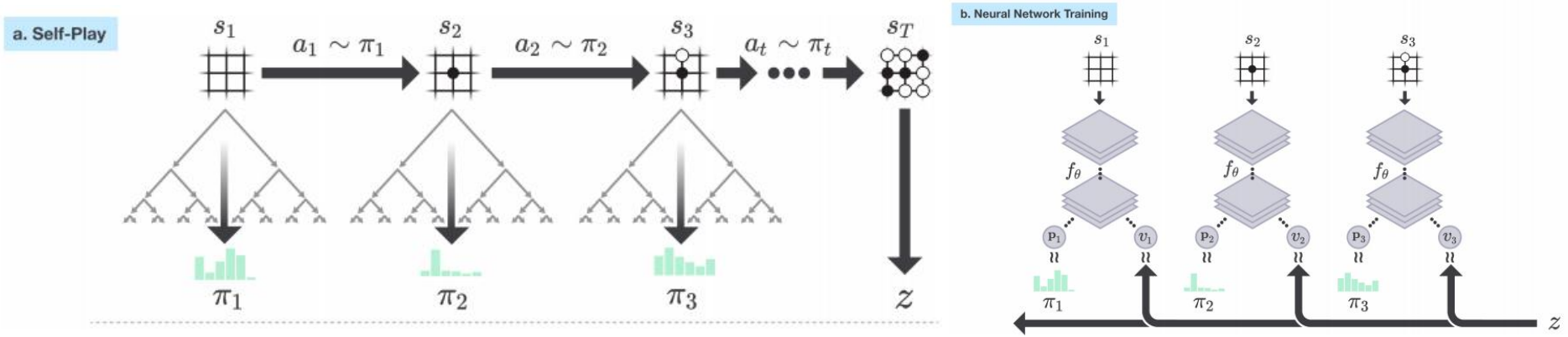


Figure 1: **Self-play reinforcement learning in *AlphaGo Zero*. a** The program plays a game $s_1, ..., s_T$ against itself. In each position s_t, a Monte-Carlo tree search (MCTS) α_θ is executed (see Figure 2) using the latest neural network f_θ. Moves are selected according to the search probabilities computed by the MCTS, $a_t \sim \boldsymbol{\pi}_t$. The terminal position s_T is scored according to the rules of the game to compute the game winner z. **b** Neural network training in *AlphaGo Zero*. The neural network takes the raw board position s_t as its input, passes it through many convolutional layers with parameters θ, and outputs both a vector $\mathbf{p_t}$, representing a probability distribution over moves, and a scalar value v_t, representing the probability of the current player winning in position s_t. The neural network parameters θ are updated so as to maximise the similarity of the policy vector $\mathbf{p_t}$ to the search probabilities $\boldsymbol{\pi_t}$, and to minimise the error between the predicted winner v_t and the game winner z (see Equation 1). The new parameters are used in the next iteration of self-play **a**.

