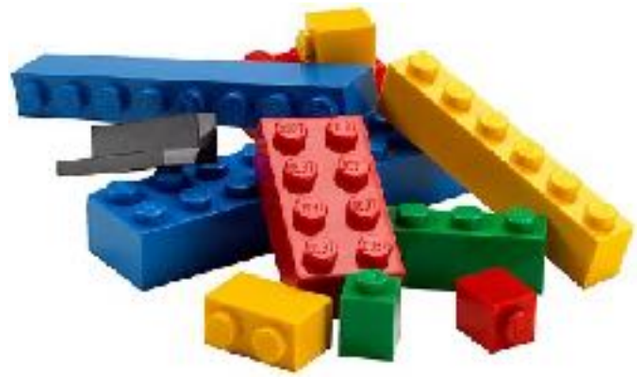
Special Layers

Playing Lego



Example

```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
model = Sequential()

model.add(Convolution2D(filters=64,kernel_size=(2,2),activation= 'relu',border_mode='valid
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(216))
model.add(Dropout(0.5))
model.add(Dense(10,activation = 'softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs=10, batch_size=10)
```

```
/Users/TigerTGV/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:7: UserWarning
to the Keras 2 API: `Conv2D(filters=64, kernel_size=(2, 2), activation="relu", input_shape
lid")`
  import sys
```

```
Train on 50000 samples, validate on 10000 samples
```

Playing Lego

- Merge Layers
  - Add
  - Subtract
  - Multiply
  - Average
  - Maximum
  - Concatenate
  - Dot
  - add
  - subtract
  - multiply
  - average
  - maximum
  - concatenate
  - dot

Add [source]

```
keras.layers.Add()
```

Layer that adds a list of inputs.

It takes as input a list of tensors, all of the same shape, and returns a single tensor (also of the same shape).

Examples

```
import keras

input1 = keras.layers.Input(shape=(16,))
x1 = keras.layers.Dense(8, activation='relu')(input1)
input2 = keras.layers.Input(shape=(32,))
x2 = keras.layers.Dense(8, activation='relu')(input2)
added = keras.layers.Add()([x1, x2])  # equivalent to added = keras.layers.add([x1, x2])

out = keras.layers.Dense(4)(added)
model = keras.models.Model(inputs=[input1, input2], outputs=out)
```

Multiply

```
keras.layers.Multiply()
```

Layer that multiplies (element-wise) a list of inputs.

It takes as input a list of tensors, all of the same shape, and returns a single tensor (also

2007 NIPS Tutorial on:

Deep Belief Nets

Geoffrey Hinton
Canadian Institute for Advanced Research
&
Department of Computer Science
University of Toronto

How many layers should we use and how wide should they be?

(I am indebted to Karl Rove for this slide)

- How many lines of code should an AI program use and how long should each line be?
  - This is obviously a silly question.
- Deep belief nets give the creator a lot of freedom.
  - How best to make use of that freedom depends on the task.
  - With enough narrow layers we can model any distribution over binary vectors (Sutskever & Hinton, 2007)
- If freedom scares you, stick to convex optimization of shallow models that are obviously inadequate for doing Artificial Intelligence.

Embedding layers

Embedding

[source]

```
keras.layers.Embedding(input_dim, output_dim, embeddings_initializer='uniform', embeddings_re
```

Turns positive integers (indexes) into dense vectors of fixed size. eg. [[4], [20]] -> [[0.25, 0.1], [0.6, -0.2]]

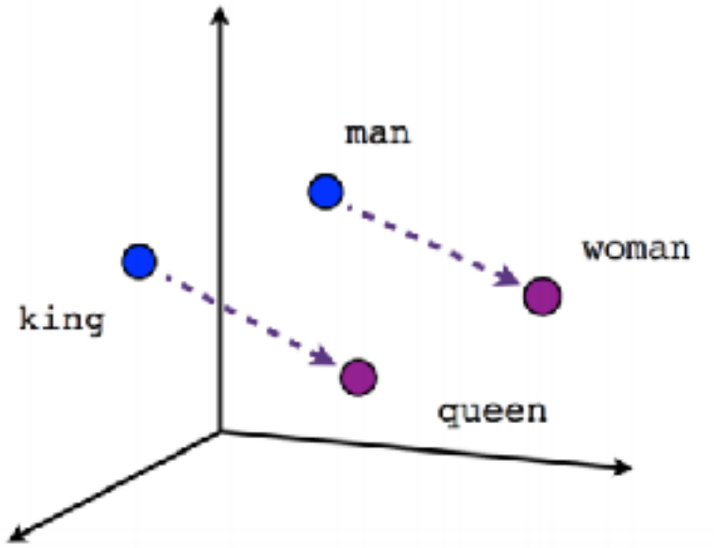
This layer can only be used as the first layer in a model.

Example

```
model = Sequential()
model.add(Embedding(1000, 64, input_length=10))
# the model will take as input an integer matrix of size (batch, input_length).
# the largest integer (i.e. word index) in the input should be no larger than 999 (vocabulary
# now model.output_shape == (None, 10, 64), where None is the batch dimension.

input_array = np.random.randint(1000, size=(32, 10))

model.compile('rmsprop', 'mse')
output_array = model.predict(input_array)
assert output_array.shape == (32, 10, 64)
```

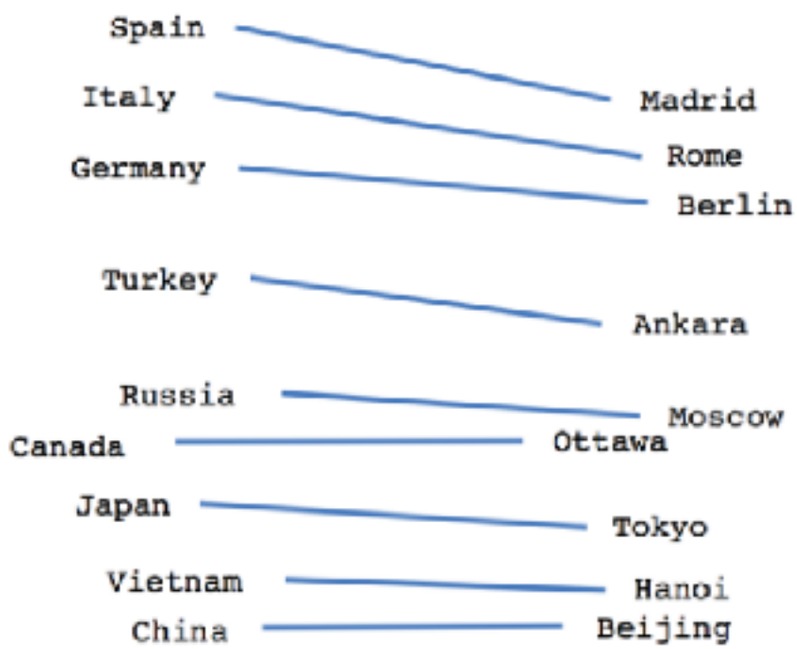
Embedding layers

Example: word2vec



Male-Female

Verb tense



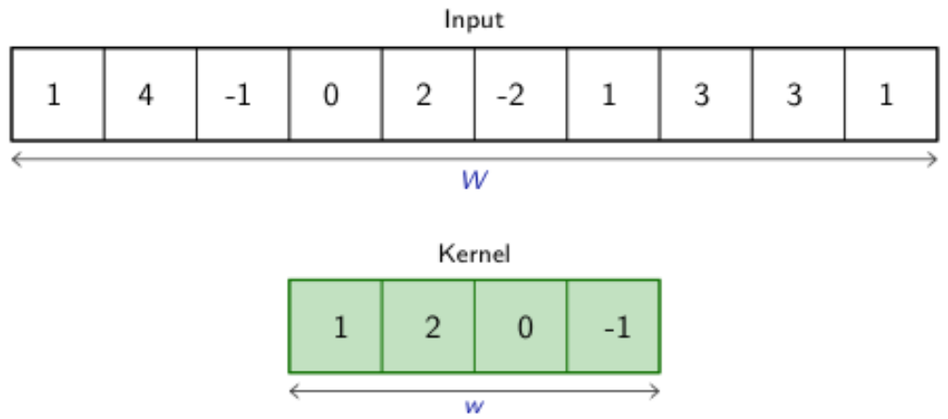
Country-Capital

Preservation of semantic and syntactic relationships

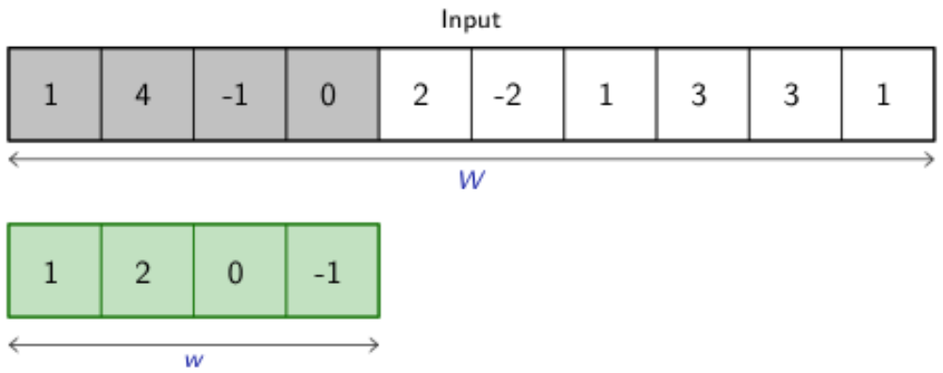
Convolutional and pooling layers

Fondamental for images and sounds!

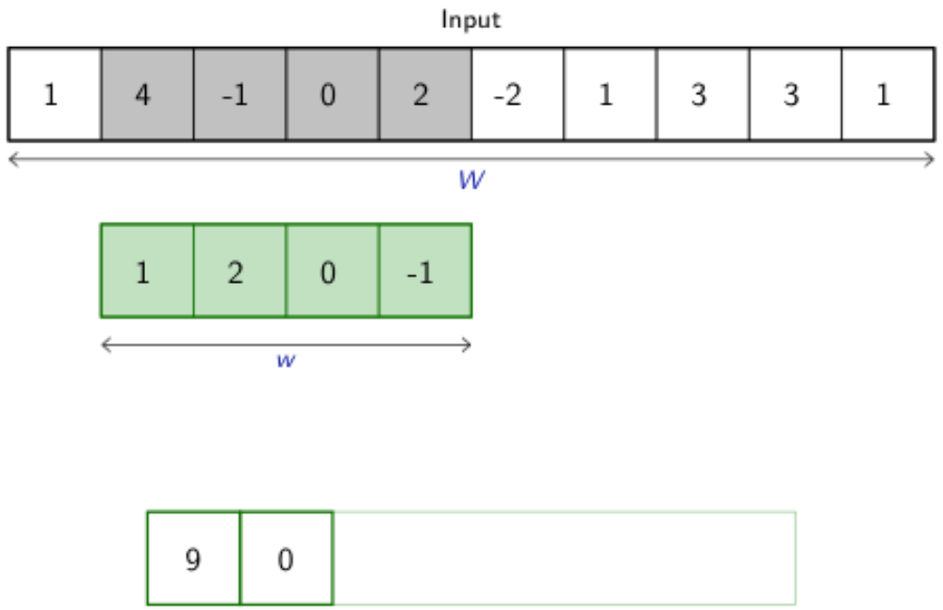
Convolution 1d



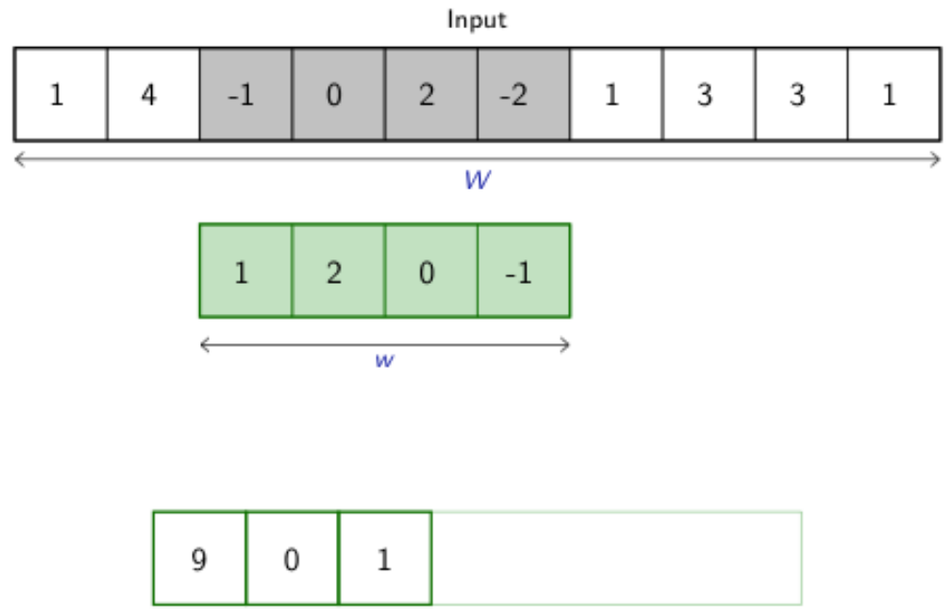
Convolution 1d



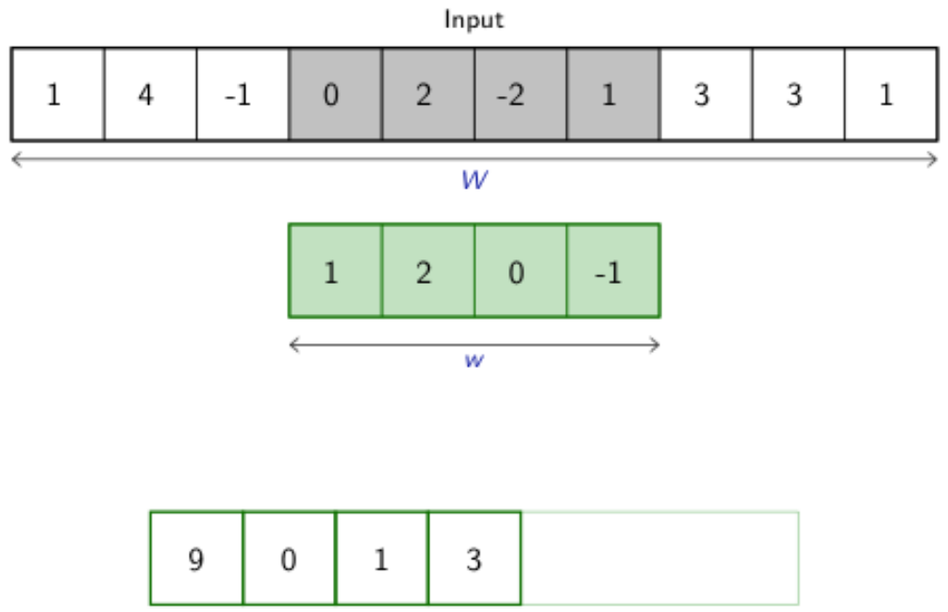
Convolution 1d



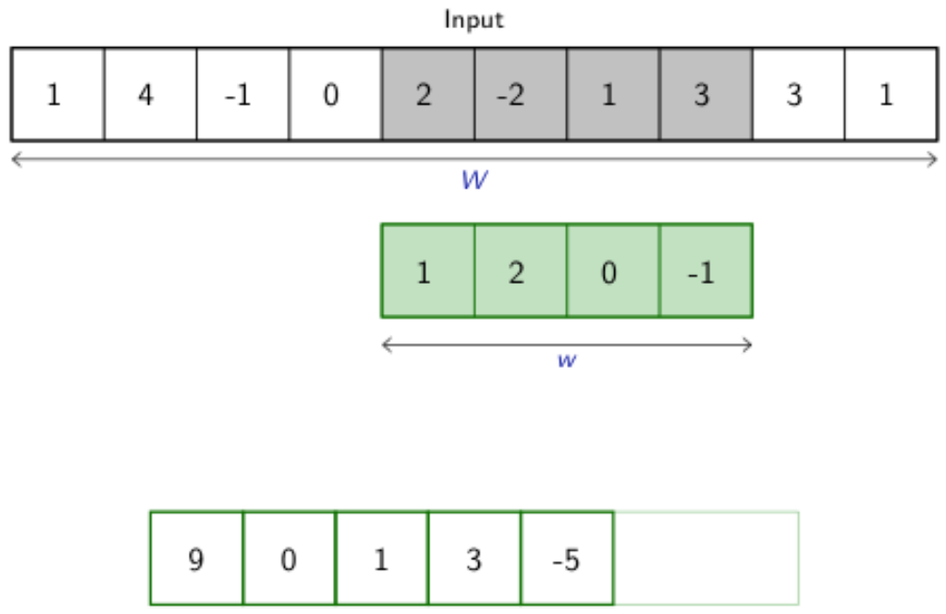
Convolution 1d



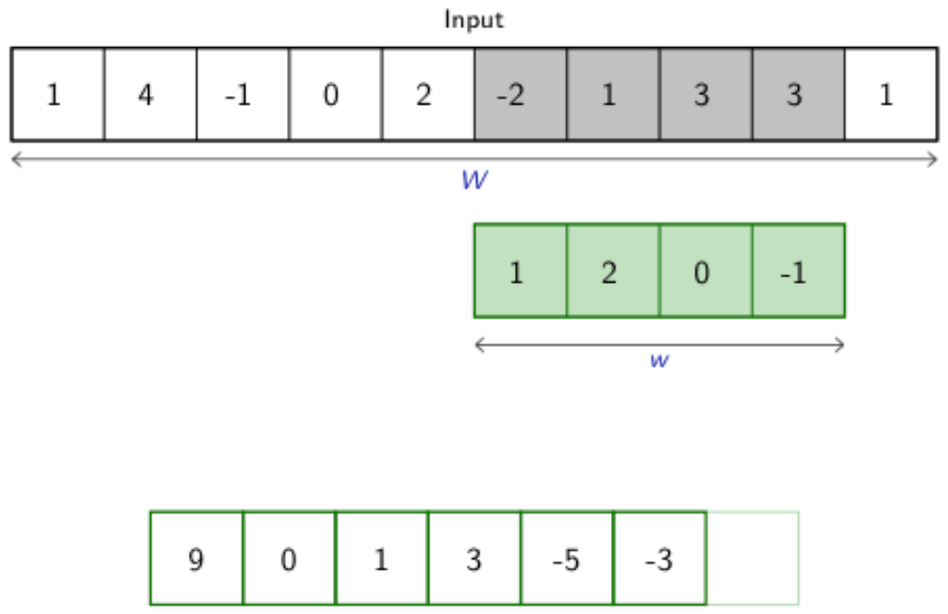
Convolution 1d



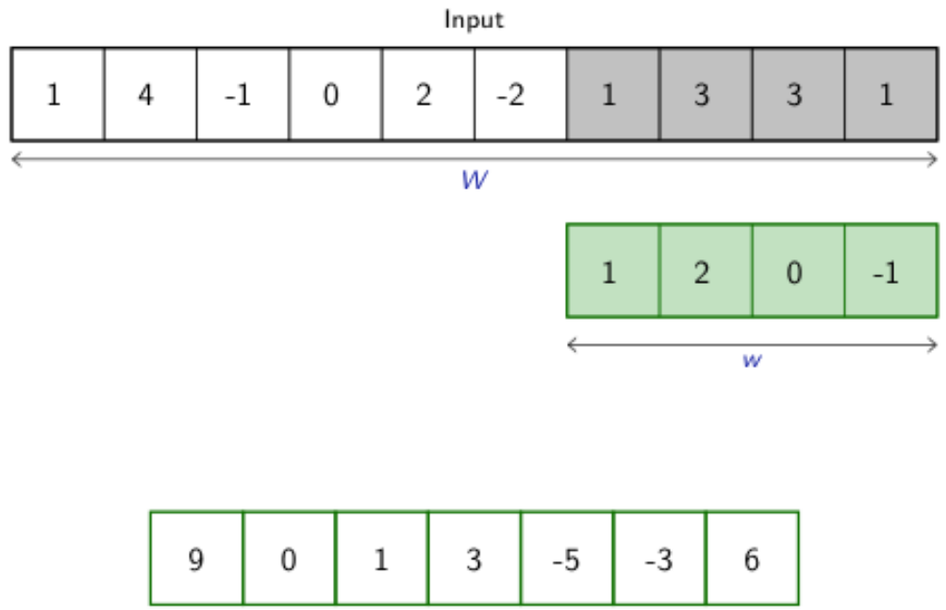
Convolution 1d



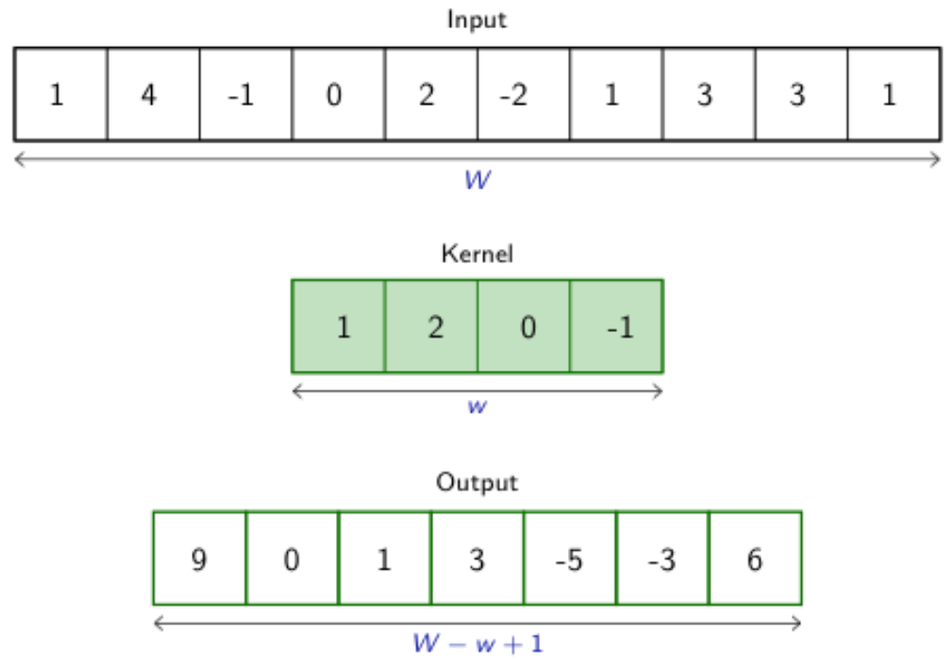
Convolution 1d



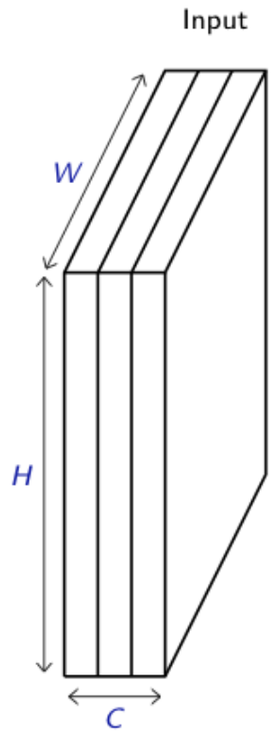
Convolution 1d



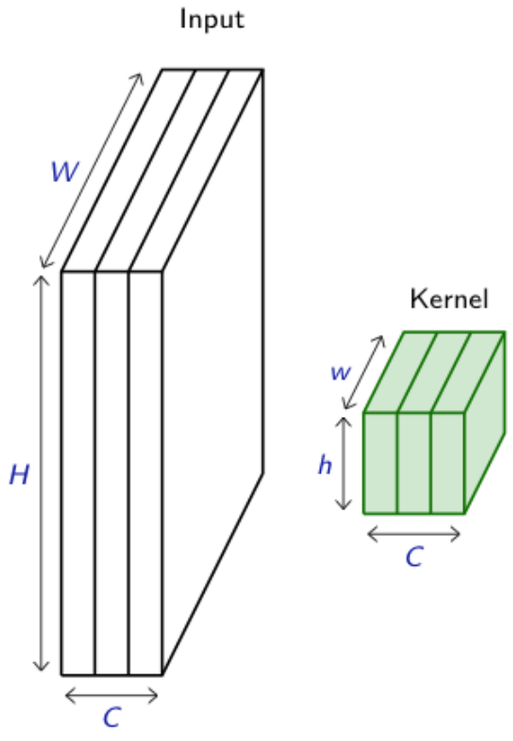
Convolution 1d



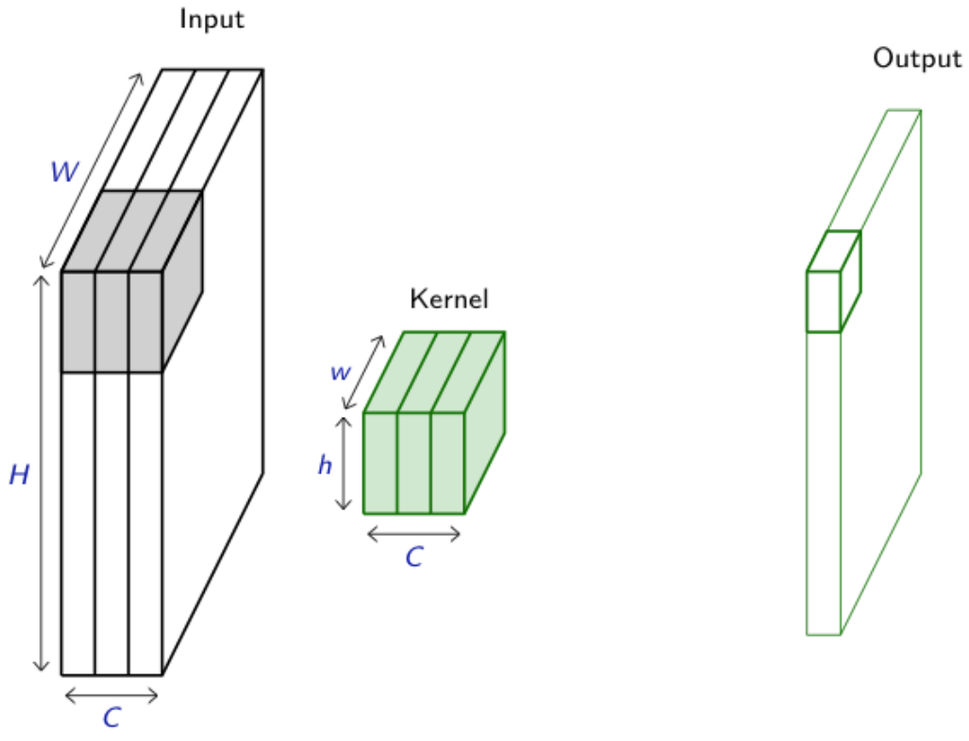
Convolution 2d



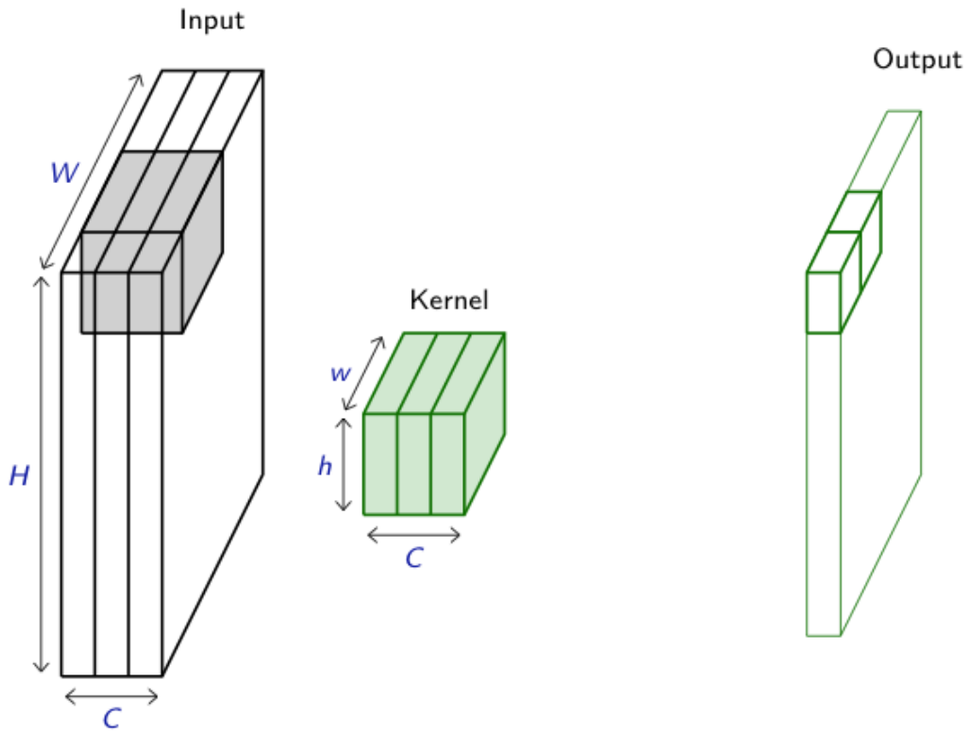
Convolution 2d



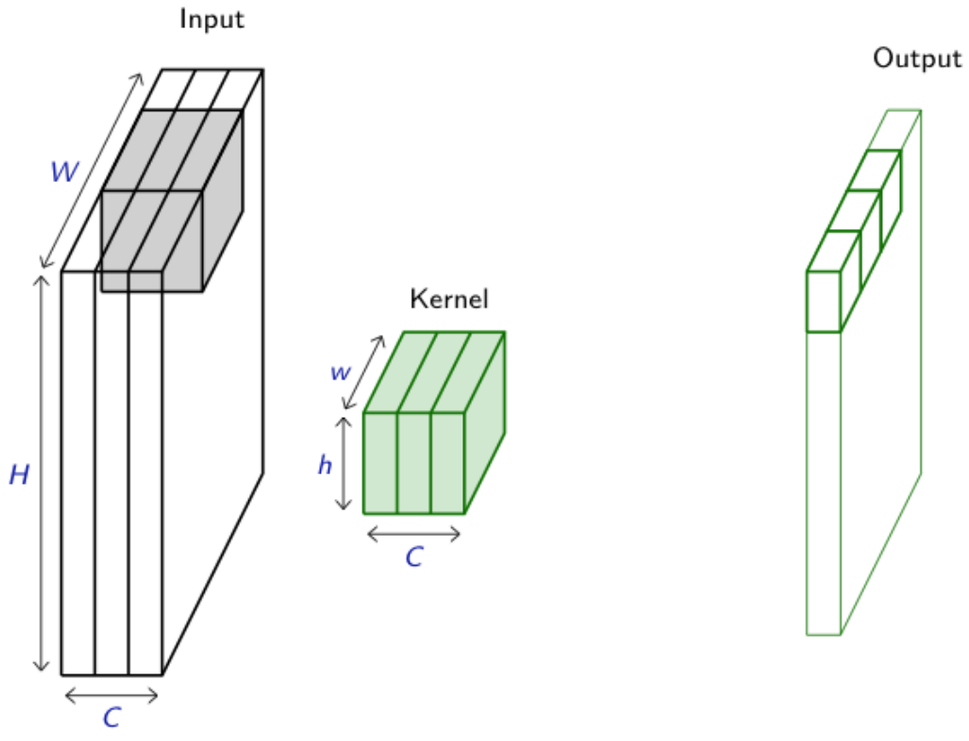
Convolution 2d



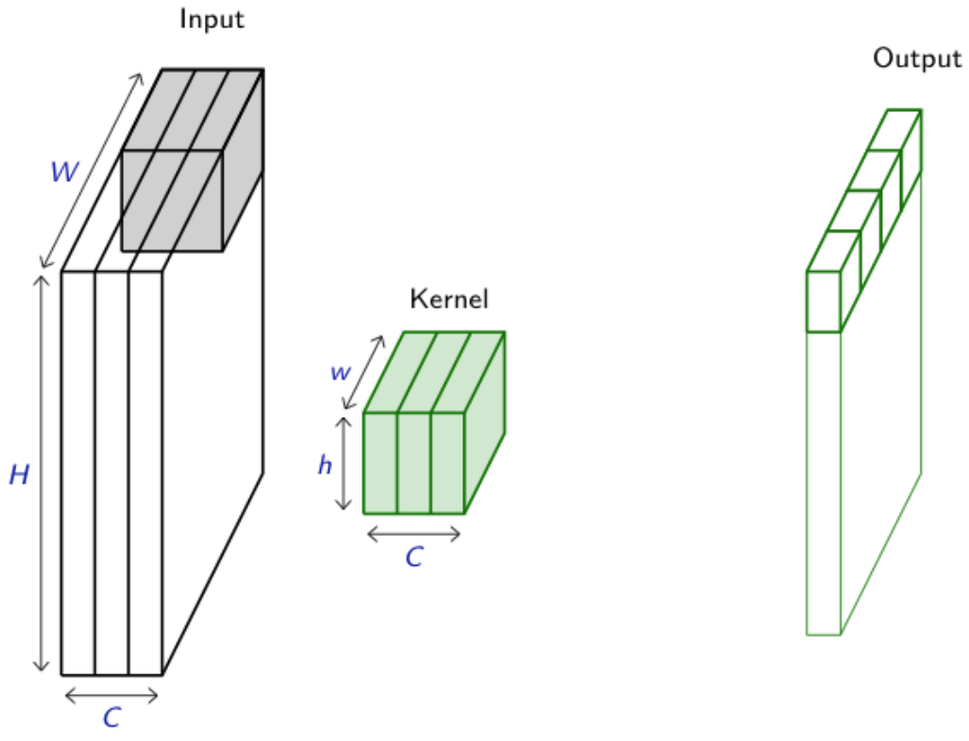
Convolution 2d



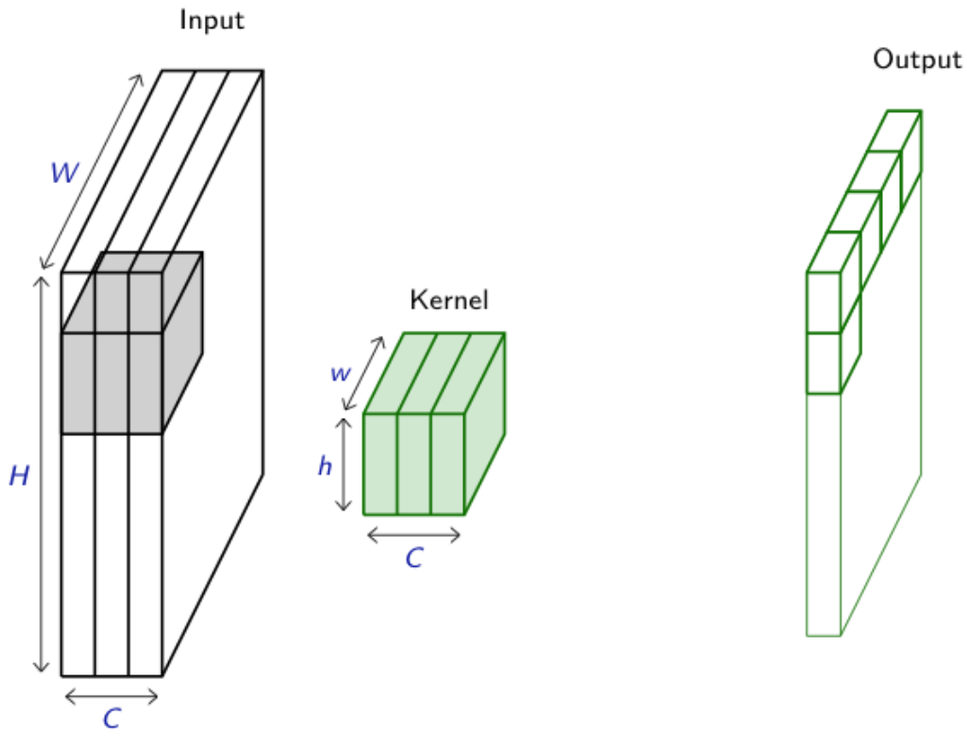
Convolution 2d



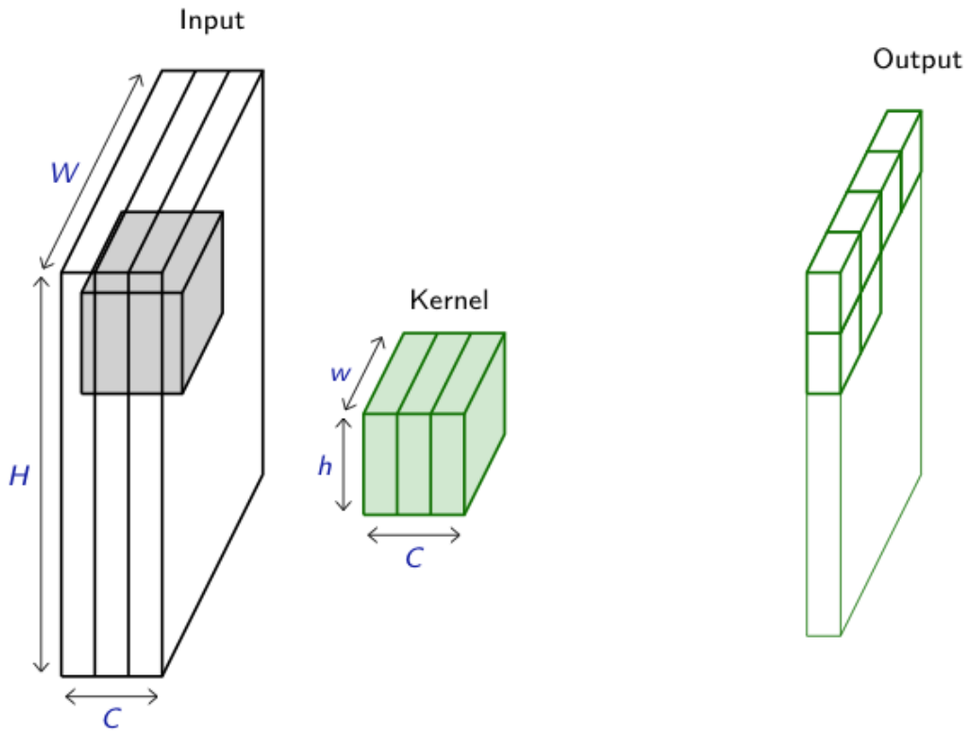
Convolution 2d



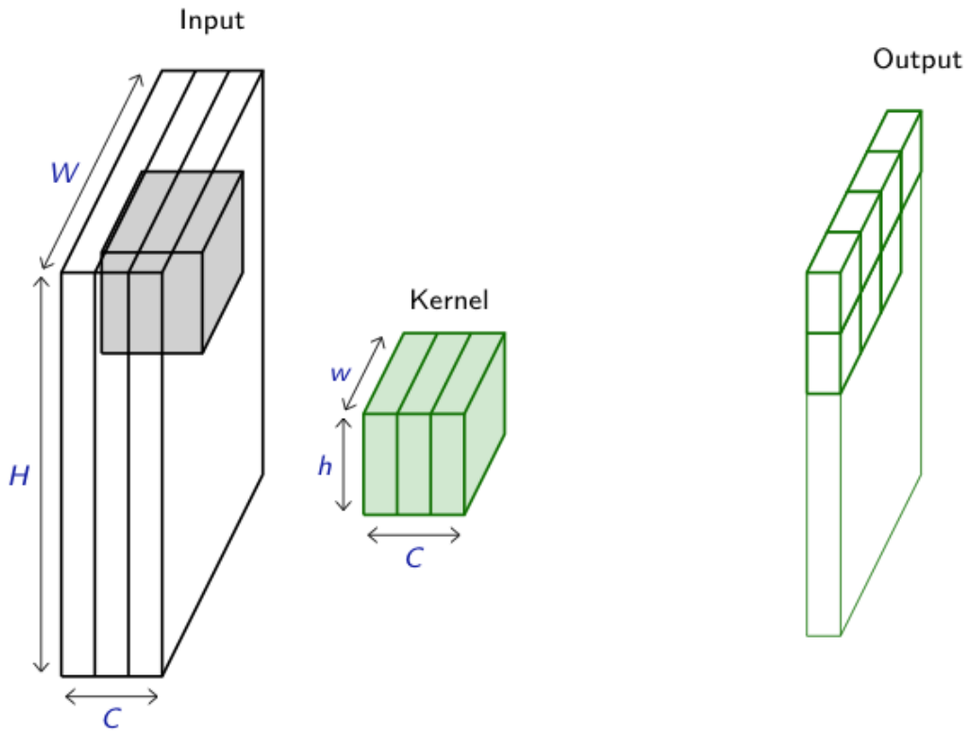
Convolution 2d



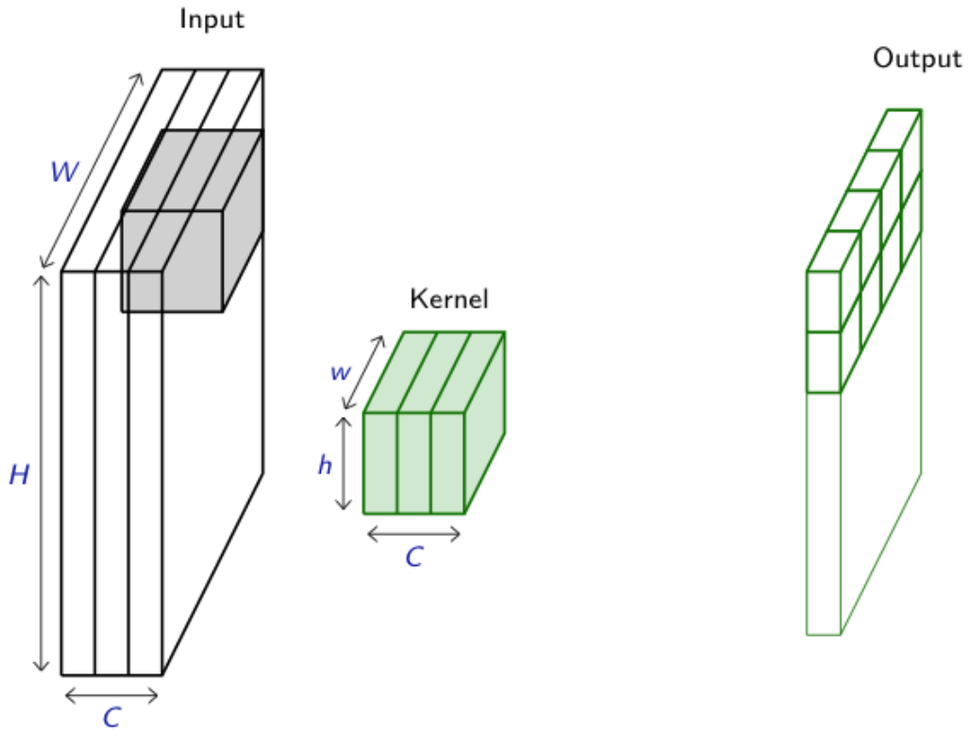
Convolution 2d



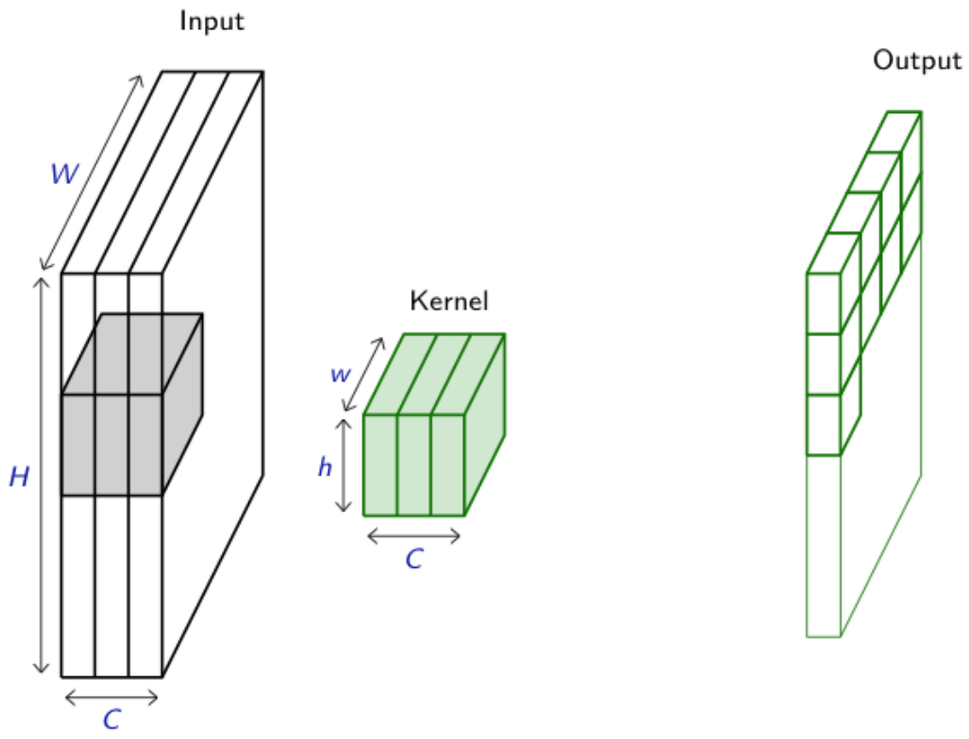
Convolution 2d



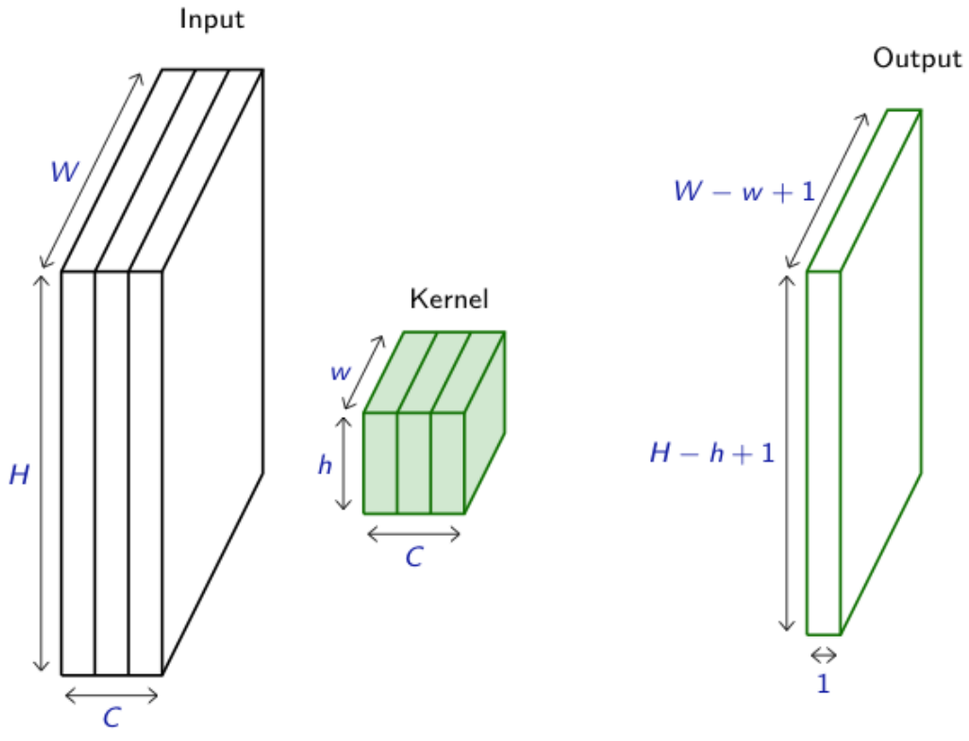
Convolution 2d



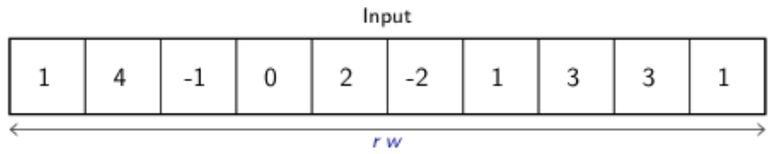
Convolution 2d



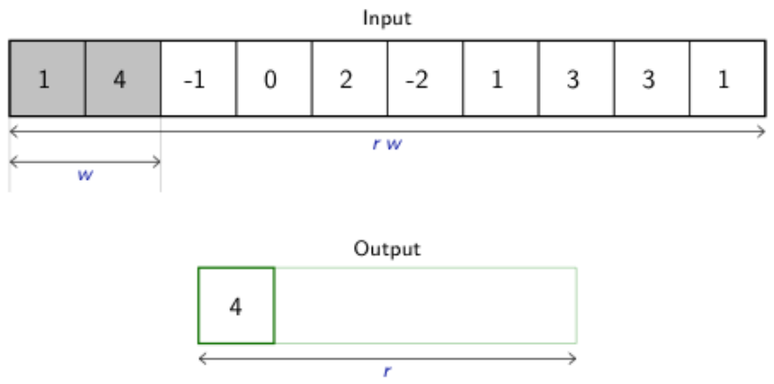
Convolution 2d



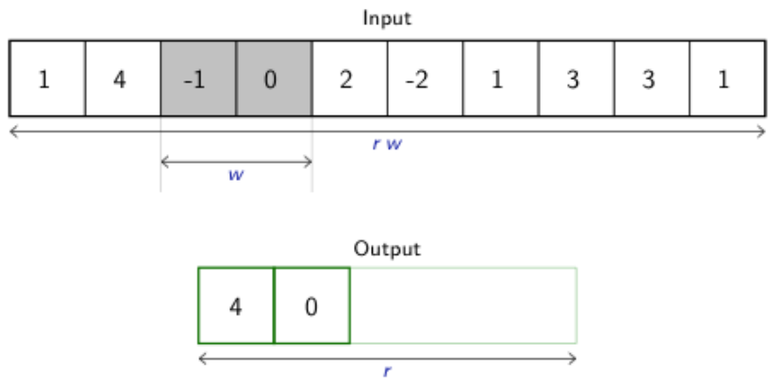
Max-Pooling 1d



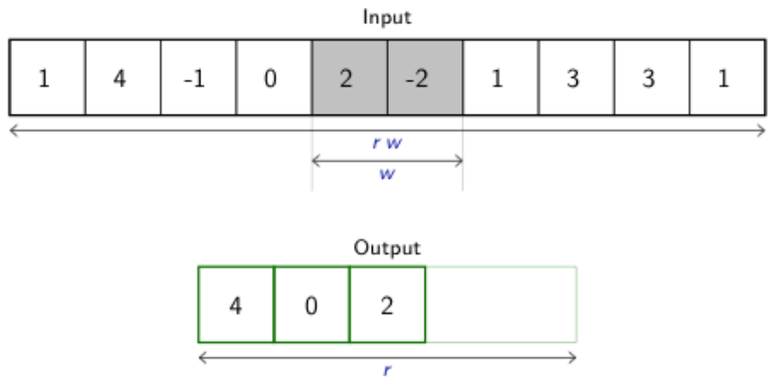
Max-Pooling 1d



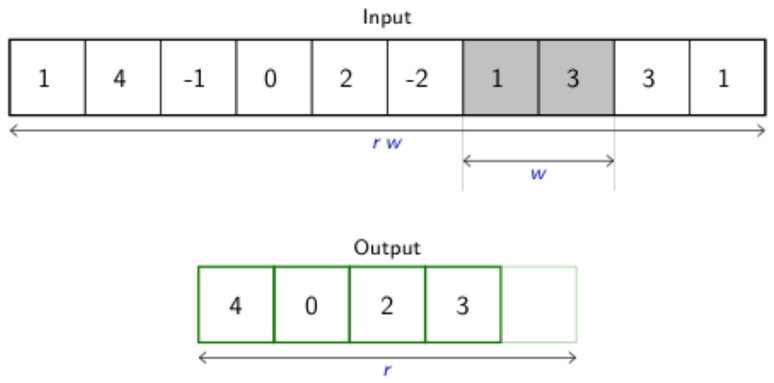
Max-Pooling 1d



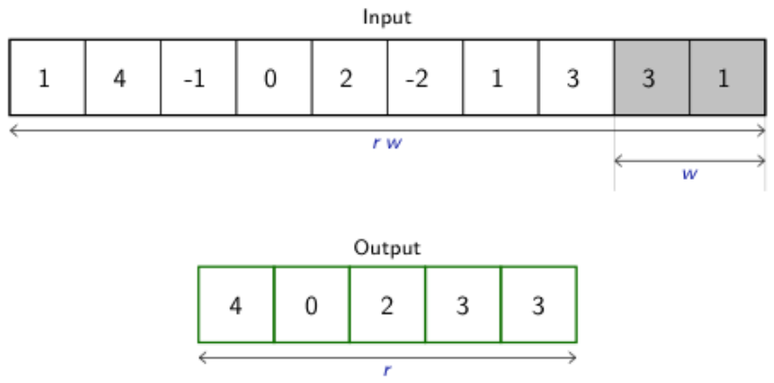
Max-Pooling 1d



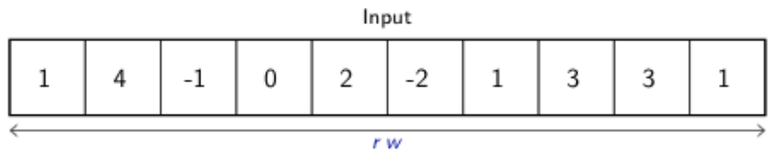
Max-Pooling 1d

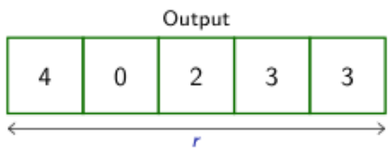


Max-Pooling 1d

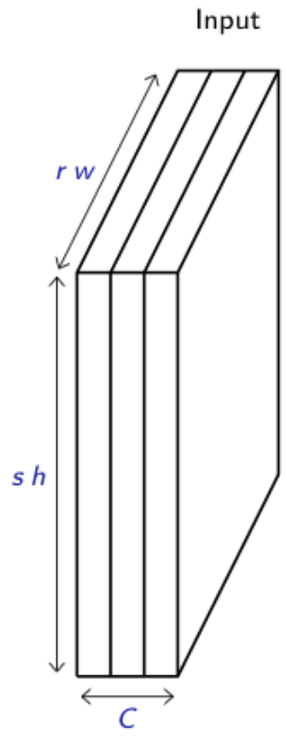


Max-Pooling 1d

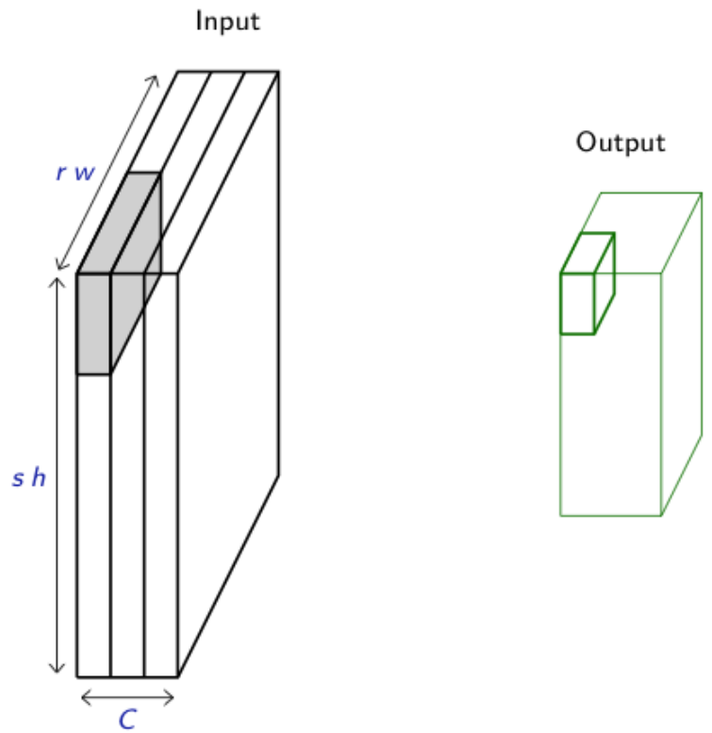




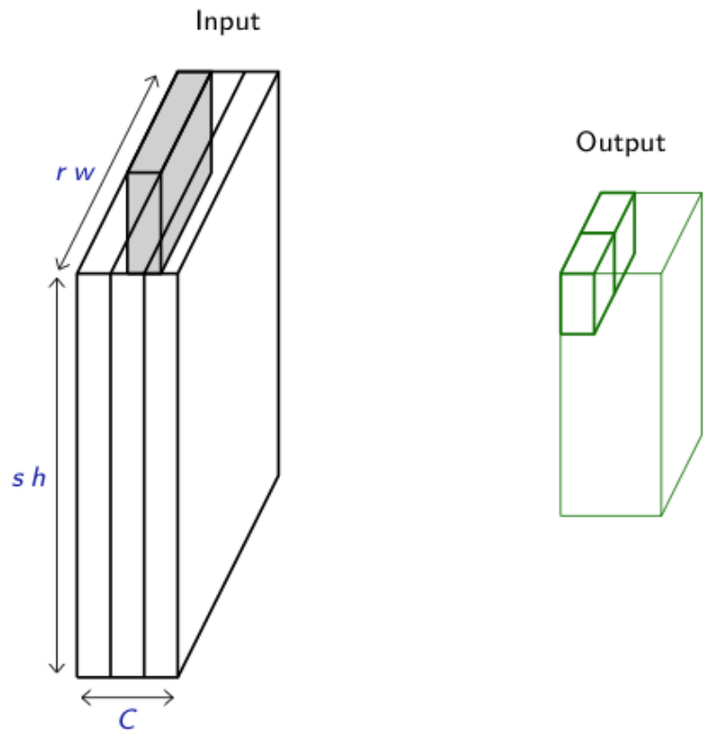
Max-Pooling 2d



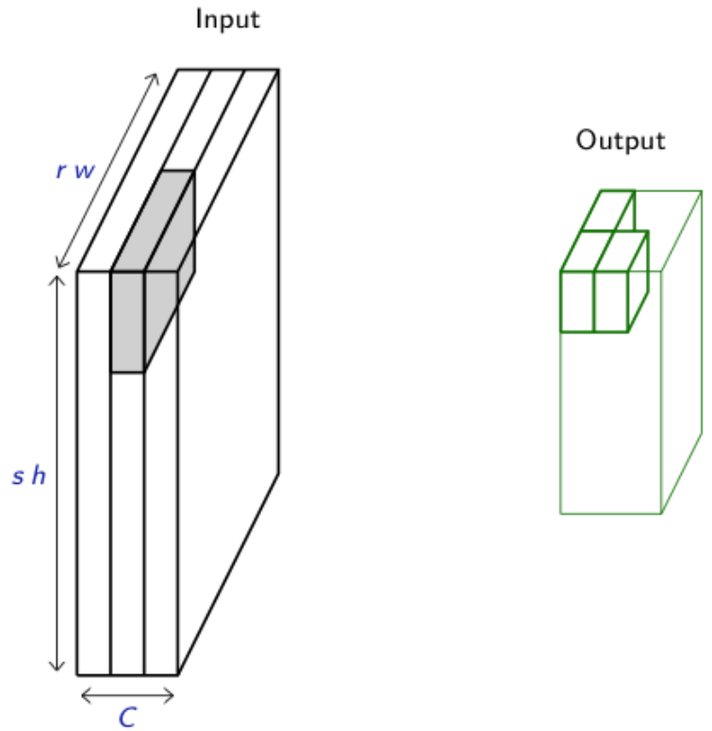
Max-Pooling 2d



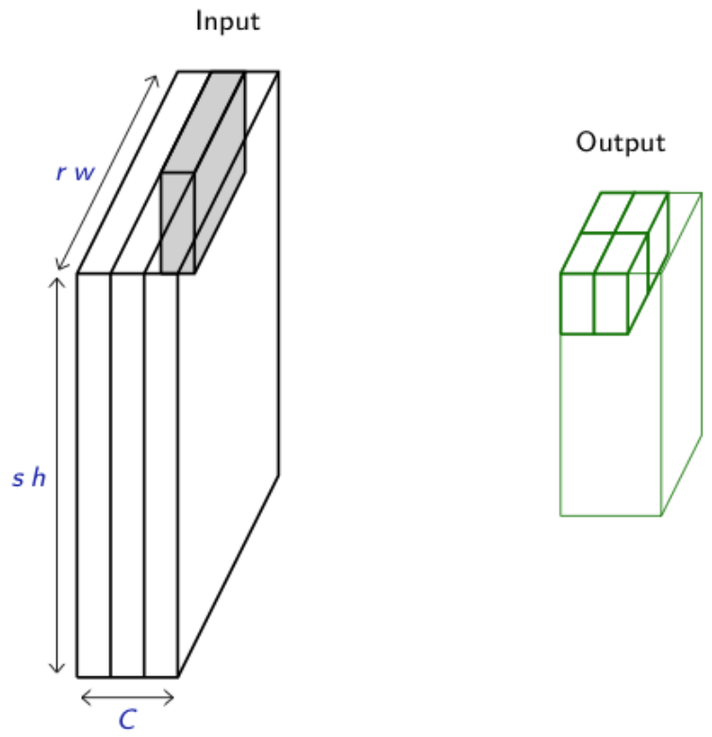
Max-Pooling 2d



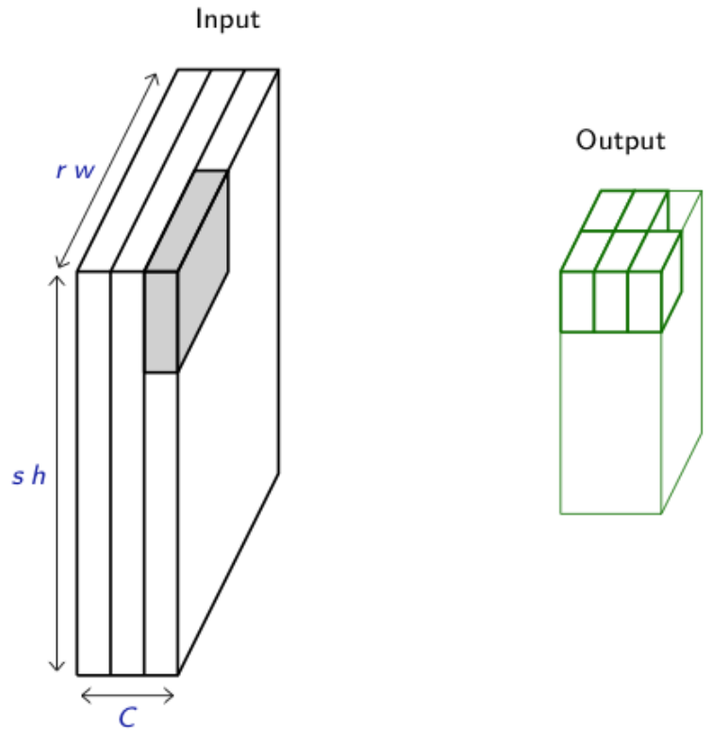
Max-Pooling 2d



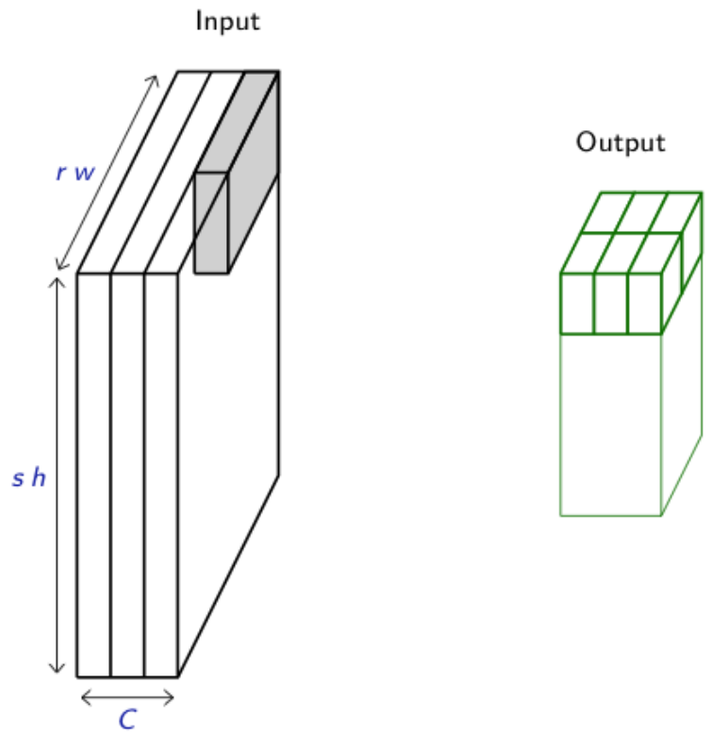
Max-Pooling 2d



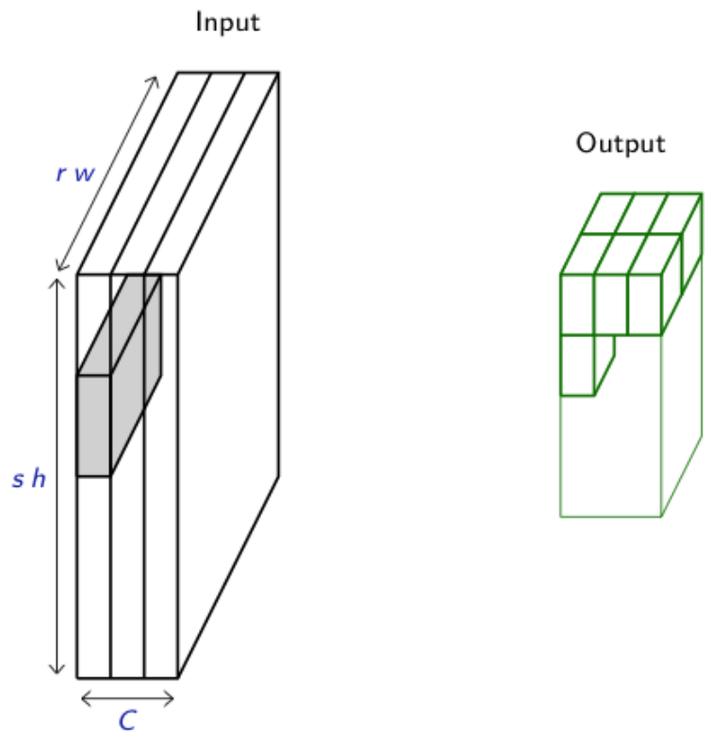
Max-Pooling 2d



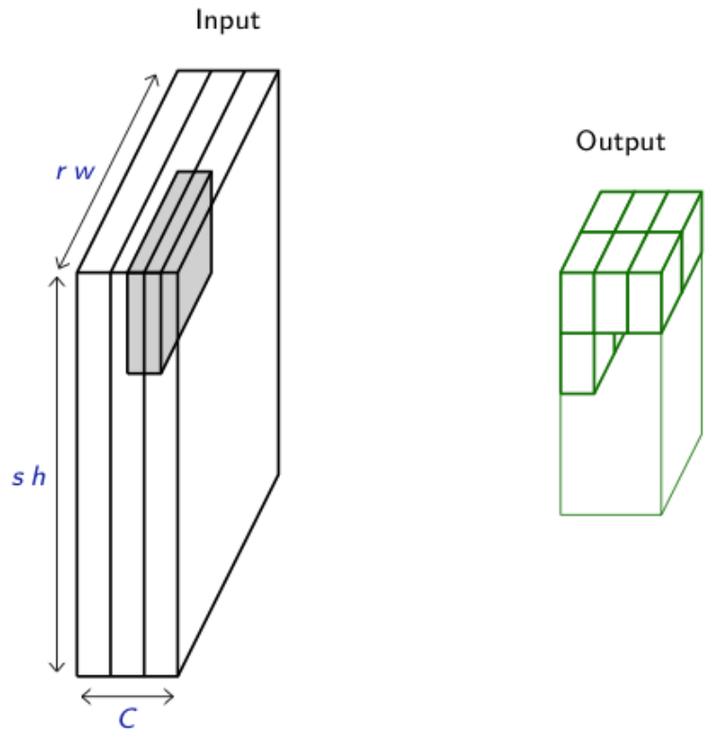
Max-Pooling 2d



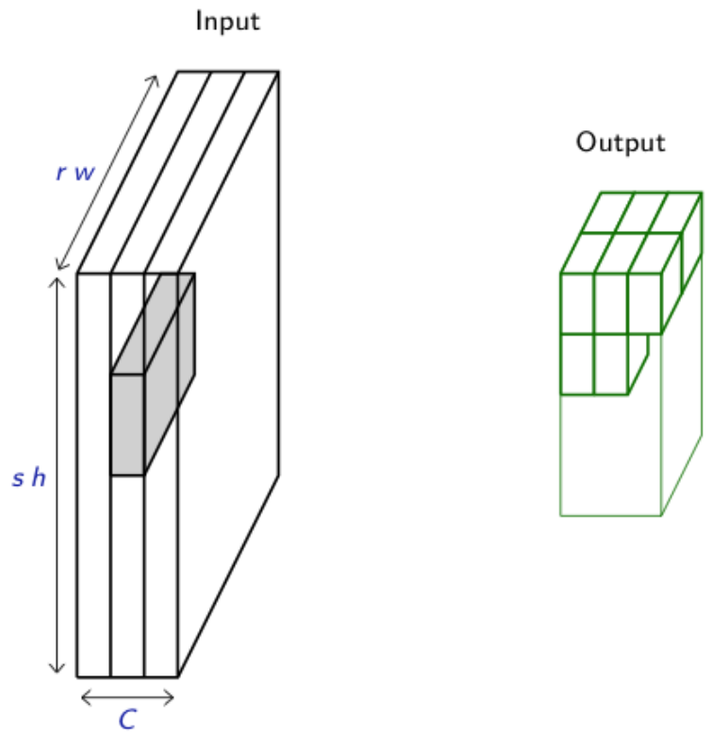
Max-Pooling 2d