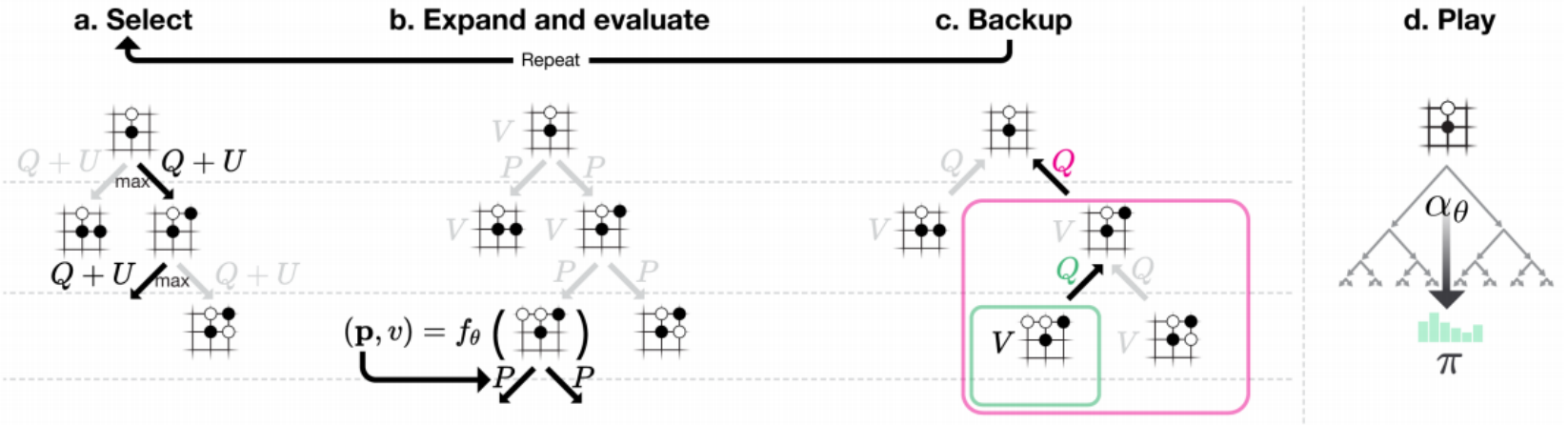


Figure 2: **Monte-Carlo tree search in *AlphaGo Zero*.** **a** Each simulation traverses the tree by selecting the edge with maximum action-value Q, plus an upper confidence bound U that depends on a stored prior probability P and visit count N for that edge (which is incremented once traversed). **b** The leaf node is expanded and the associated position s is evaluated by the neural network $(P(s,\cdot), V(s)) = f_\theta(s)$; the vector of P values are stored in the outgoing edges from s. **c** Action-values Q are updated to track the mean of all evaluations V in the subtree below that action. **d** Once the search is complete, search probabilities $\boldsymbol{\pi}$ are returned, proportional to $N^{1/\tau}$, where N is the visit count of each move from the root state and τ is a parameter controlling temperature.

Heuristic Search

- Go has a large branching factor (300)
- Humans have good abstractions of what moves to perform at a state
- AlphaGo learned a policy network
- Now it can narrow the heuristic search to only consider moves in the policy network
- Search also needs a heuristic function to determine how good a state is (value network)