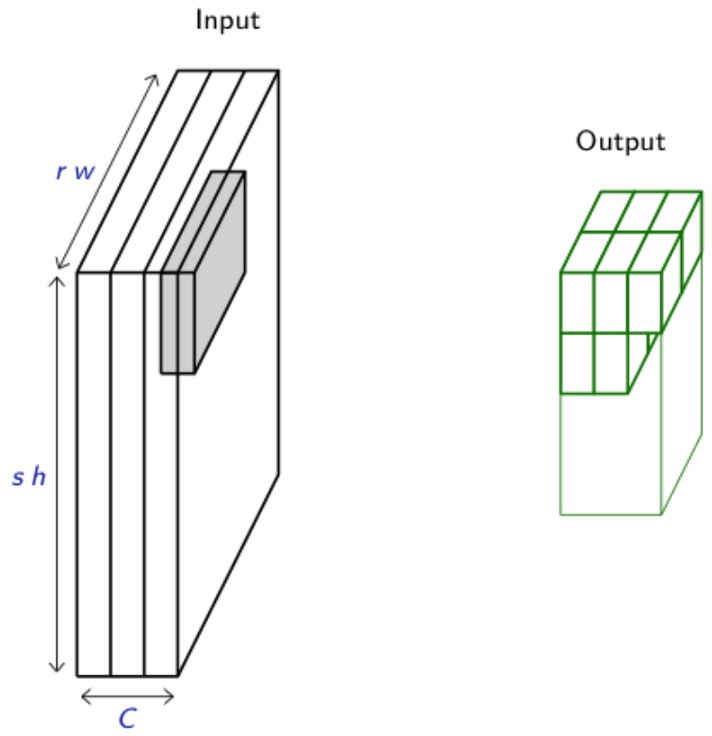
Max-Pooling 2d



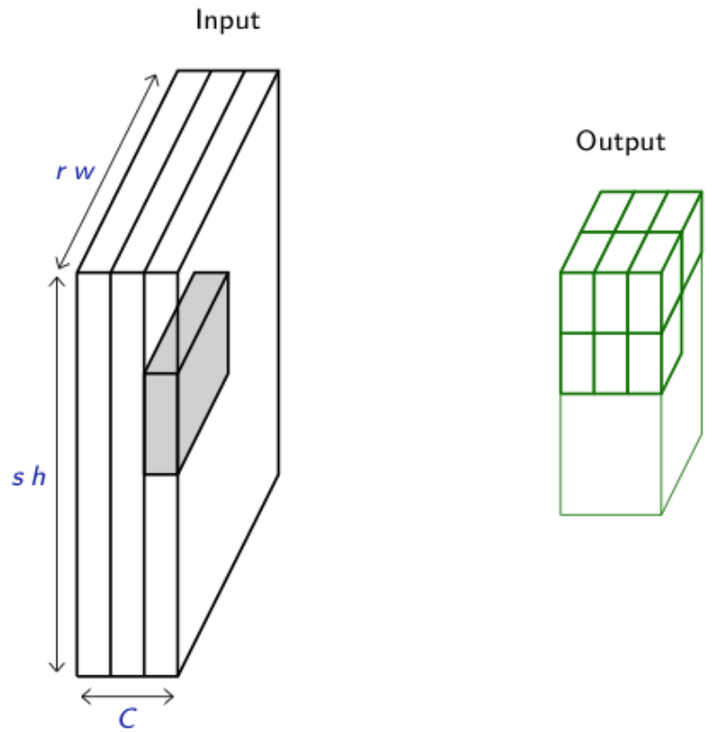
Max-Pooling 2d



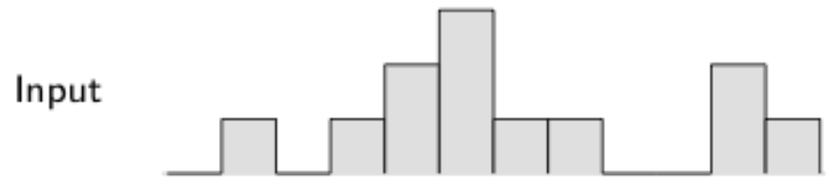
Max-Pooling 2d



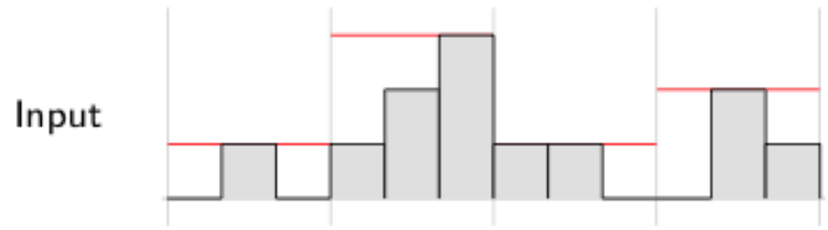
Max-Pooling 2d



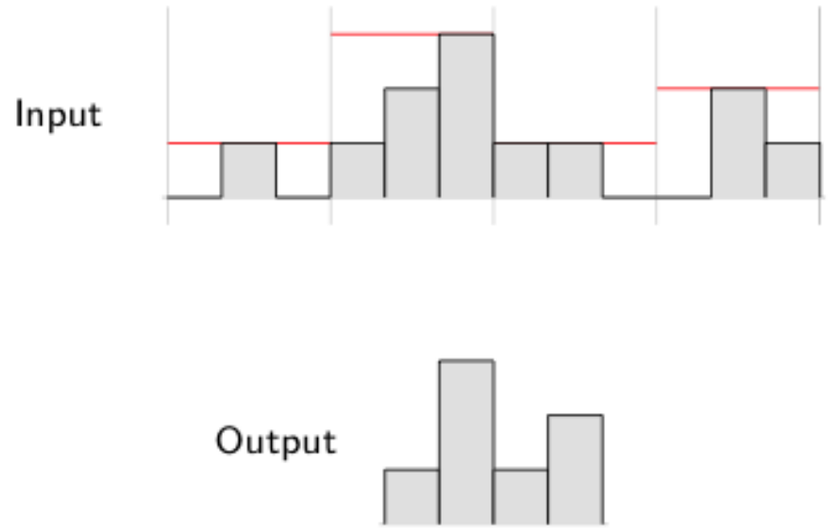
Translation invariance from pooling



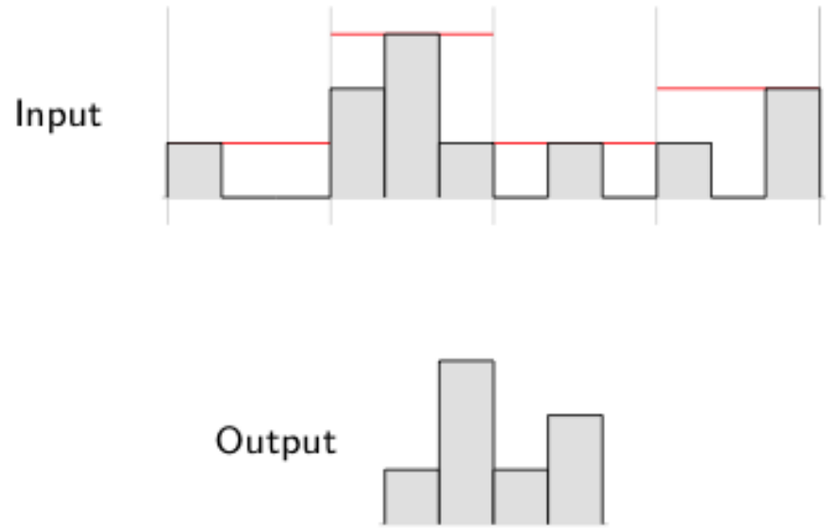
Translation invariance from pooling



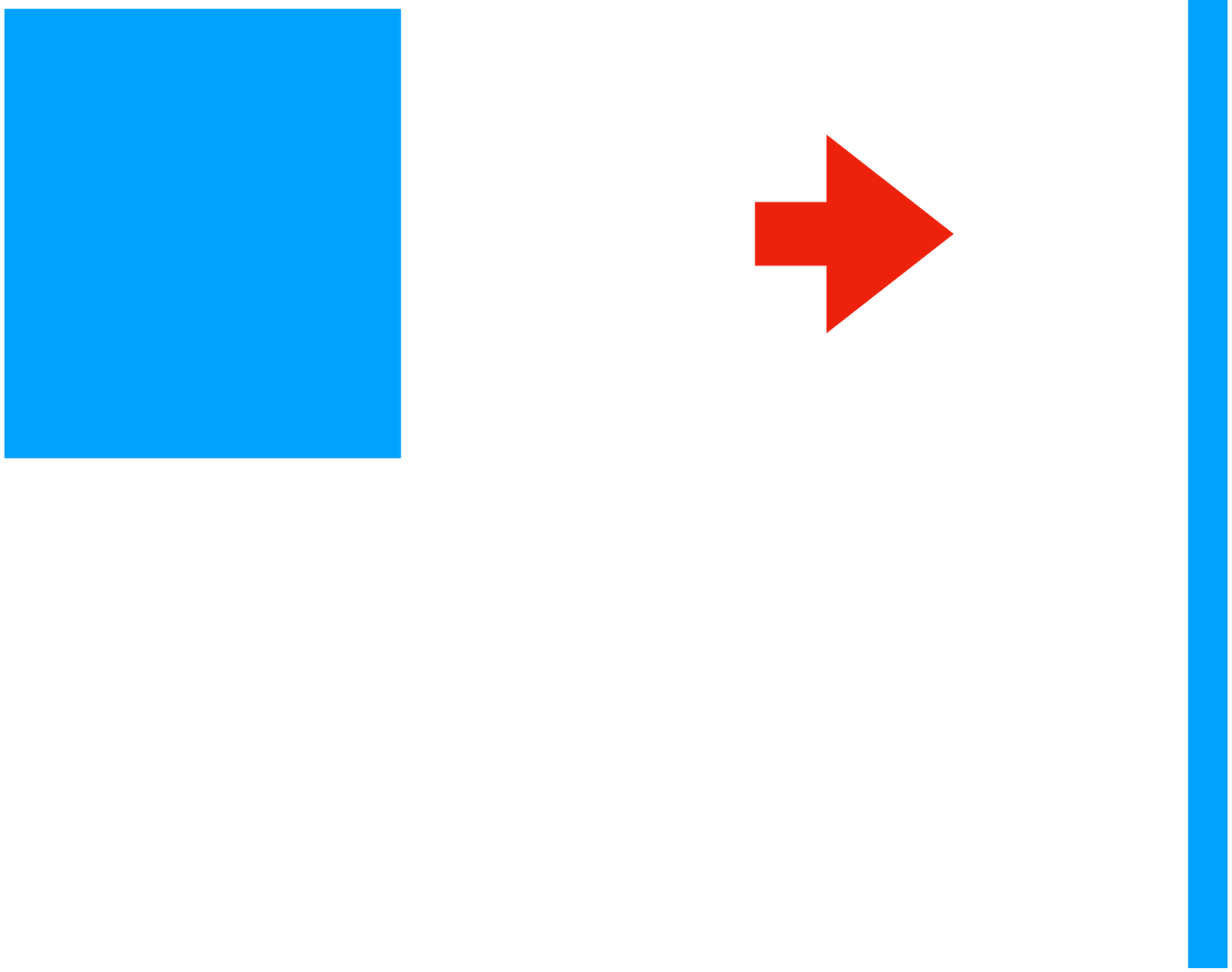
Translation invariance from pooling



Translation invariance from pooling



Flatten
2D->1D



Conv-nets

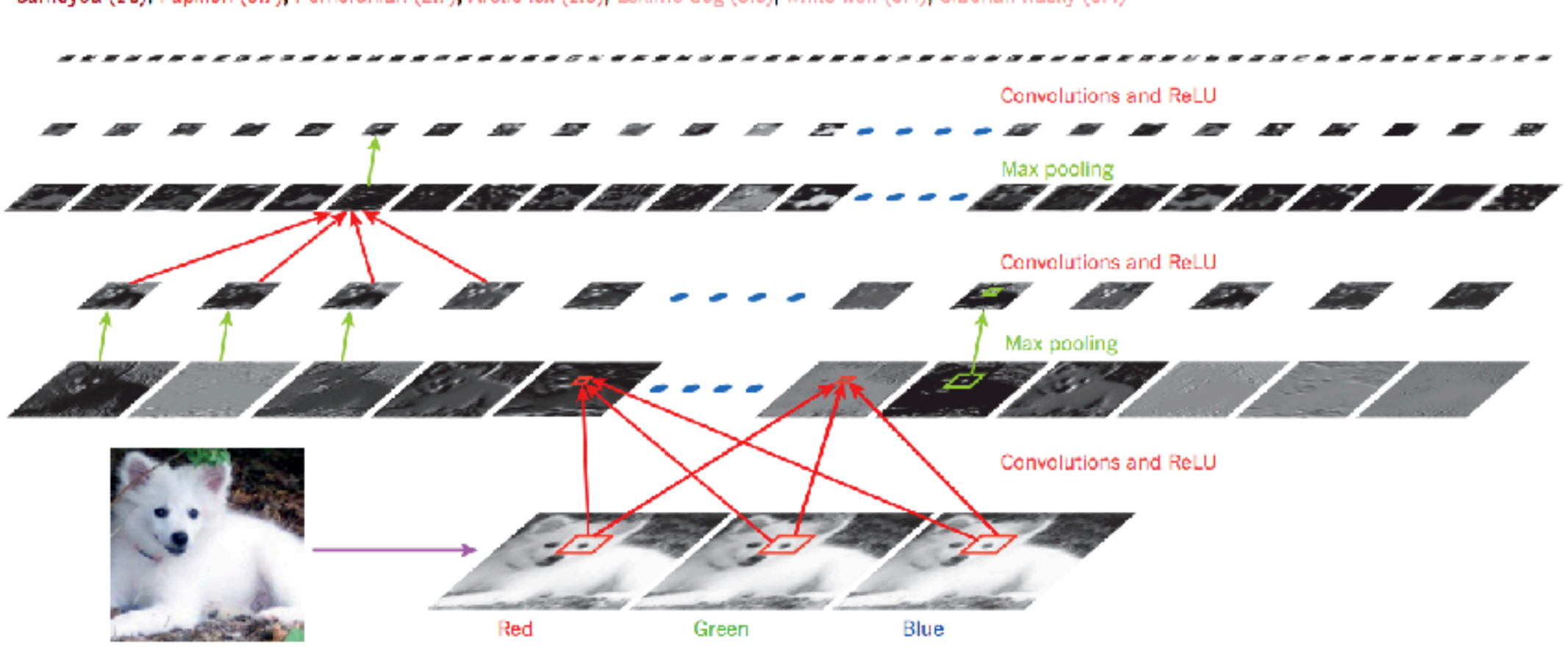
REVIEW

doi:10.1038/nature14539

Deep learning

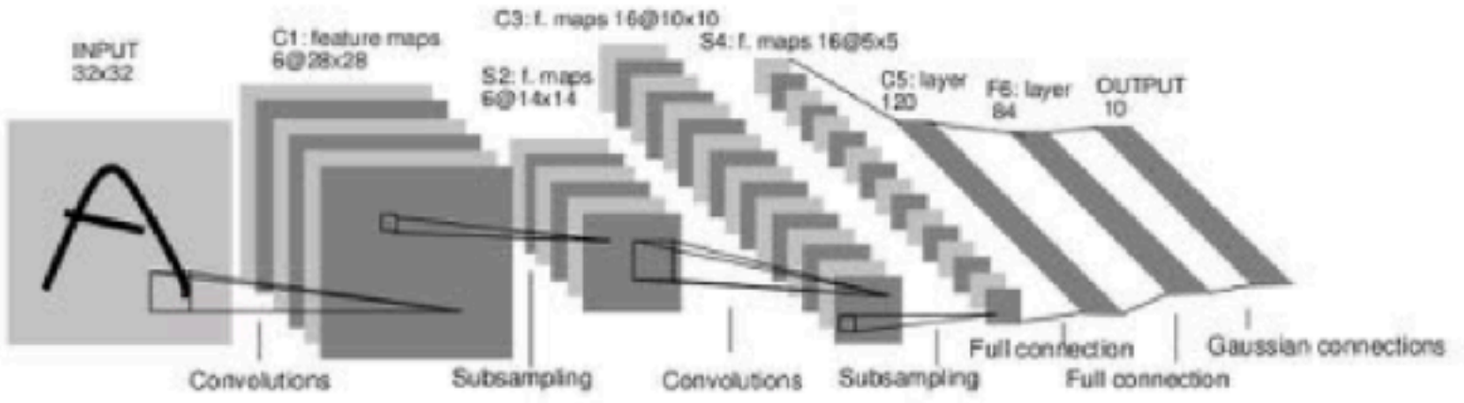
Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.



ConvNet

- Neural network with specialized connectivity structure
- Stack multiple stage of feature extractors
- Higher stages compute more global, more invariant features
- Classification layer at the end



LeNet5

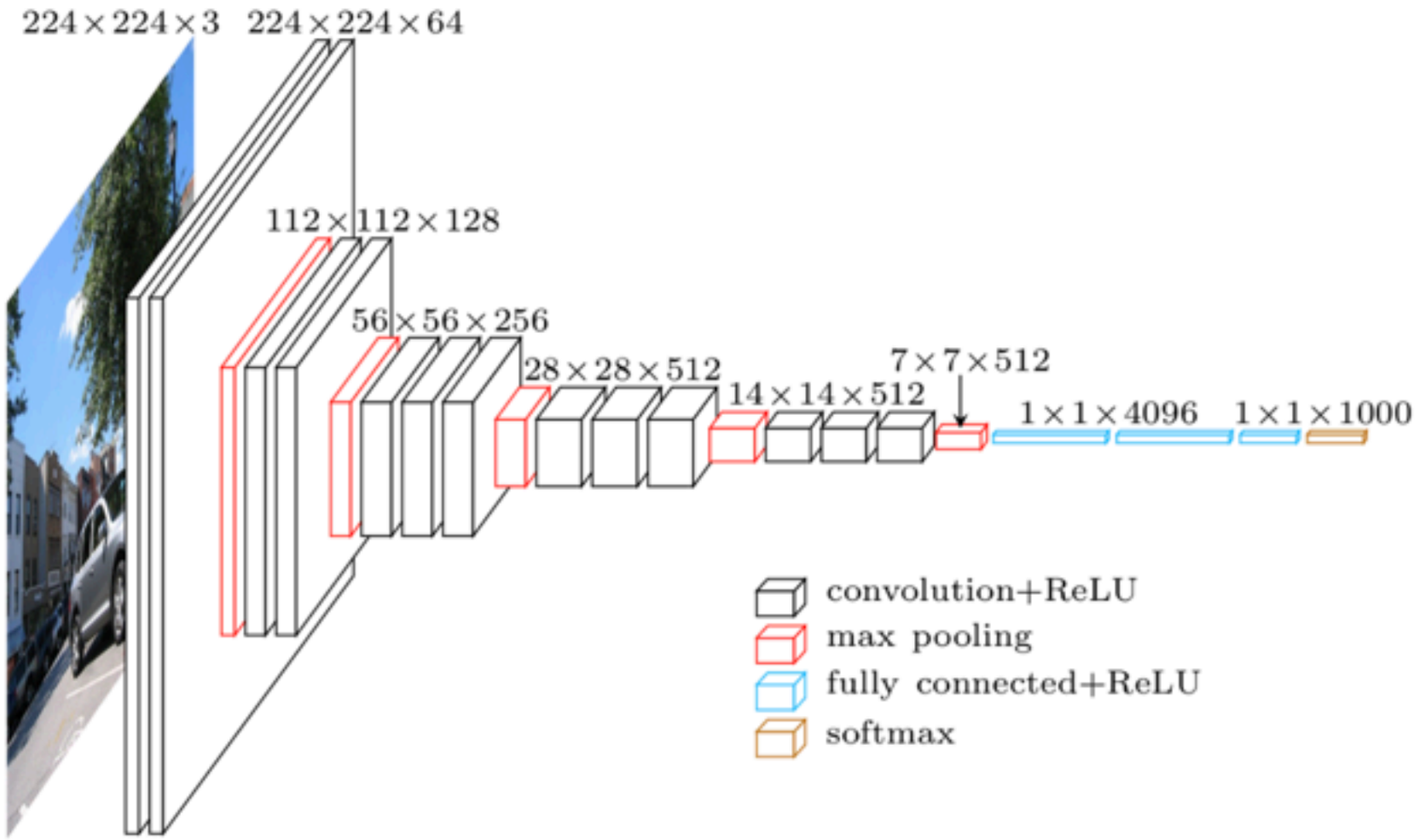
10 classes, input 1 x 28 x 28

```
(features): Sequential (
(0): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
(1): ReLU (inplace)
(2): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))
(3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
(4): ReLU (inplace)
(5): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))
)
(classifier): Sequential (
(0): Linear (400 -> 120)
(1): ReLU (inplace)
(2): Linear (120 -> 84)
(3): ReLU (inplace)
(4): Linear (84 -> 10) )
```

AlexNet

```
(features): Sequential (
(0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
(1): ReLU (inplace)
(2): MaxPool2d (size=(3, 3), stride=(2, 2), dilation=(1, 1))
(3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
(4): ReLU (inplace)
(5): MaxPool2d (size=(3, 3), stride=(2, 2), dilation=(1, 1))
(6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(7): ReLU (inplace)
(8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(9): ReLU (inplace)
(10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(11): ReLU (inplace)
(12): MaxPool2d (size=(3, 3), stride=(2, 2), dilation=(1, 1))
)

(classifier): Sequential (
(0): Dropout (p = 0.5)
(1): Linear (9216 -> 4096)
(2): ReLU (inplace)
(3): Dropout (p = 0.5)
(4): Linear (4096 -> 4096)
(5): ReLU (inplace)
(6): Linear (4096 -> 1000)
)
```

VGG-16


224 × 224 × 3
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
softmax

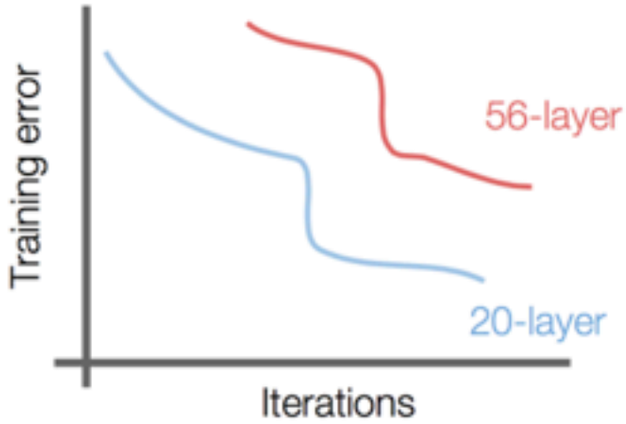
```
model = Sequential()
model.add(ZeroPadding2D((1, 1), input_shape=(3, 224, 224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2, 2), strides=(2, 2)))

model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1, 1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))

# Add another conv layer with ReLU + GAP
model.add(Convolution2D(num_input_channels, 3, 3, activation='relu', border_mode="same"))
model.add(AveragePooling2D((14, 14)))
model.add(Flatten())
# Add the W layer
model.add(Dense(nb_classes, activation='softmax'))
```

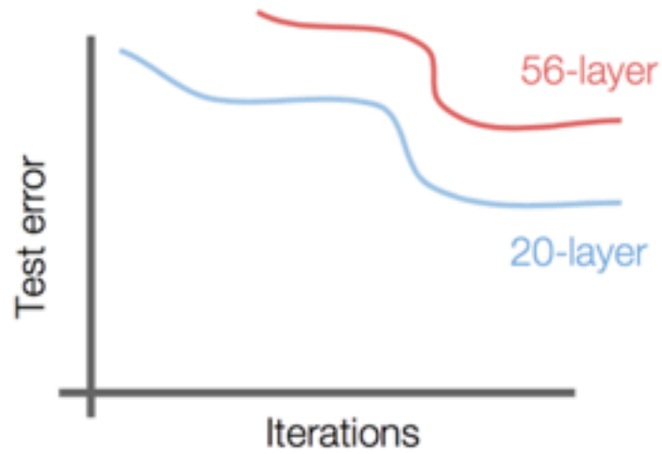
VGG-19

```
(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU (inplace)
(2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(3): ReLU (inplace)
(4): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))
(5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(6): ReLU (inplace)
(7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(8): ReLU (inplace)
(9): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))
(10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(11): ReLU (inplace)
(12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(13): ReLU (inplace)
(14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(15): ReLU (inplace)
(16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(17): ReLU (inplace)
(18): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))
(19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(20): ReLU (inplace)
(21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(22): ReLU (inplace)
(23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(24): ReLU (inplace)
(25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(26): ReLU (inplace)
(27): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))
(28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(29): ReLU (inplace)
...
```

A saturation point

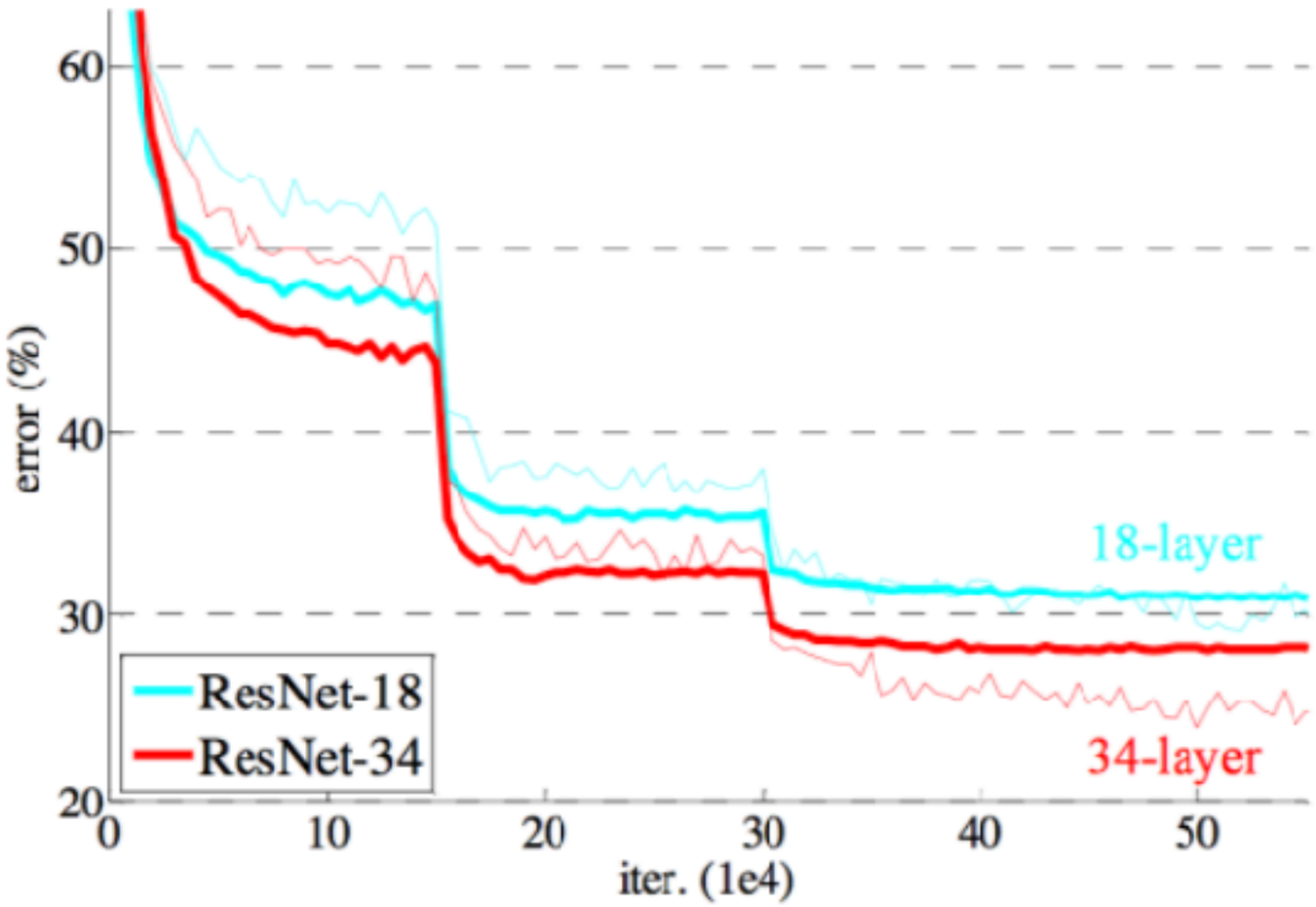
If we continue stacking more layers on a CNN:



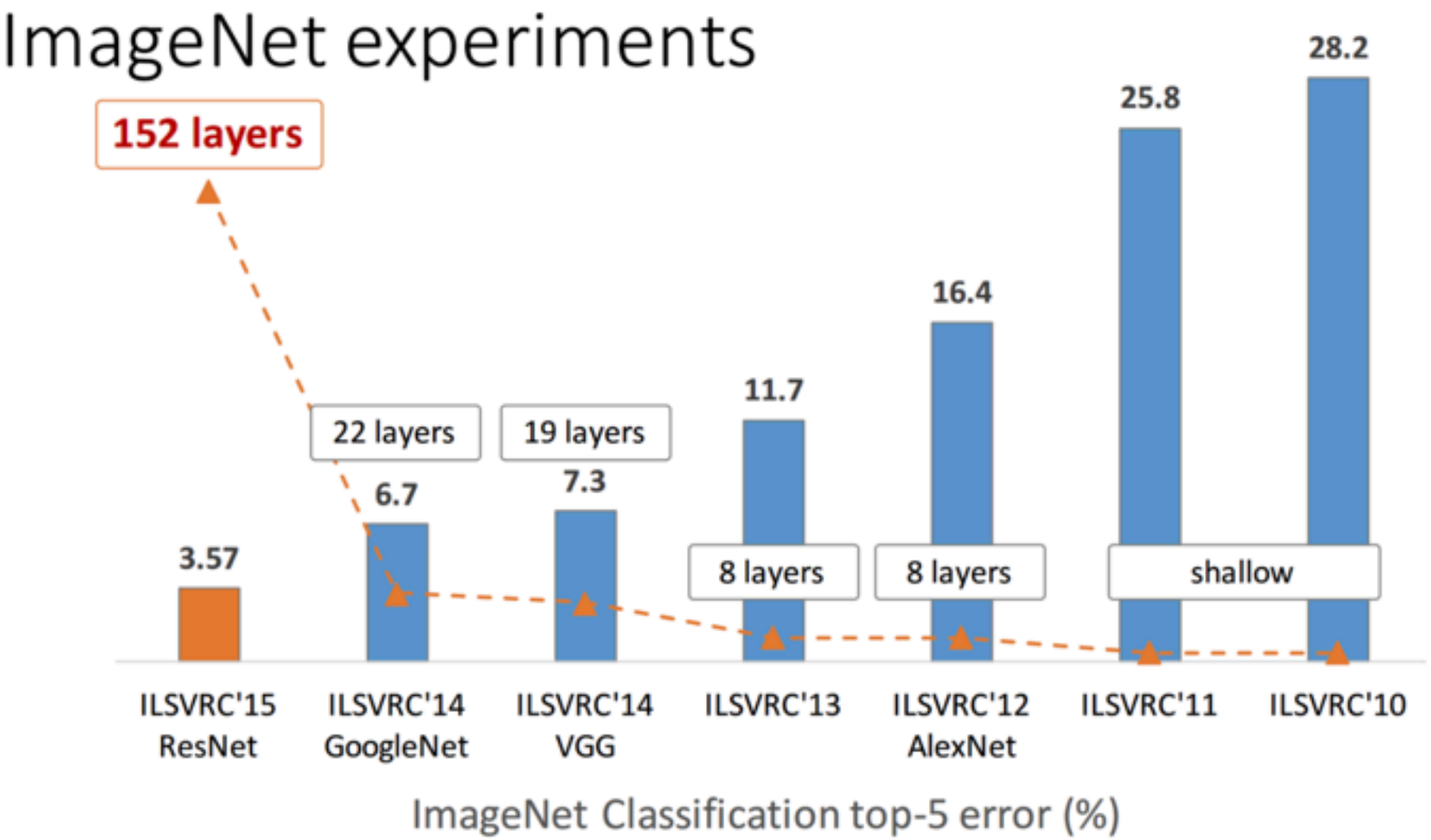


Deeper models are harder to optimize

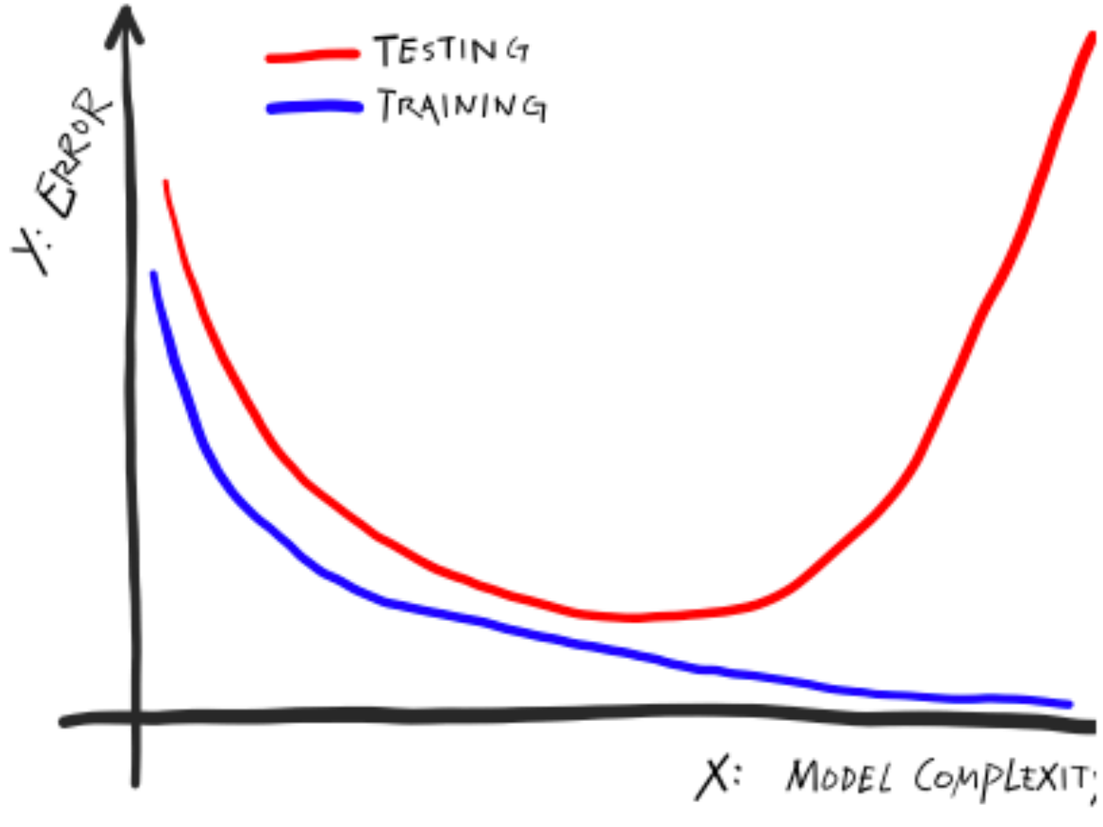
Resnets: Skiped-connections



Deeper is better



Regularization



Deep Learning

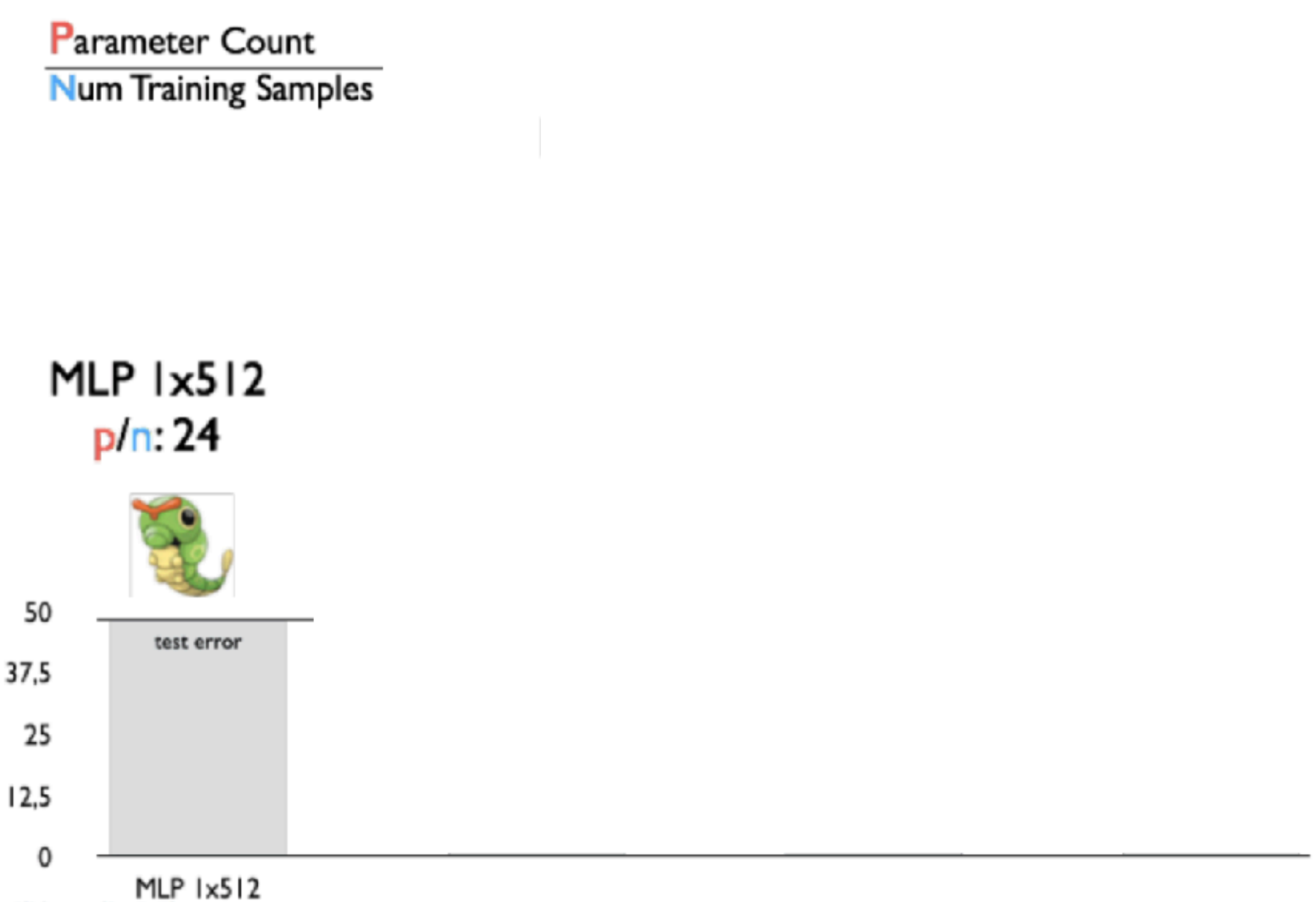
Slide credit: C. Zhang

Regularization

Parameter Count
Num Training Samples

MLP 1x512

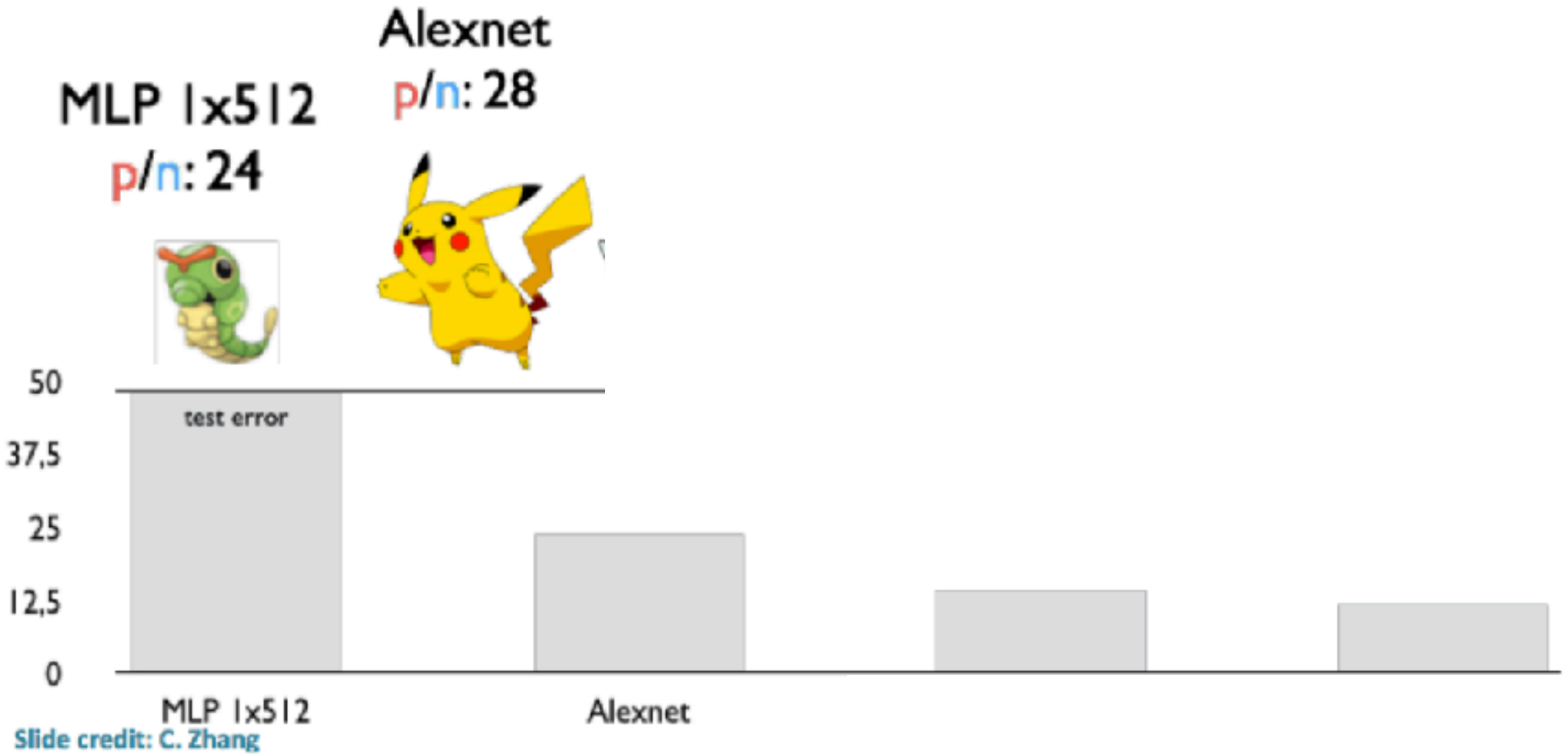
p/n: 24



Slide credit: C. Zhang

Regularization

Parameter Count
Num Training Samples

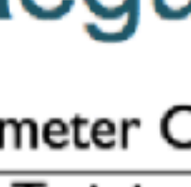


Slide credit: C. Zhang

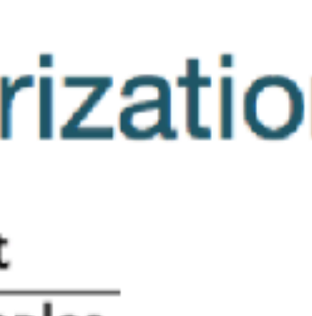
Regularization

Parameter Count
Num Training Samples

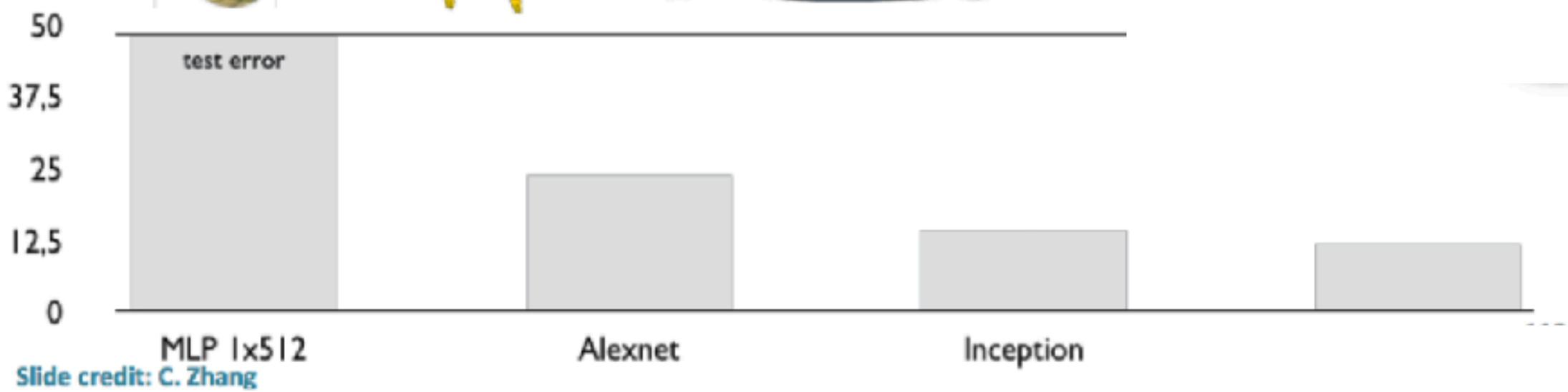
MLP 1x512
p/n: 24



Alexnet
p/n: 28



Inceptior
p/n: 33

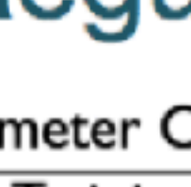


Slide credit: C. Zhang

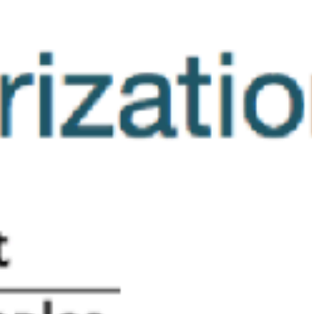
Regularization

Parameter Count
Num Training Samples

MLP 1x512
p/n: 24

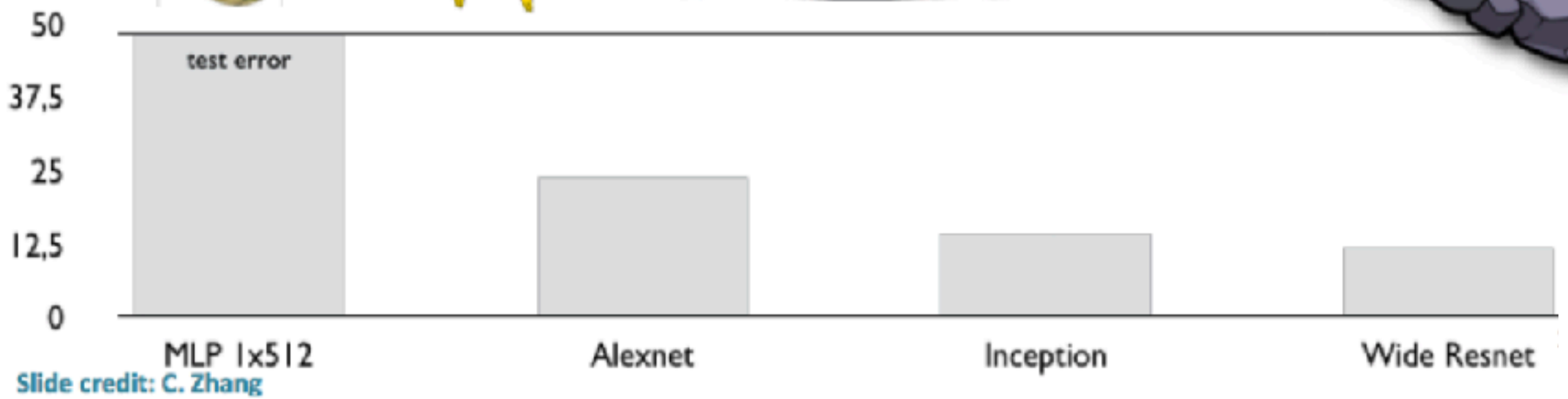


Alexnet
p/n: 28



Inception
p/n: 33

Wide Resnet
p/n: 179

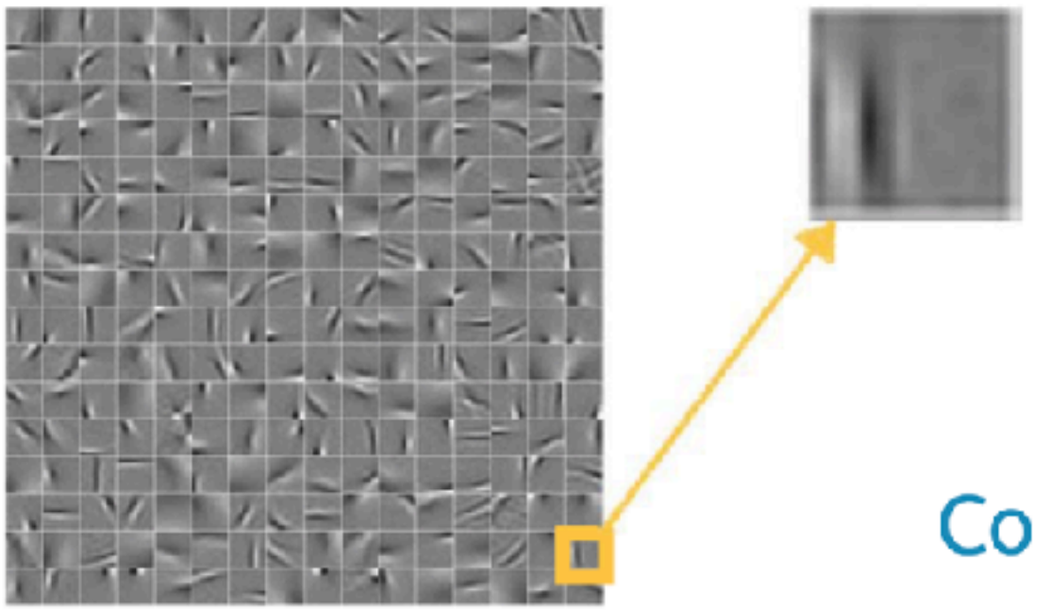


Slide credit: C. Zhang

What is learned in conv-nets?

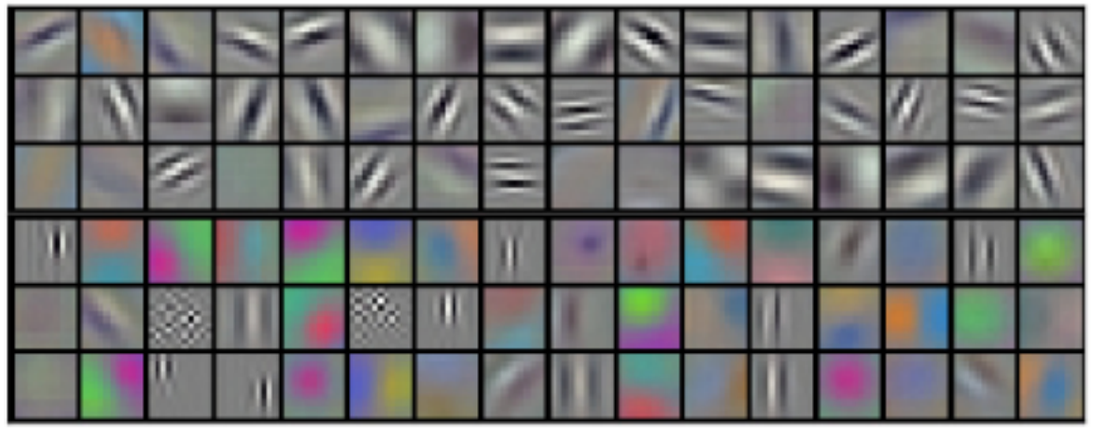
Convolutions

- A bank of 256 filters (learned from data)
- Each filter is 1d (it applies to a grayscale image)
- Each filter is 16 x 16 pixels

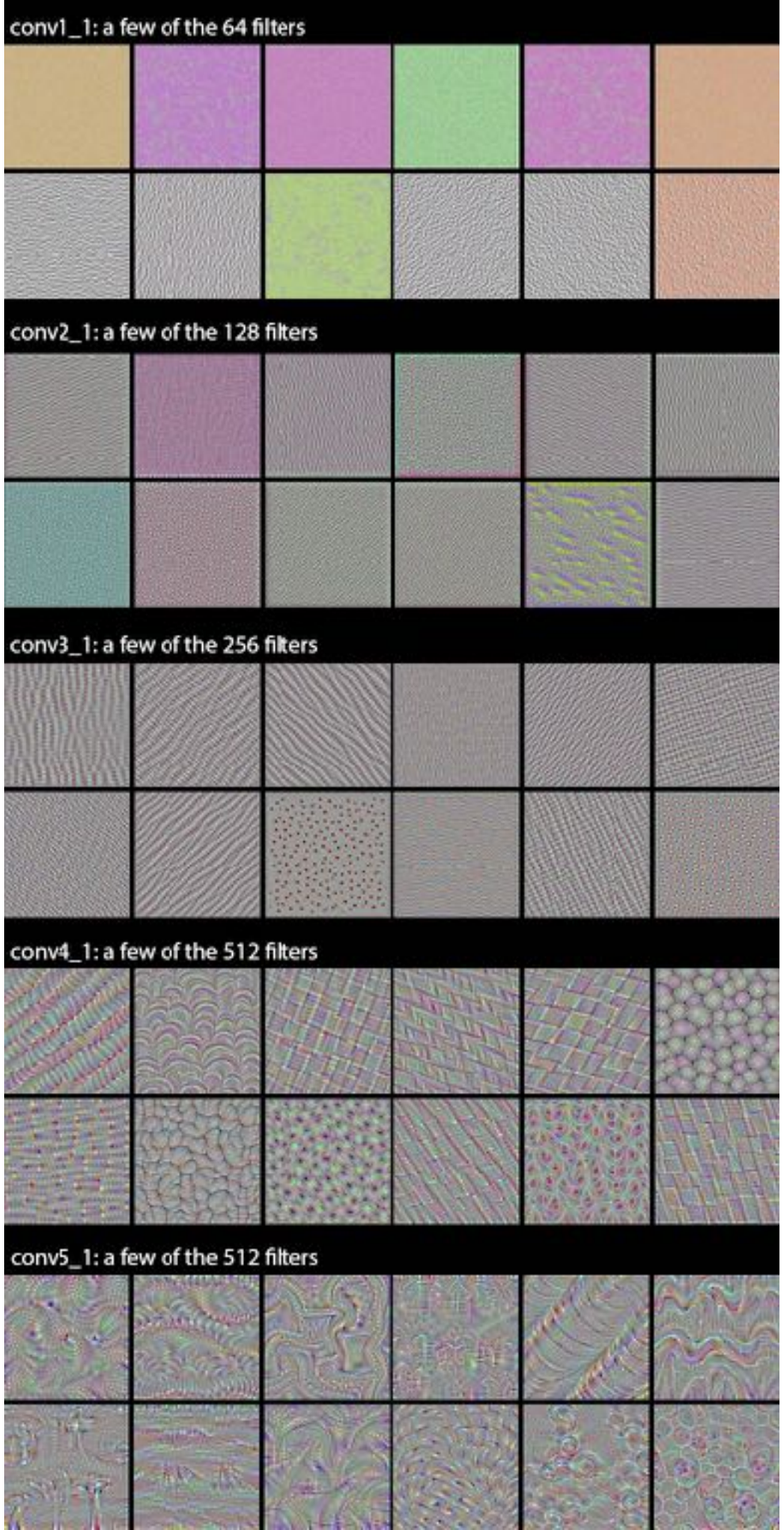


Convolutions

- A bank of 256 filters (learned from data)
- 3D filters for RGB inputs

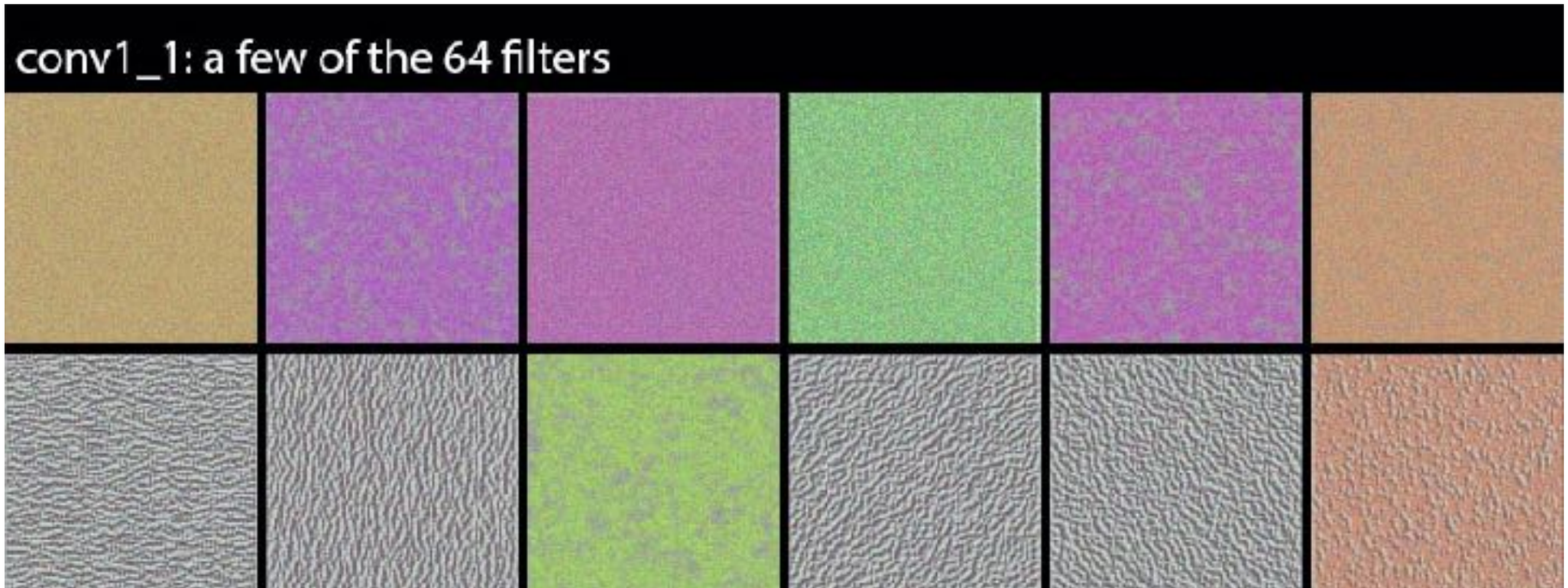


What sort of images maximise the activity for a given neutron in each layers?