https://www.deepstack.ai/

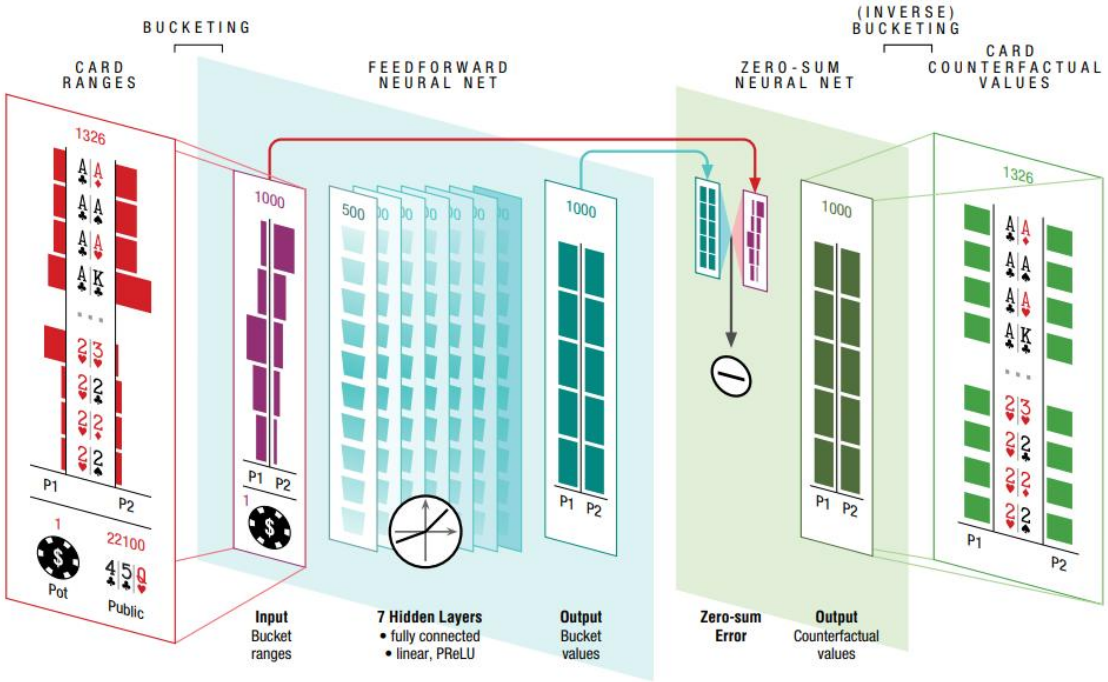
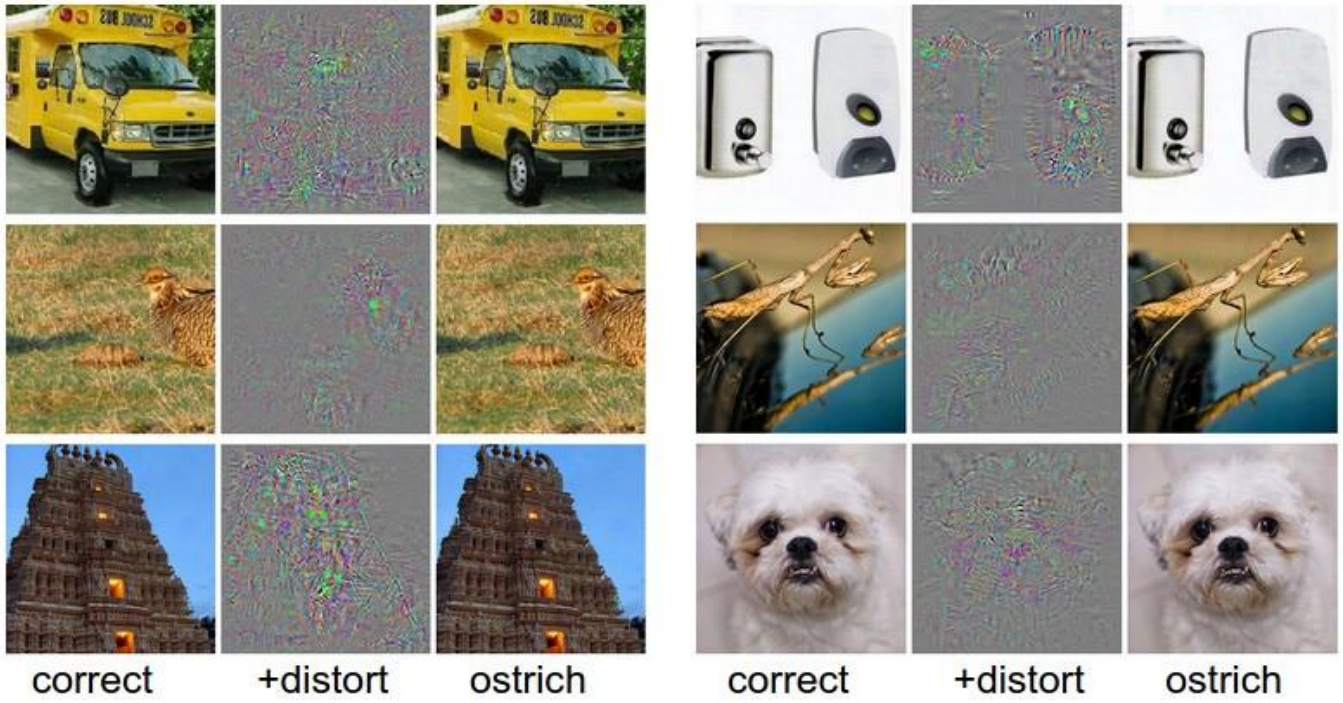