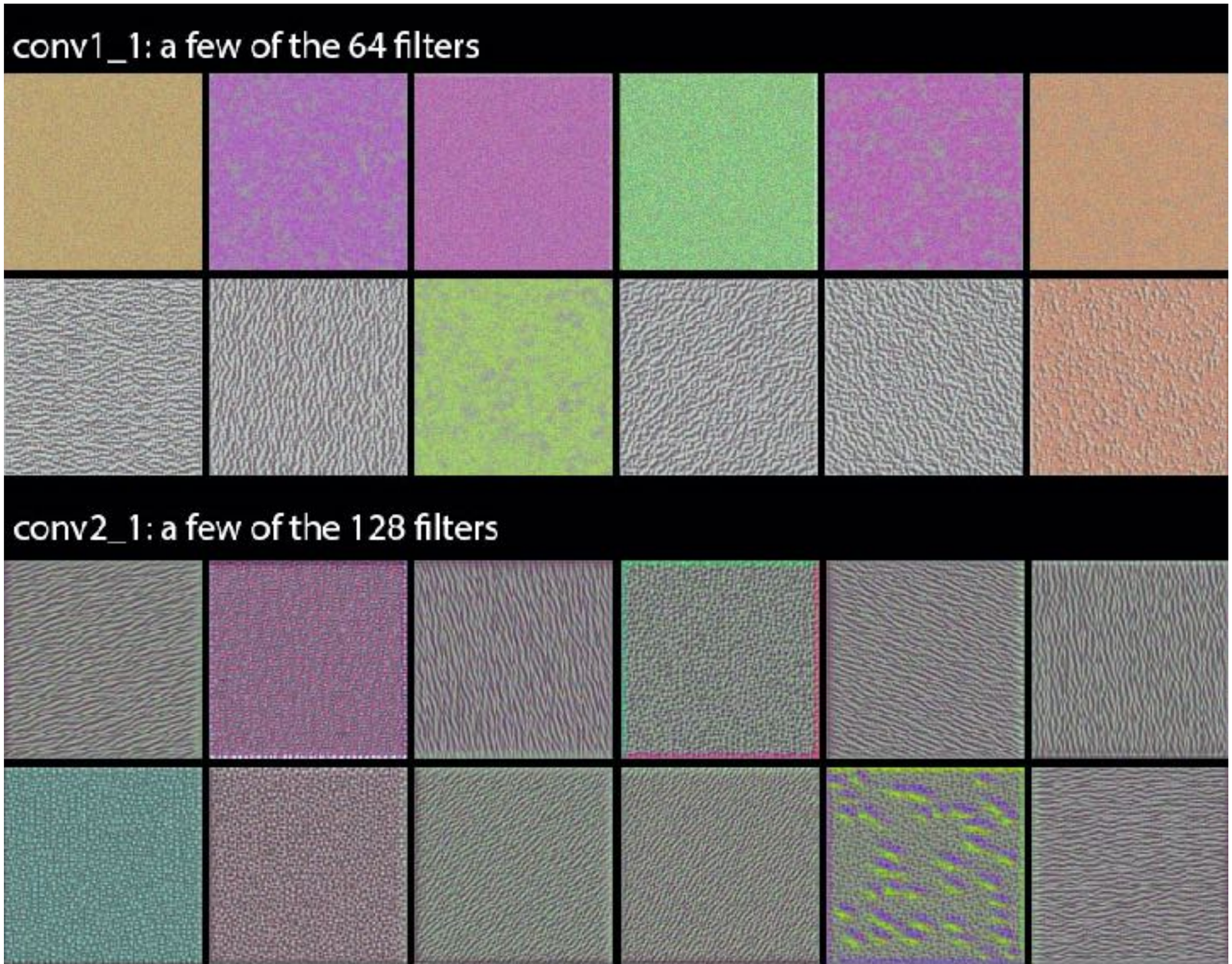
Slide credit: Francois Chollet

What sort of images maximise the activity for a given neutron in each layers?



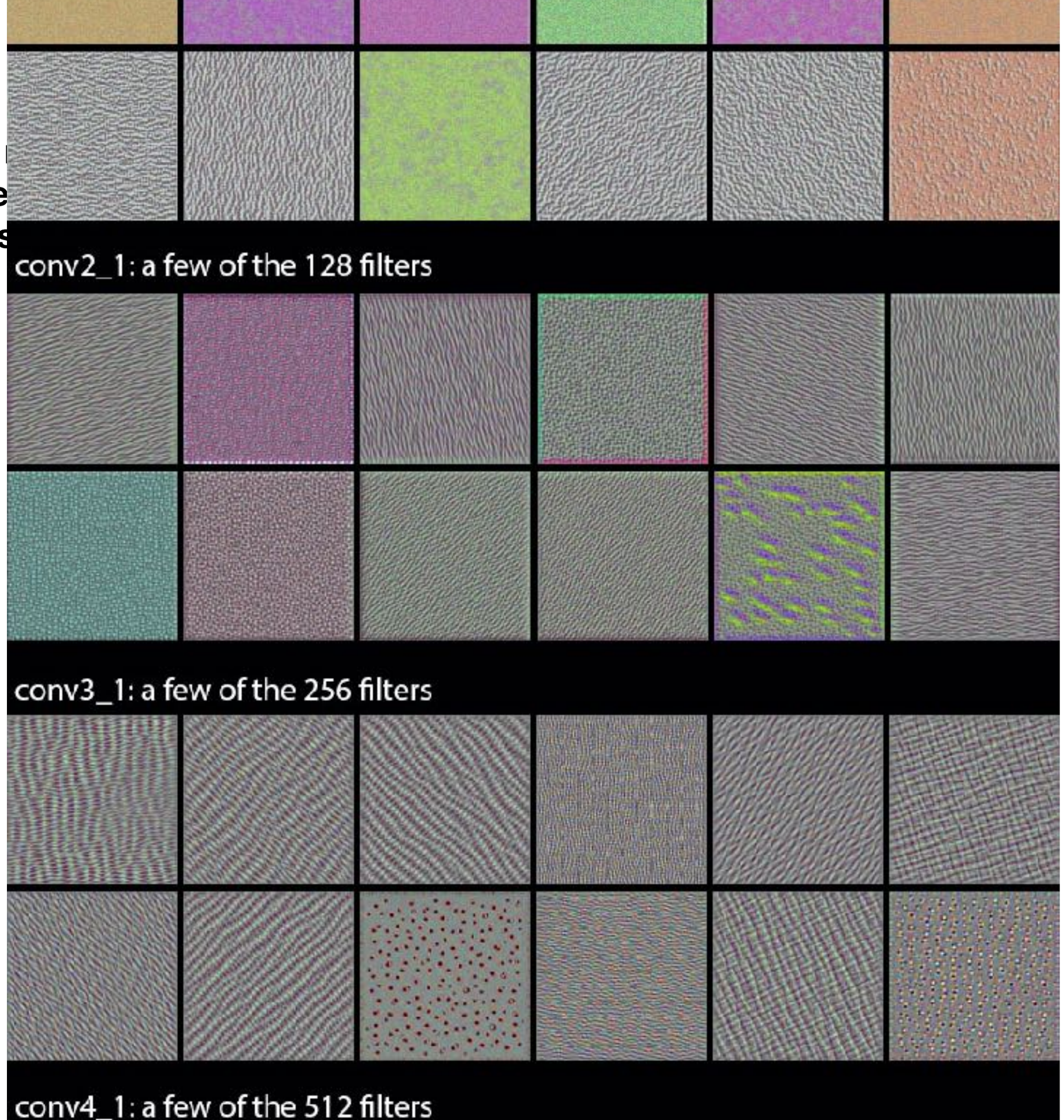
Slide credit: Francois Chollet

What sort of images maximise the activity for a given neutron in each layers?



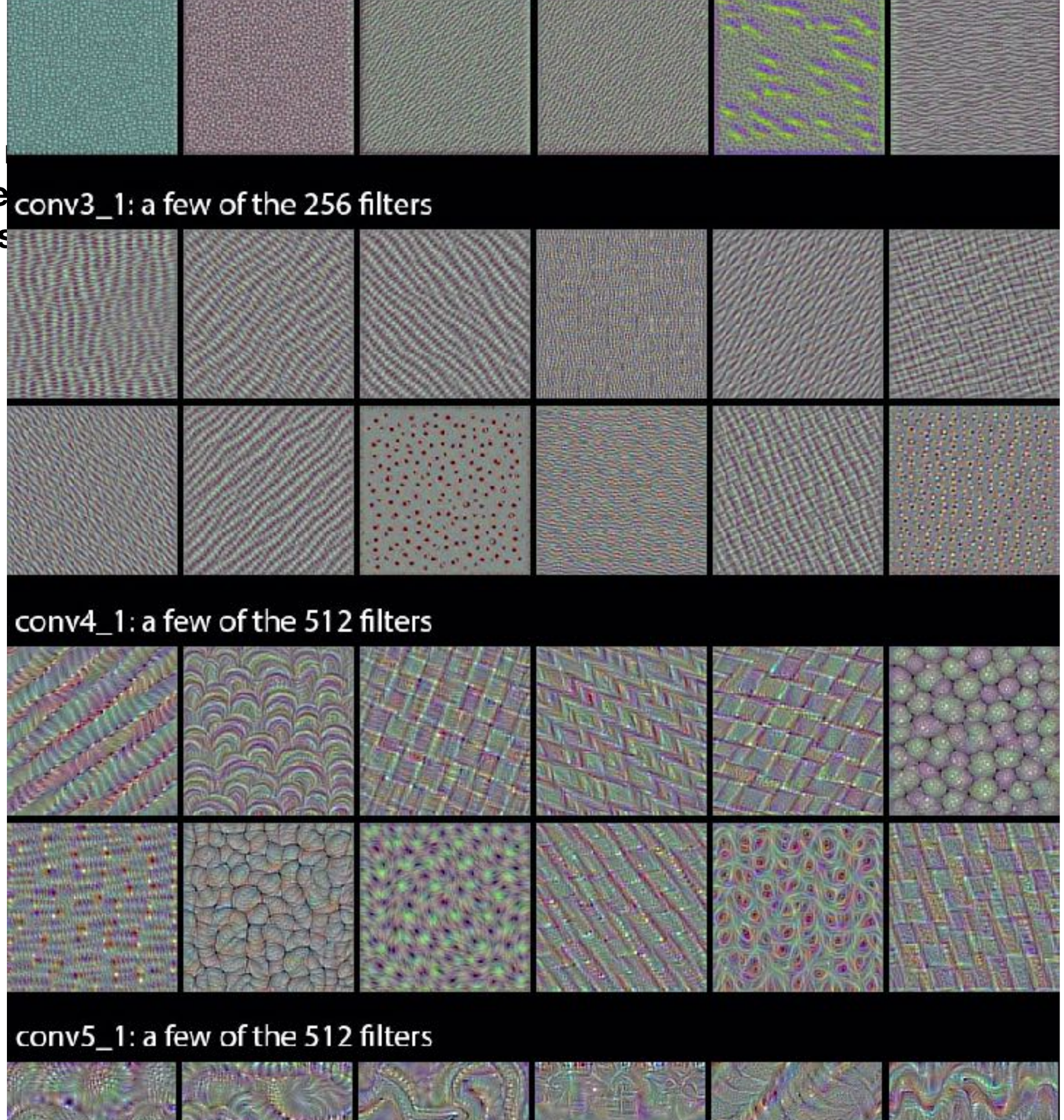
Slide credit: Francois Chollet

What sort of images
the activity for a give
in each layers



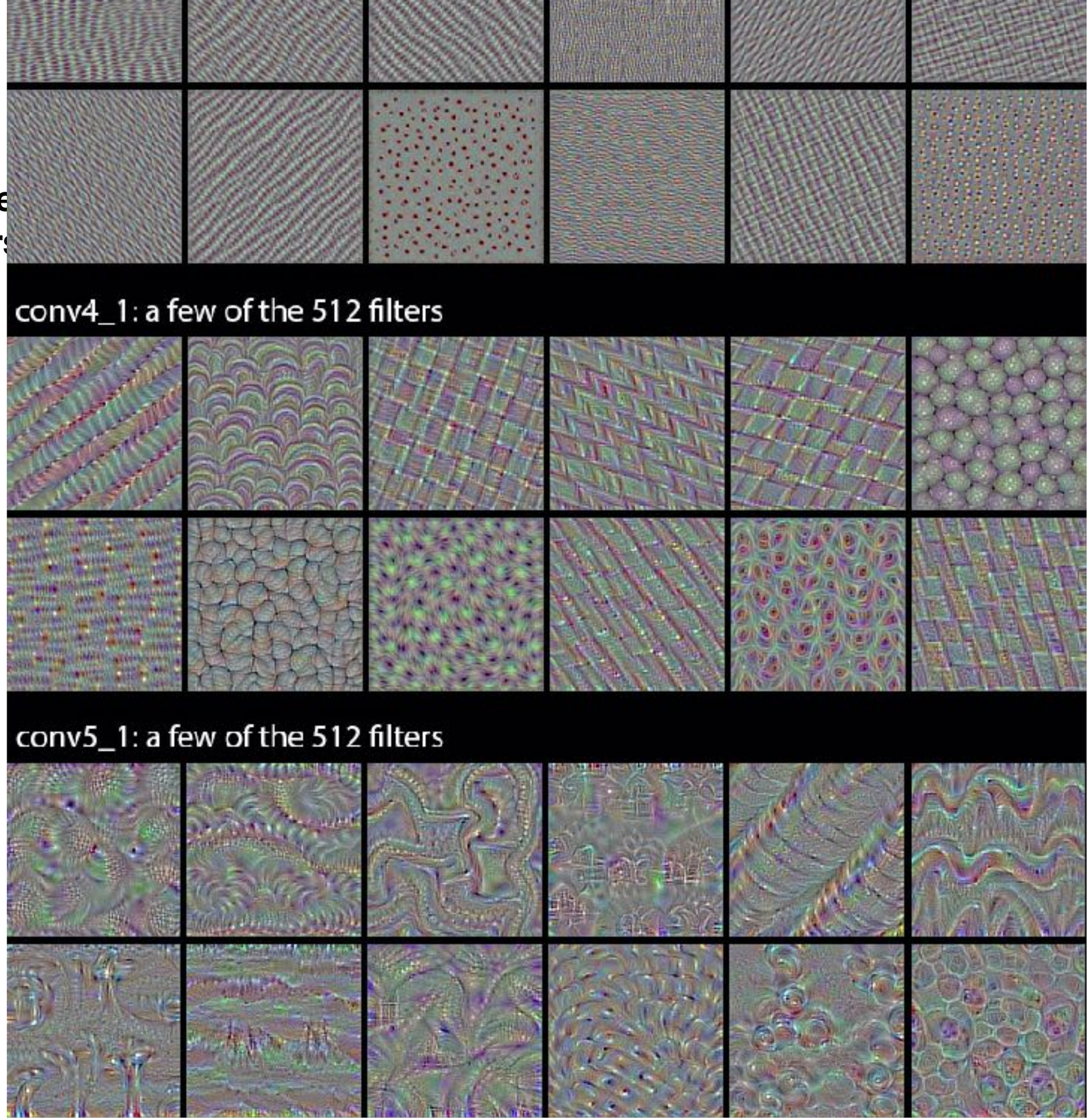
Slide credit: Francois Chollet

What sort of images
the activity for a give
in each layers



Slide credit: Francois Chollet

What sort of images
the activity for a give
in each layer



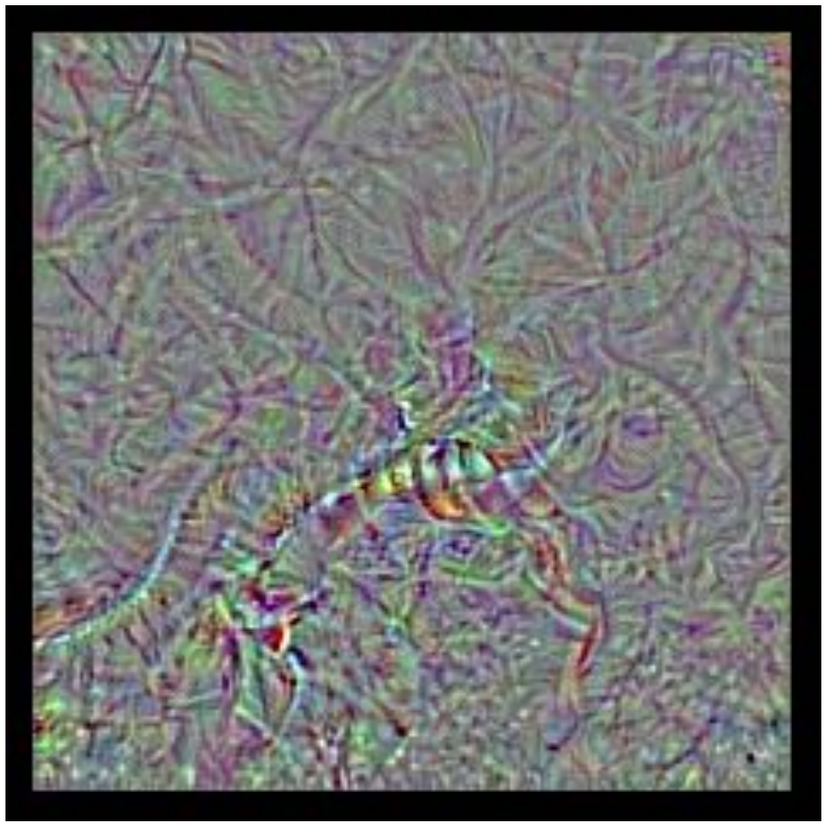
Slide credit: Francois Chollet

What sort of images maximise the activity for a given neutron in each layers?

What sort of images maximise the activity for <u>the final neutron</u> For a given category?

Let's try with a see-snake!
(vgg-16 trainde on imagenet With hundred categories)

This is a see-snake
« I am 99% positive! »



<u>Slide credit: Francois Chollet</u>

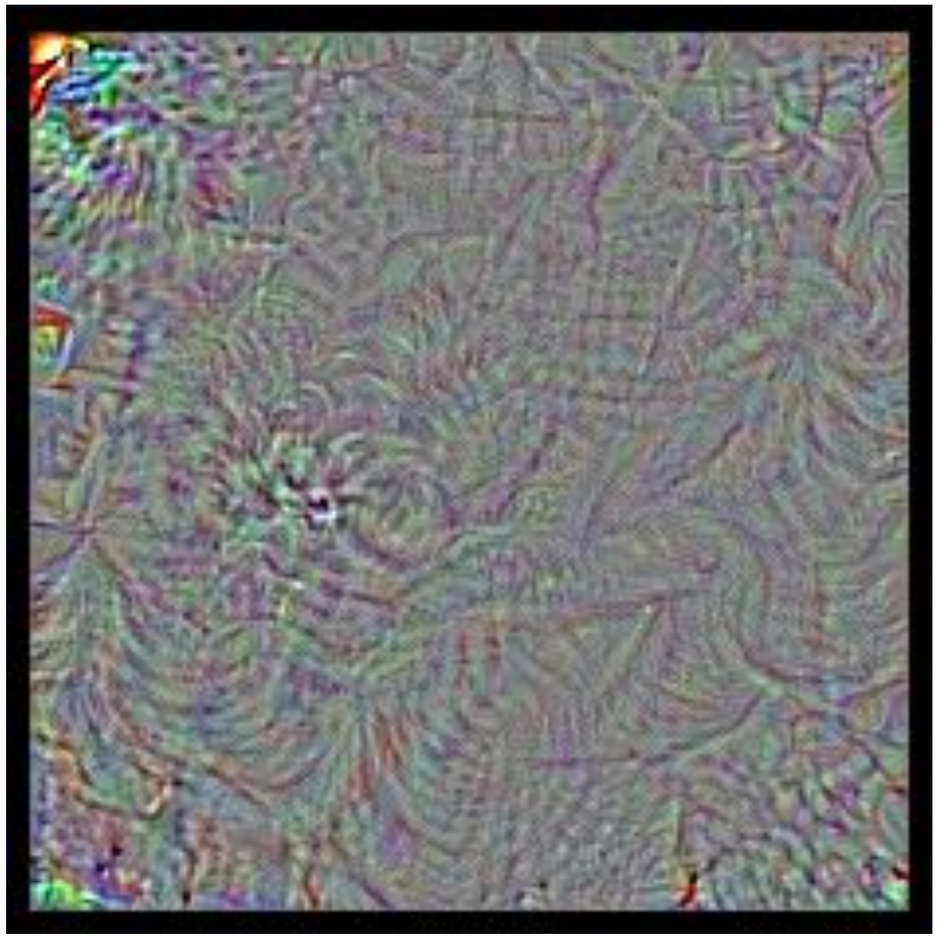
What sort of images maximise the activity for a given neutron in each layers?

What sort of images maximise the activity for <u>the final neutron</u> For a given category?

Let's try with a magpie!
(vgg-16 trainde on imagenet With hundred categories)

<u>Slide credit: Francois Chollet</u>

This is a magpie
« I am 99% positive! »



Think about this next time your hear some big-name CEO appear in the news to warn you against the existential threat posed by our recent advances in deep learning.

Another idea!

Occlusion sensitivity

- An approach to understand the behavior of a network is to look at the output of the network "around" an image.
- We can get a simple estimate of the importance of a part of the input image by computing the difference between:

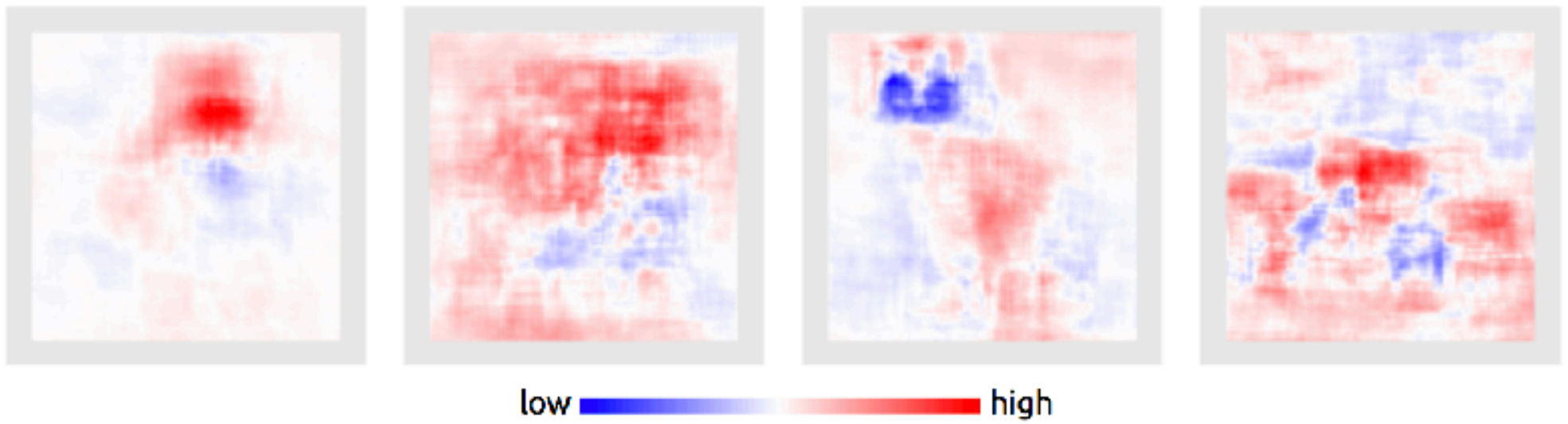
1. the value of the maximally responding output unit on the image, and
2. the value of the same unit with that part occluded.

This is computationally intensive since it requires as many forward passes as there are locations of the occlusion mask, ideally the number of pixels.

Occlusion sensitivity

Original images

Occlusion sensitivity, mask 32×32, stride of 2, VGG16



Computer Vision is Easy: Transfer Learning

(state of art result in few minutes)

Transfer Learning with CNNs

1. Train on ImageNet

2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end

3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

This is it!

Thank you and have fun with the notebooks

Transfer learning for cats vs. dogs classification

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
from keras.optimizers import adam
from keras import applications
from keras import backend as K
```

```
Using TensorFlow backend.
```

Loading the data

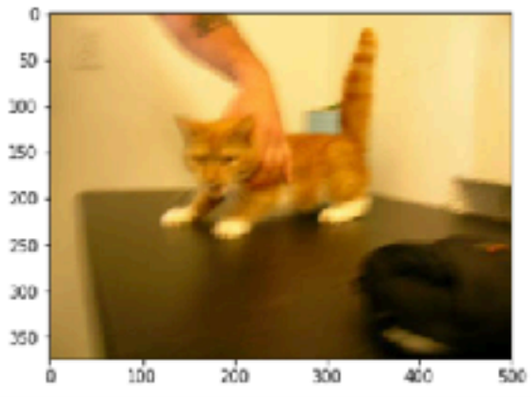
We are going to use the data from the Kaggle contest. The dataset is quite big: there are 25000 training samples, and 12500 test samples, all of different sizes. We are thus going to work with only a few of them: we are taking 3000 samples from the training set, and using 2000 of them for training and 1000 for validation.

BEFORE PROCEEDING: please download the smaller dataset here and extract inside the `lec4/mldata` folder.

Let's load the data - Keras has some nice routines for that purpose. We start by loading a single image, just to see how it looks like (try changing the filename to see some others). Note we are doing color images, and thus working with 3-dimensional arrays.

```
# Load image and transform it to a Numpy array
img = load_img('mldata/catsvsdogs/train/cats/cat.0.jpg')
x = img_to_array(img)
print("array size:", x.shape)

# Show image
plt.imshow(x / 255.)
```

```
array size: (374, 500, 3)
<matplotlib.image.AxesImage at 0x7fe862790390>
```



Classification on MNIST using Convolutional Ne (CNN)

In this notebook, we are going to use CNNs to perform classification on the MNIST dataset. MNIST in lecture 3 using a MultiLayer Perceptron (i.e. a dense network), so we are going to MLPs.

A nice description of CNNs can be found here.

Loading the data

As in the previous notebook, we begin by

- loading and normalizing the dataset
- performing the usual splitting in training/test set
- transforming the labels into one-hot-encoding type vectors.

```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import RMSprop
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

This is it!

Thank you and have fun with the notebooks

Learning to add number with a recurent neural ne

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural
relationships between elements in an input sequence. A good demonstration of
together using a mathematical operation such as a sum and outputting the res
sequence learning for performing addition" as in http://papers.nips.cc/paper
networks.pdf and effectivly, we indeed teach the machine to add numbers.

Let us see how this works. First, we create a python class (an object) that allows
categories, as well as to perform the reverse operation. This will save us a lot of ti

```
--
Iteration 93
Train on 4500 samples, validate on 500 samples
Epoch 1/1
4500/4500 [==============================] - 1s - loss: 0.0831 - acc:
val_acc: 0.9713
Q 34+14
T 48
☑ 48
---
Q 61+22
T 83
☑ 83
---
Q 21+97
T 118
☑ 118
---
Q 77+24
T 101
☑ 101
---
```

Recommender systems on the Movielens dataset

A recommender system is a machine learning algorithm that seeks to predict the rating a user would give to an item. One approach to the design of recommender systems that has been widely used is *collaborative filtering*.

This problem has become quite popular a few years ago with the *NetWix Prize*: an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings (and without the users or the films being identified except by numbers assigned for the contest -- this is different from so-called content-based methods).

First, let us import the usual suspects

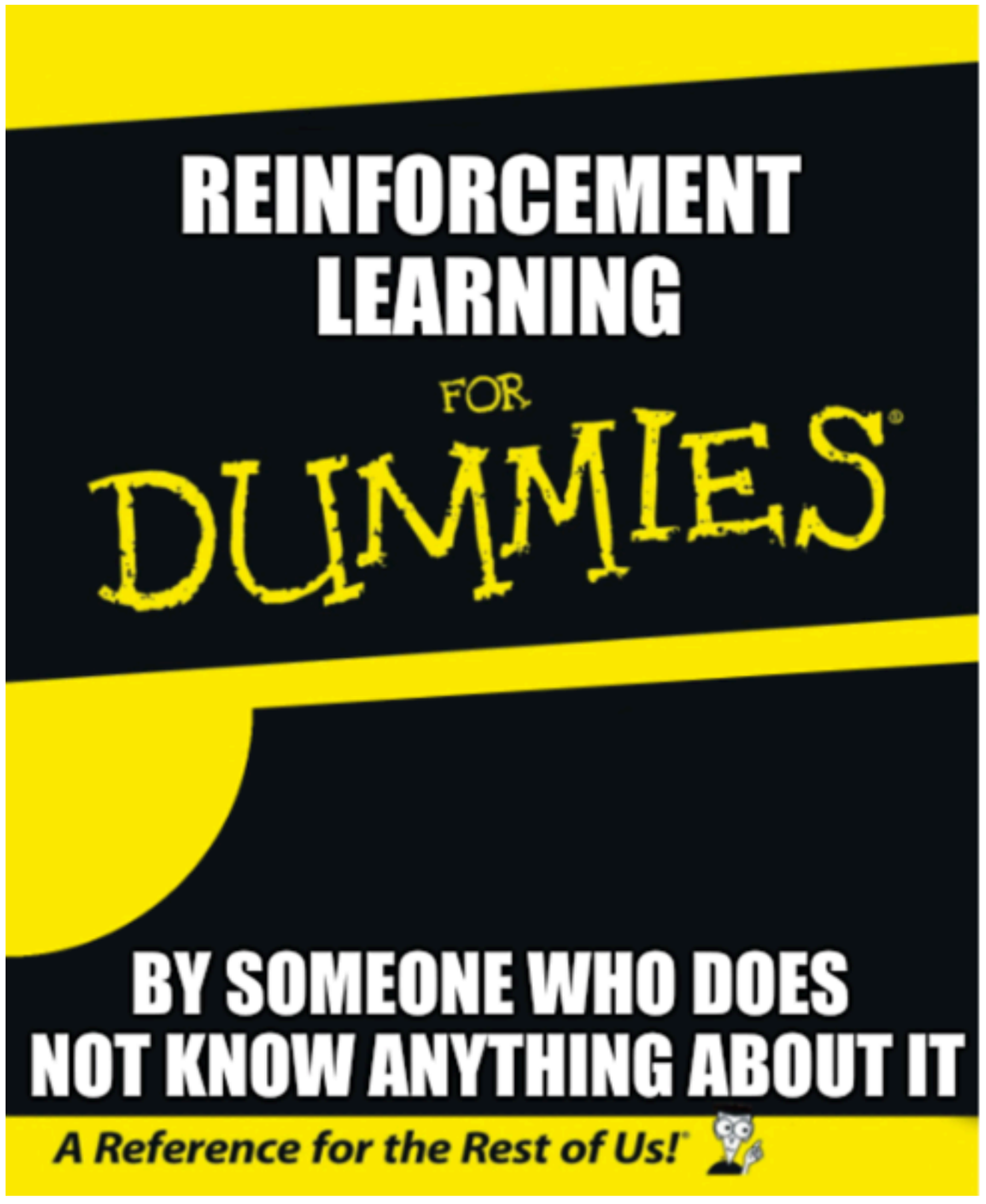
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import h5py

import keras
from keras.models import Model
from keras.layers import Input, Embedding, Flatten, dot, add
from keras.regularizers import l2
from keras.optimizers import adam
```

Moving to neural nets

Now that we have tried to be clever, using the kind of techniques that were used
generic powerful techniques, like... a neural network! Indeed, rather than creatin
factorization with bias), it's often both easier and more accurate to use a standard

Let's try it! Here, we simply concatenate the user and movie embeddings into a sin

```
from keras.layers import concatenate, Dense, Dropout

# Generate embeddings and concatenate them
user_input = Input(shape=(1,), dtype='int64', name='user')
U = Embedding(n_users, n_factors, input_length=1, embeddings_
movie_input = Input(shape=(1,), dtype='int64', name='movie')
V = Embedding(n_movies, n_factors, input_length=1, embeddings_
Y = concatenate([U, V])
Y_r = Flatten()(Y)

# Specify neural network architecture
Y_nn = Dropout(0.3)(Y_r)
Y_nn = Dense(70, activation='relu')(Y_nn)
Y_nn = Dropout(0.75)(Y_nn)
Y_nn = Dense(1)(Y_nn)
nn = Model([user_input, movie_input], Y_nn)