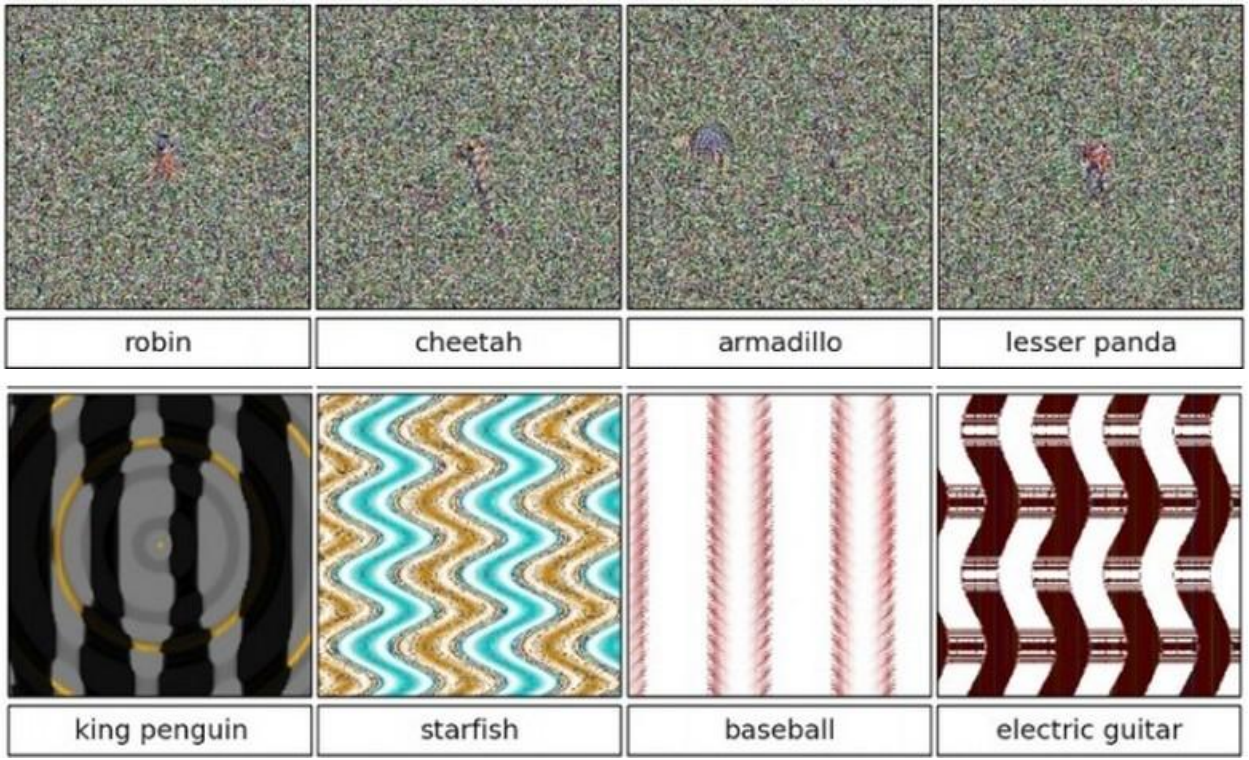
Figure 3: **Deep counterfactual value network.** The inputs to the network are the pot size, public cards, and the player ranges, which are first processed into hand clusters. The output from the seven fully connected hidden layers is post-processed to guarantee the values satisfy the zero-sum constraint, and then mapped back into a vector of counterfactual values.

What do DNN see?

Tricking DNN



Tricking DNN



Not Human

- Deep Neural Networks are an amazing feat of engineering that produce spectacular results
- They are not magic, they are math
- They are easy to fool, if you try