nn.summary()
```


REINFORCEMENT
LEARNING
FOR
DUMMIES
BY SOMEONE WHO DOES
NOT KNOW ANYTHING ABOUT IT
A Reference for the Rest of Us!

This tutorial


Playing Atari with Deep Reinforcement Learning

Abstract

1 Introduction


nature
International journal of science

Article | Published: 18 October 2017

Mastering the game of Go without human knowledge

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel & Demis Hassabis

Nature **550**, 354–359 (19 October 2017) | Download Citation

Abstract

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by

DeepMind

RL 101

States & Actions

Rewards, Policy & Value

States & Actions

Curent state of the system

Actions of the player

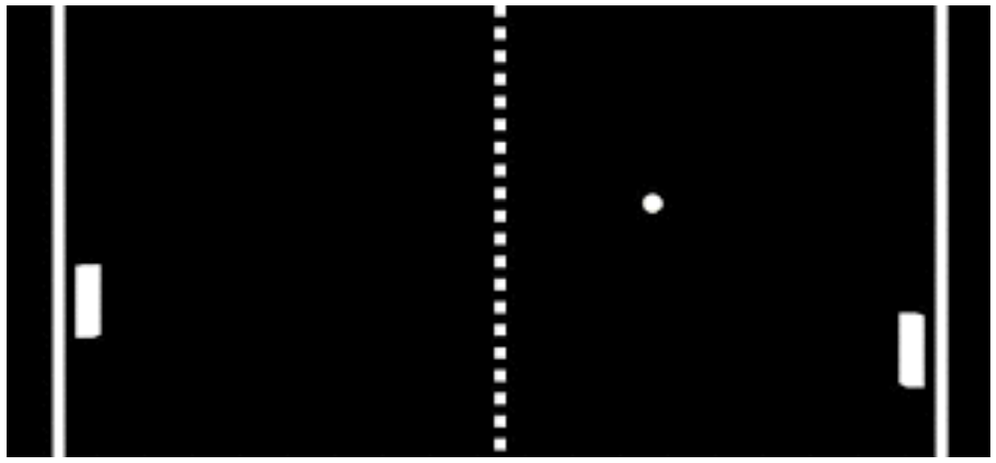
States & Actions

Curent state of the system

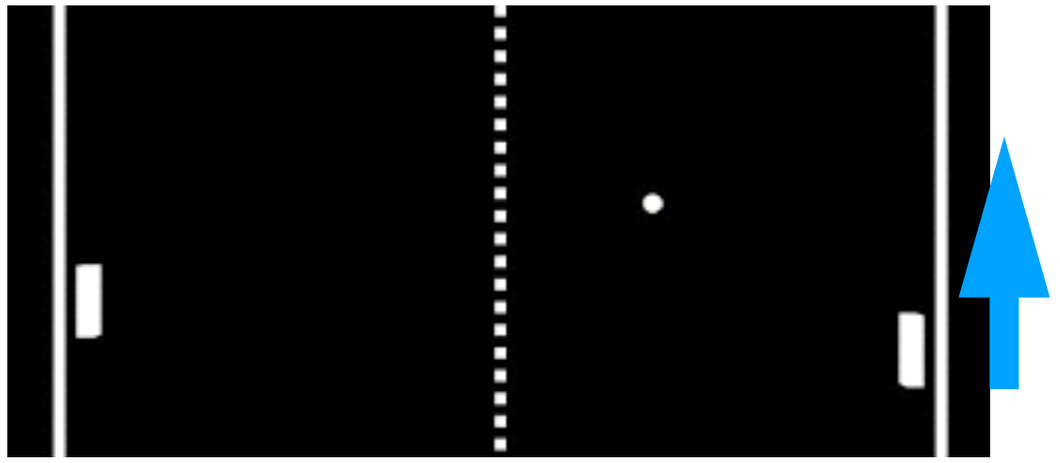
Actions of the player

States & Actions

Curent state of the system



Actions of the player



States & Actions

Curent state of the system

Actions of the player

Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

the "Value" of the position includes future rewards

$$V(\{a_t, s_t\}) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$$

Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

the "Value" of the position includes future rewards

$$V(\{a_t, s_t\}) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$$

More precisely, it includes future rewards under perfect play

$$V(\{a_t, s_t\}) = r(t) + \sum_{t=\tau+1}^{\infty} \gamma^{\tau} r^{\mathrm{BEST}}(\tau)$$

Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

the "Value" of the position includes future rewards

$$V(\{a_t, s_t\}) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$$

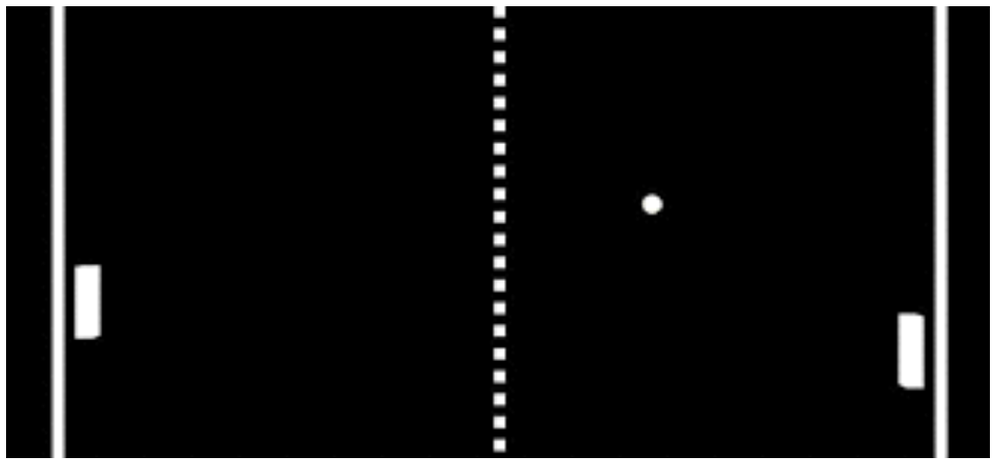
Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

the "Value" of the position includes future rewards

$$V(\{a_t, s_t\}) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$$



Score = 1 if ball still in the game

Rewards & value

Immediate reward **total Value (including the future)**

Reward r^t = score at time t as a result of the action

the "Value" of the position includes future rewards

$$V(\{a_t, s_t\}) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$$

Score = ???

Policy

What action are you playing given the state ?

$$\pi(a \mid s) \equiv \mathbb{P}(a \mid s)$$

Policy $\pi(a \mid s) \equiv \mathbb{P}(a \mid s)$

Reward r^t = score at time t as a result of the action

Value $V(\{a_t, s_t\}) = r(t) + \sum_{t=\tau+1}^{\infty} \gamma^{\tau} r^{\mathrm{BEST}}(\tau) = \sum_{\tau=t}^{\infty} \gamma^{\tau} r(\tau)$

Q-Learning

Imagine one has access to a “cheat” table Q *(s,a) giving,
for every state of the system s*, the <u>values</u> *for each action a under perfect play

$$Q^*(s,a) = r(a) + \sum_{t=1}^{\infty} \gamma^t r^{\text{best|a}}(t)$$

Q-Learning

Imagine one has access to a “cheat” table Q *(s,a) giving,
for every state of the system s*, the <u>values</u> *for each action a under perfect play

$$Q^*(s,a) = r(a) + \sum_{t=1}^{\infty} \gamma^t r^{\text{best|a}}(t)$$

(s) -> action a|s —> (s’)

$$Q^*(s,a) = r(a) + \sum_{t=1}^{\infty} \gamma^t r^{\text{best|a}}(t) = r(a) + \gamma V(s')$$

Q-Learning

Imagine one has access to a "cheat" table Q *(s,a) giving,
for every state of the system s*, the <u>values</u> *for each action a under perfect play

$$Q^*(s,a) = r(a) + \sum_{t=1}^{\infty} \gamma^t r^{\text{best|a}}(t) = r(a) + \gamma V(s')$$

Q-Learning

Imagine one has access to a "cheat" table Q *(s,a) giving,
for every state of the system s*, the values *for each action a under perfect play

$$Q^*(s,a) = r(a) + \sum_{t=1}^{\infty} \gamma^t r^{\text{best|a}}(t) = r(a) + \gamma V(s')$$

Then the optimal policy is

$$\pi^*(a \mid s) = \delta(a, \text{argmax}_{\text{a}}(Q^*(s,a)))$$

How to find the Q-table?

(i) Start with an estimated one, possibility random $Q^{t=0}(s, a)$

(ii) Iterate the following equation:

$$Q^{t+1}(s, a) = r_a(t) + \gamma \max_{a^{t+1}} \left[Q^t(s^{t+1}, a^{t+1}) \right]$$

(iii) Claim: the table will converge to Q*

Proof!

$$Q^{t+1}(s,a) = r_a(t) + \gamma \max_{a^{t+1}} \left[Q^t(s^{t+1}, a^{t+1}) \right]$$

(a) Q* is a fixed point!

$$Q^*(s,a) = r_a + \sum_{t=1}^{\infty} \gamma^t r^{\text{best}|a}(t) = r(a) + \gamma V(s')$$

$$Q^*(s,a) = r_a + \gamma \max_{a'} \left[Q^*(s' \mid a, a') \right]$$

Proof!

$$Q^{t+1}(s,a) = r_a(t) + \gamma \max_{a^{t+1}} \left[Q^t(s^{t+1}, a^{t+1}) \right]$$

(b) the iteration is contractive

$$Q_1^{t+1}(s,a) - Q_2^{t+1}(s,a) = r(s,a) + \gamma \max_{a^{t+1}} \left[Q_1^t(s^{t+1}, a^{t+1}) \right] - r(s,a) - \gamma \max_{a^{t+1}} \left[Q_2^t(s^{t+1}, a^{t+1}) \right]$$

$$Q_1^{t+1}(s,a) - Q_2^{t+1}(s,a) = \gamma \left(\max_{a^{t+1}} \left[Q_1^t(s^{t+1}, a^{t+1}) \right] - \max_{a^{t+1}} \left[Q_2^t(s^{t+1}, a^{t+1}) \right] \right)$$

$$Q_1^{t+1}(s,a) - Q_2^{t+1}(s,a) \leq \gamma \max_{a',s'} \left(\left[Q_1^t(s',a') \right] - \left[Q_2^t(s',a') \right] \right)$$

$$\| Q_1^{t+1}(s,a) - Q_2^{t+1}(s,a) \|_\infty \leq \gamma \| \left[Q_1^t(s',a') \right] - \left[Q_2^t(s',a') \right] \|_\infty$$

Proof!

$$Q^{t+1}(s,a) = r_a(t) + \gamma \max_{a^{t+1}} \left[Q^t(s^{t+1}, a^{t+1}) \right]$$

(a) Q* is a fixed point!

(b) the iteration is contractive

Convergence to Q*

Banach fixed-point theorem

From Wikipedia, the free encyclopedia

In mathematics, the **Banach–Caccioppoli fixed-point theorem** (also known as the **contraction mapping theorem** or **contraction mapping principle**) is an important tool in the theory of metric spaces; it guarantees the existence and uniqueness of fixed points of certain self-maps of metric spaces, and provides a constructive method to find those fixed points. It can be understood as an abstract formulation of Picard's method of successive approximations.[1] The theorem is named after Stefan Banach (1892–1945) and Renato Caccioppoli (1904–1959), and was first stated by Banach in 1922. Caccioppoli independently proved the theorem in 1931.[2]

Bellman equation

$$Q^{t+1}(s,a) = r(t) + \gamma \max_{a^{t+1}} \left[Q^{t}(s^{t+1}, a^{t+1}) \right]$$

Bellman equation

$$Q^{t+1}(s,a) = r(t) + \gamma \max_{a^{t+1}} \left[Q^{t}(s^{t+1}, a^{t+1}) \right]$$

Mostly **useless** in practice: we can never apply the Bellman operator **exactly**, except for very small problems where we can easily iterate over all (s,a)

Instead try out randomly some actions & iterate a damped version of the algorithm

$$Q^{t+1}(s,\ a) = (1-\delta) * Q^{t}(s,\ a) + \delta(R(a) * \gamma \max_{a'} Q^{*}(s'|a,\ a'))$$

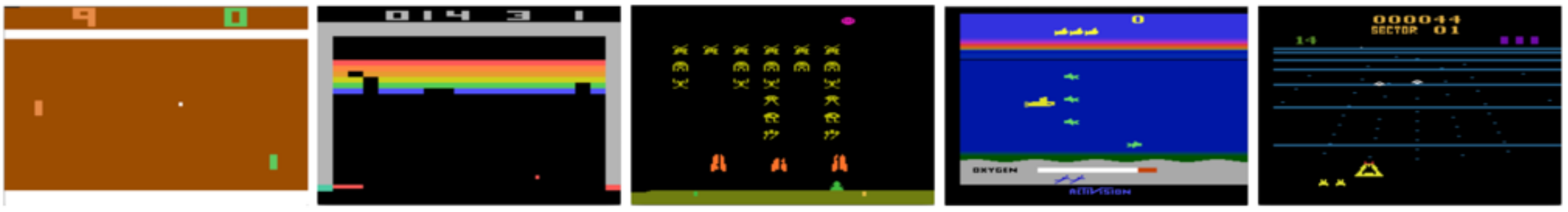
How to try "randomly ?"

Follow current policy (exploitation "in-policy") p=1-ε

Try random moves (exploration "out-policy") p=ε

Atari games

Reinforcement leanring



Atari games

Reinforcement leanring





Ask a computer to learn how to play breakout such that the <u>score is good at the end of the game</u>

Rules: The computer “see” the image & control the joystick.
Asked to maximise the end score

Reinforcement learning

Action a^t = action of the joystick

Reward r^t = score at time t as a result of the action

Q-learning

Imagine one has access to a table Q *(s,a) giving, *for every state of the system s*, the (weighted) product of all best score in the future *for each action a*

$$Q^*(s,a) = \sum_{t=0}^{\infty} \gamma^t r^{\text{best}}(t)$$

Reinforcement learning

Action a^t = action of the joystick

Reward r^t = score at time t as a result of the action

Q-learning

Imagine one has access to a table Q *(s,a) giving, *for every state of the system s*, the (weighted) product of all best score in the future *for each action a*

$$Q^*(s,a) = \sum_{t=0}^{\infty} \gamma^t r^{\text{best}}(t)$$

(Bellmann equation)

$$Q^*(s,a) = r(t=0) + \gamma \max_{a^{t+1}} \left[Q^*(s^{t+1}, a^{t+1}) \right]$$

Reinforcement learning

Action a^t = action of the joystick

Reward r^t = score at time t as a result of the action

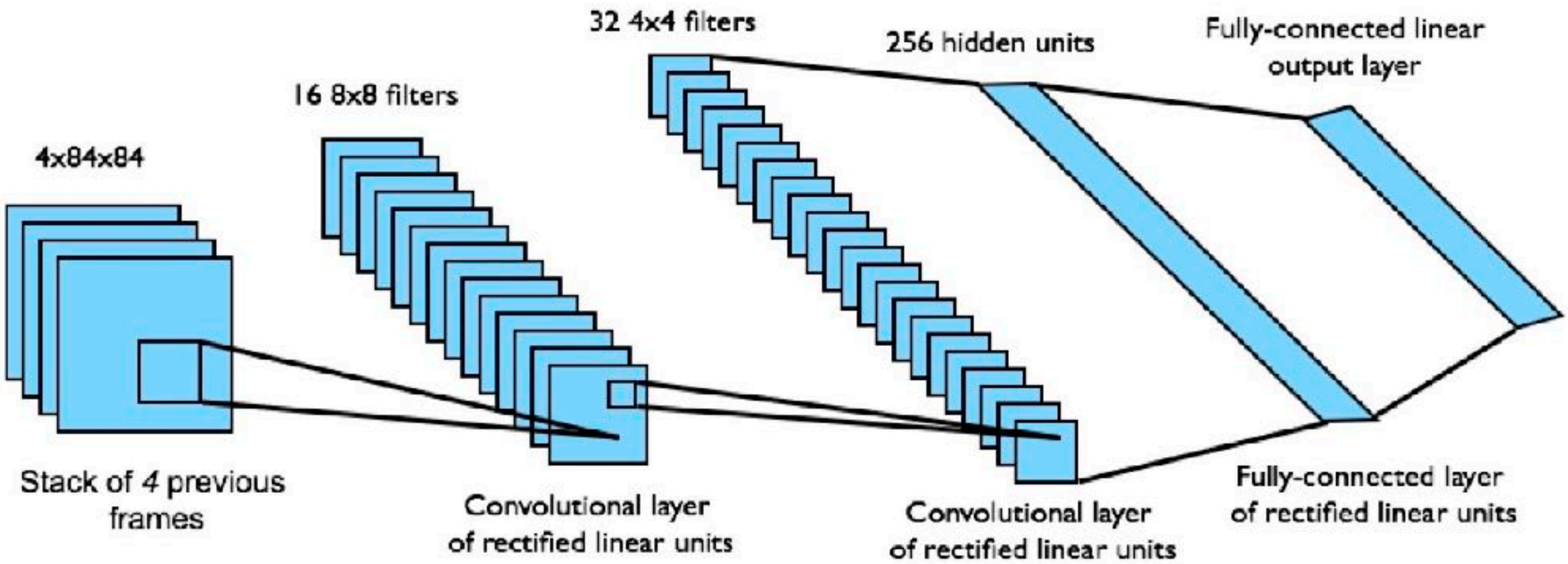
Q-learning

Given an estimate of the ideal Q(s,a), define the cost

$$\text{Cost} = \sum_{s} \left(Q(s, a) - r(t) - \gamma \max_{a'} \left[Q(s^{t+1}, a^{t+1}) \right] \right)^2$$

Learn the function Q(s,a) by gradient descent

Represent Q(s,a) as a convnet



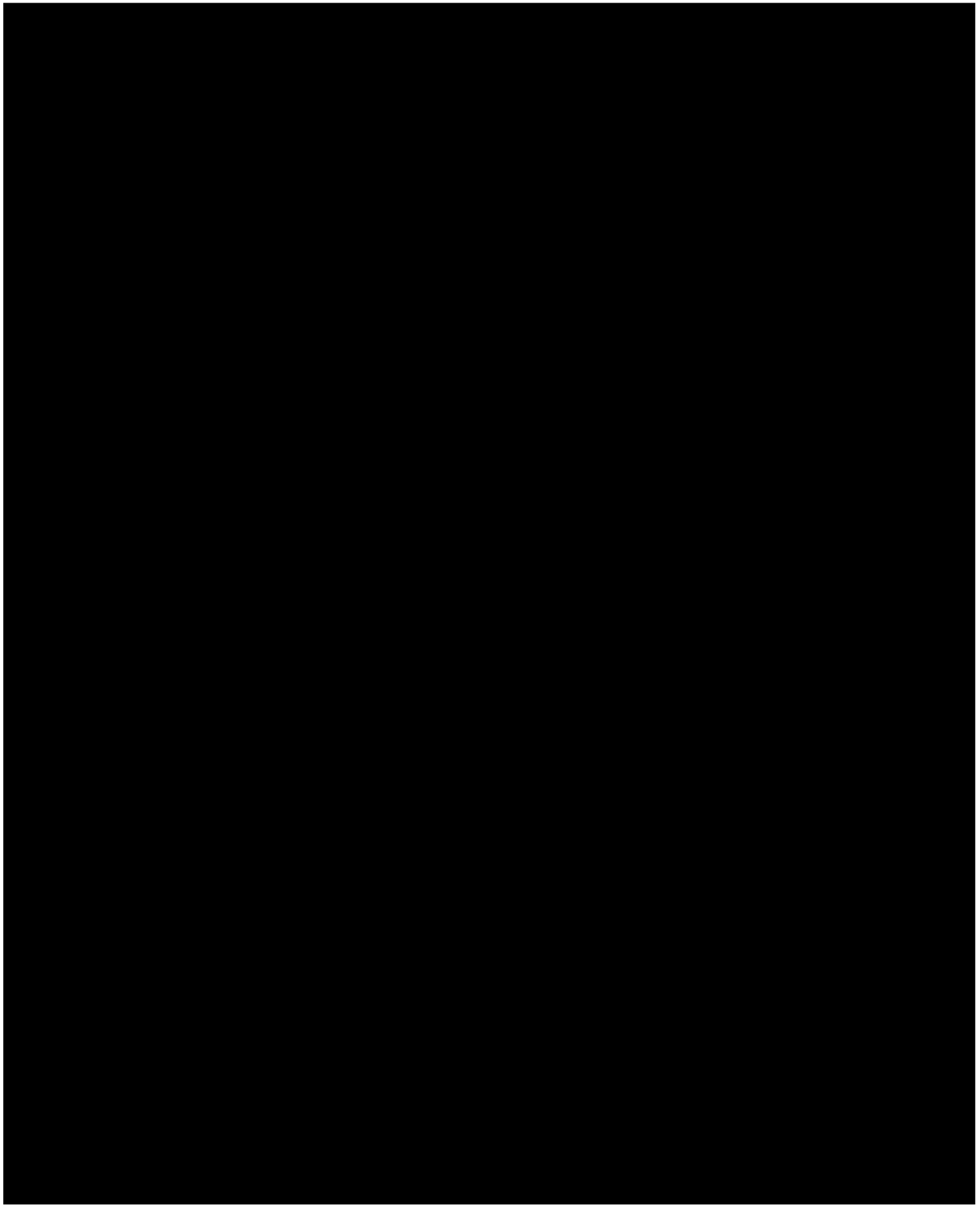
Experience Replay

Create a “replay buffer” that stores current state of the games.
At each step: sample a small batch of “experience” to feed our neural network.

Advantage: more efficient sampling go the q-table, and intuitively more pleasing

Ex: if we are in the first level and then the second (which is totally different), our agent can forget how to behave in the first level.

By learning how to play on water level, our agent will forget how to behave on the first level




Before training
peaceful swimming

Policy Gradients

Optimize directly the policy π(a|s) without using the Q-table
Advantage : can be use for continuous variable, direct optimization

$$J(\theta) = \mathbb{E}_{\pi_\theta, \vec{s}}\left[V(\vec{s})\right]$$

Compute the gradient of J !

REINFORCE

$$J(\theta) = \mathbb{E}[f(X)] = \int_x f(x) p_{\theta}(x)\, dx$$

$$\nabla_\theta J(\theta) = \int_x f(x) \nabla_\theta p_{\theta}(x)\, dx = \int_x f(x) p(x) \nabla_\theta \log(p_{\theta}(x))\, dx = \mathbb{E}[f(x) \nabla_\theta \log(p_{\theta}(x))$$

Now, assume that we are dealing with trajectories generated by a Markov chain:

$$p_{\theta}(\tau) = p(s_0) \prod_{i=0}^{\infty} \pi_{\theta}(a_i|s_i) p(s_{i+1}|s_i)$$

$$\log p_{\theta}(\tau) = \log p(s_0) + \sum_i (\log \pi_{\theta}(a_i|s_i) + \log p(s_{i+1}|s_i))$$

While the transition π depends on theta, the actual dynamics does not 8

$$g = \nabla_\theta J(\theta) = \mathbb{E}\left[f(\tau) \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) \right]$$

REINFORCE

$$g = \nabla_\theta J(\theta) = \mathbb{E}\left[f(\tau) \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) \right]$$

Approximate the expectation by an empirical sum over many “plays”

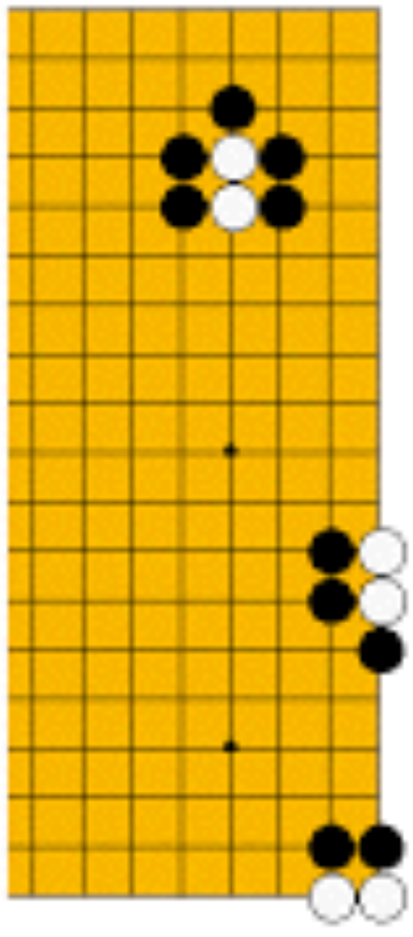
$$g = \nabla_\theta J(\theta) \approx \frac{1}{P} \sum_{p=1}^{P} f(\tau) \sum_i \nabla_\theta \log \pi_\theta(a_i \,|\, s_i)$$

Go

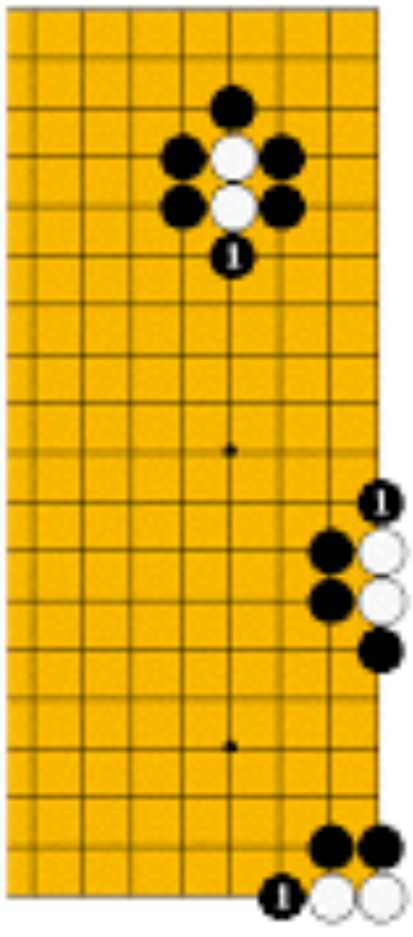
2,500 years old: Oldest game still play today

One type of piece, one type of move

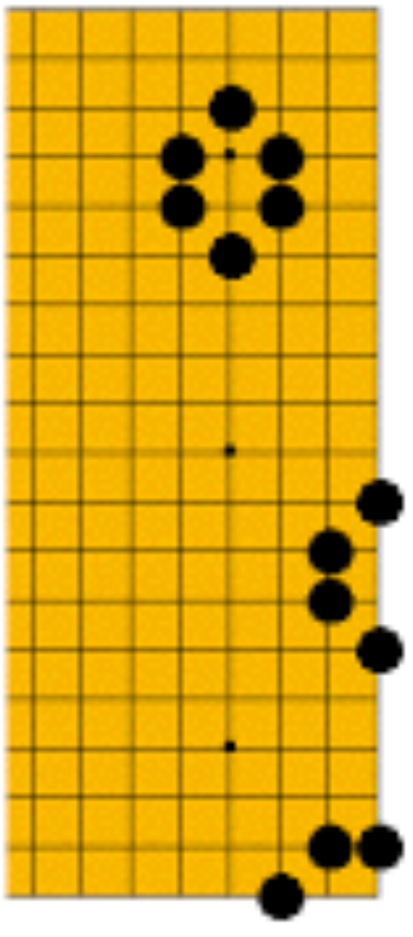
Once placed on the board, stones may not be moved, but stones are removed from the board if "captured". Capture happens when a stone or group of stones is surrounded by opposing stones on all orthogonally-adjacent points



Dia. 18

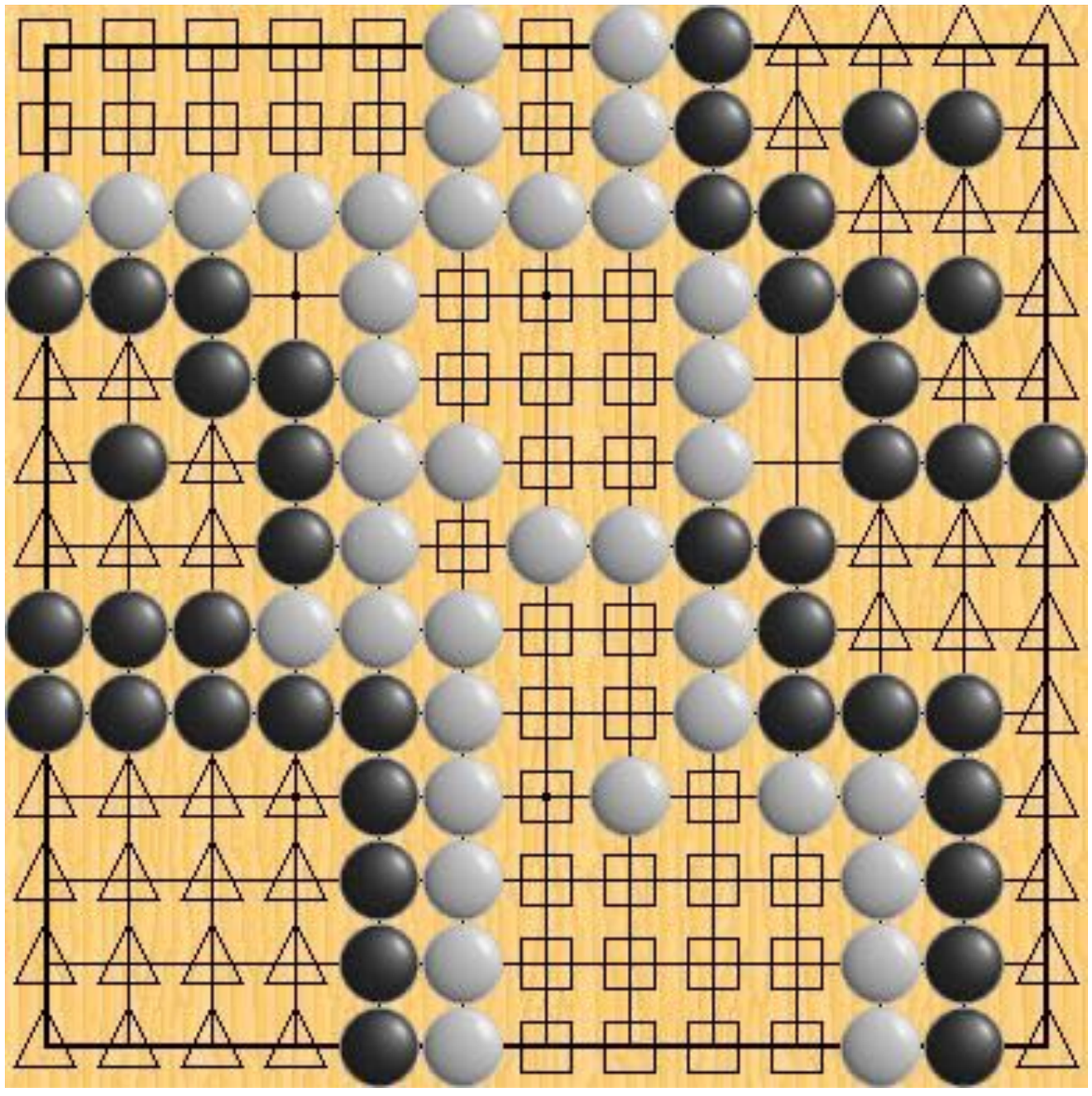


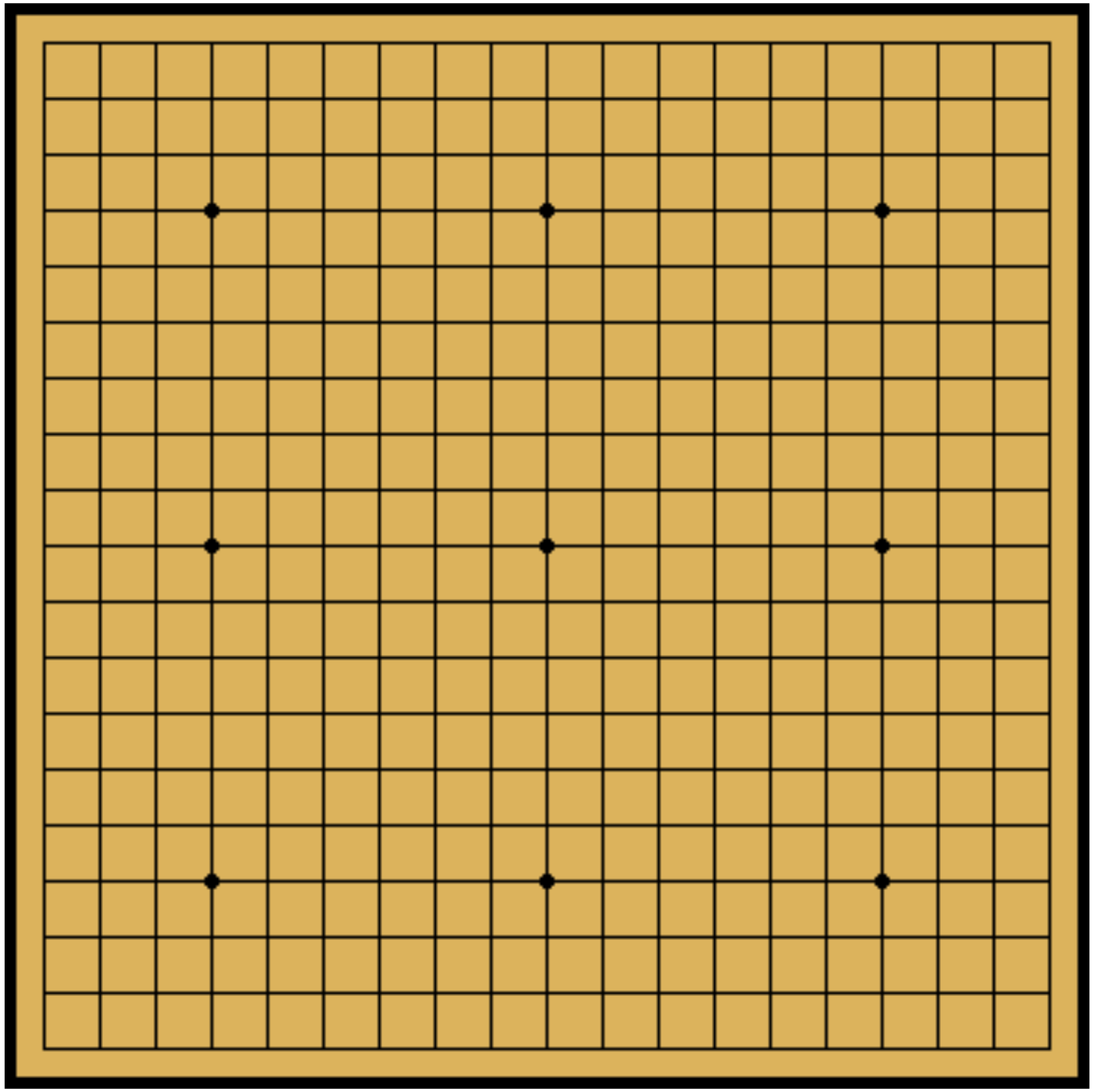
Dia. 19



Dia. 20

A player's score is determined by the number of stones that player has on the board plus the empty area surrounded by that player's stones.






nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO
CONSERVATION
SONGBIRDS À LA CARTE
Illegal harvest of millions of Mediterranean birds
PAGE 452
RESEARCH ETHICS
SAFEGUARD TRANSPARENCY
Don't let openness backfire on individuals
PAGE 459
POPULAR SCIENCE
WHEN GENES GOT 'SELFISH'
Dawkins's calling card 40 years on
PAGE 462
NATUREASIA.COM
28 January 2016
Vol. 529, No. 7587

STEP 1

Supervised learning of policy networks

We trained a 13-layer policy network, which we call the SL policy network, from 30 million positions from the KGS Go Server. The net- work predicted expert moves on a held out test set with an accuracy of 57.0% using all input features, and 55.7% using only raw board position and move history as inputs,

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}$$

STEP 2

Reinforcement learning of policy networks

Play many games

Compute policy gradient, use REINFORCE

$$\Delta\rho \propto \frac{\partial \log p_\rho(a_t|s_t)}{\partial \rho} z_t$$

The outcome $z_t = \pm r(s_T)$ is the terminal reward at the end of the game from the perspective of the current player at time step t: +1 for winning and −1 for losing.

We evaluated the final RL algorithm by sampling each move $a_t \sim p_\varrho(\cdot|s_t)$ from its output probability distribution over actions. When played head-to-head, the RL policy network won more than 80% of games against the SL policy network. We also tested against the strongest open-source Go program, Pachi, a sophisticated Monte Carlo search program, ranked at 2 amateur *dan* on KGS, that executes 100,000 simulations per move. Using no search at all, the RL policy network won 85% of games against Pachi.

STEP 3

Reinforcement learning of value networks

$$v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t \ldots T} \sim p]$$

Complicated function, approximate by a neural net

Play many games with the policy RL, train the weights of the value network by regression on state-outcome pairs (s, z), using stochastic gradient descent to minimize the mean squared error (MSE) between the predicted value $v_\theta(s)$, and the corresponding outcome z

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

At this pont we have

Policy (supervised) $p_\sigma(a \mid s)$

Policy (RL) $p_\rho(a \mid s)$

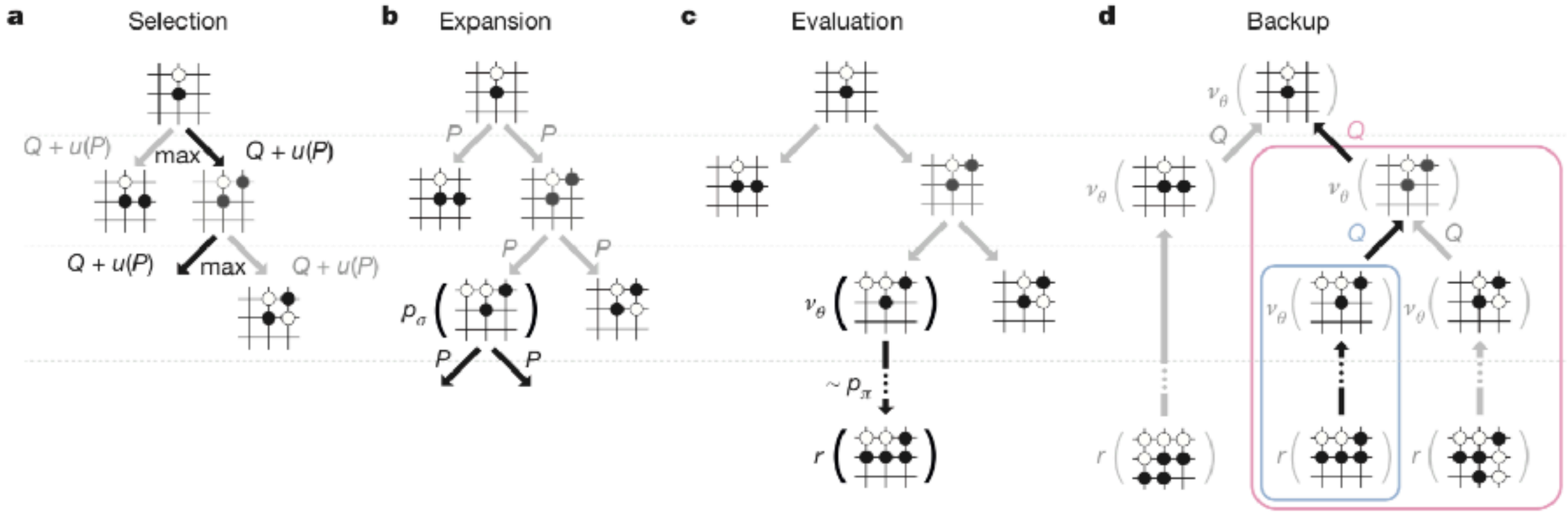
value networks $v^p(s) = \mathbb{E}[z_t | s_t = s,\ a_{t \ldots T} \sim p]$

How to use this? Monte-Carlo Tree-Search

Guiding the search in the tree of possibilities!

How to use this?

Monte-Carlo Tree-Search



At each time step t of each simulation, an action a_t is selected from state s_t :

$$a_t = \text{argmax}_a \left[Q(s_t, a) + \frac{p_\sigma(s, a)}{1 + N(s, a)} \right]$$

$$N(s, a) = \sum_{i=1}^{n} 1(s, a, i) \qquad Q(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{n} 1(s, a, i) V(s)$$

$$V(s) = (1 - \lambda)\nu_\theta(s) + \lambda z_{\text{fast}}$$

Fan Hui

October 2015

Lee Sedol

March 2016

Move 37

See the exact moment the world champion of Go realises DeepMind is vastly superior [GIF]

Jim Edwards
Mar. 12, 2016, 2:30 PM 9,626

FACEBOOK LINKEDIN TWITTER

Go fans have noticed a couple of key moments in the match between world champion Lee Sedol and Google's DeepMind AlphaGo

HOW GOOGLE'S AI VIEWED THE MOVE NO HUMAN COULD UNDERSTAND

Game 2 [edit]

AlphaGo (black) won the second game. Lee stated afterwards that "AlphaGo played a nearly perfect game",[49] "from very beginning of the game I did not feel like there was a point that I was leading".[50] One of the creators of AlphaGo, Demis Hassabis, said that the system was confident of victory from the midway point of the game, even though the professional commentators could not tell which player was ahead.[50]

Michael Redmond (9p) noted that AlphaGo's 19th stone (move 37) was "creative" and "unique".[30] Lee took an unusually long time to respond to the move.[30] An Younggil (8p) called AlphaGo's move 37 "a rare and intriguing shoulder hit" but said Lee's counter was "exquisite". He stated that control passed between the players several times before the endgame, and especially praised AlphaGo's moves 151, 157, and 159, calling them "brilliant".[51]

Alphago Zero (2017)

AlphaGo Zero: Learning from scratch

Artificial intelligence research has made rapid progress in a wide variety of domains from speech recognition and image classification to genomics and drug discovery. In many cases, these are specialist systems that leverage enormous amounts of human expertise and data.

However, for some problems this human knowledge may be too expensive, too unreliable or simply unavailable. As a result, a long-standing ambition of AI research is to bypass this step, creating algorithms that achieve superhuman performance in the most challenging domains with no human input. In our most recent paper, published in the journal Nature, we demonstrate a significant step towards this goal.

nature
International journal of science

Article | Published: 18 October 2017

Mastering the game of Go without human knowledge

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel & Demis Hassabis

Nature **550**, 354–359 (19 October 2017) | Download Citation

Abstract

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by

A single network, Learning with RL

1 Reinforcement Learning in *AlphaGo Zero*

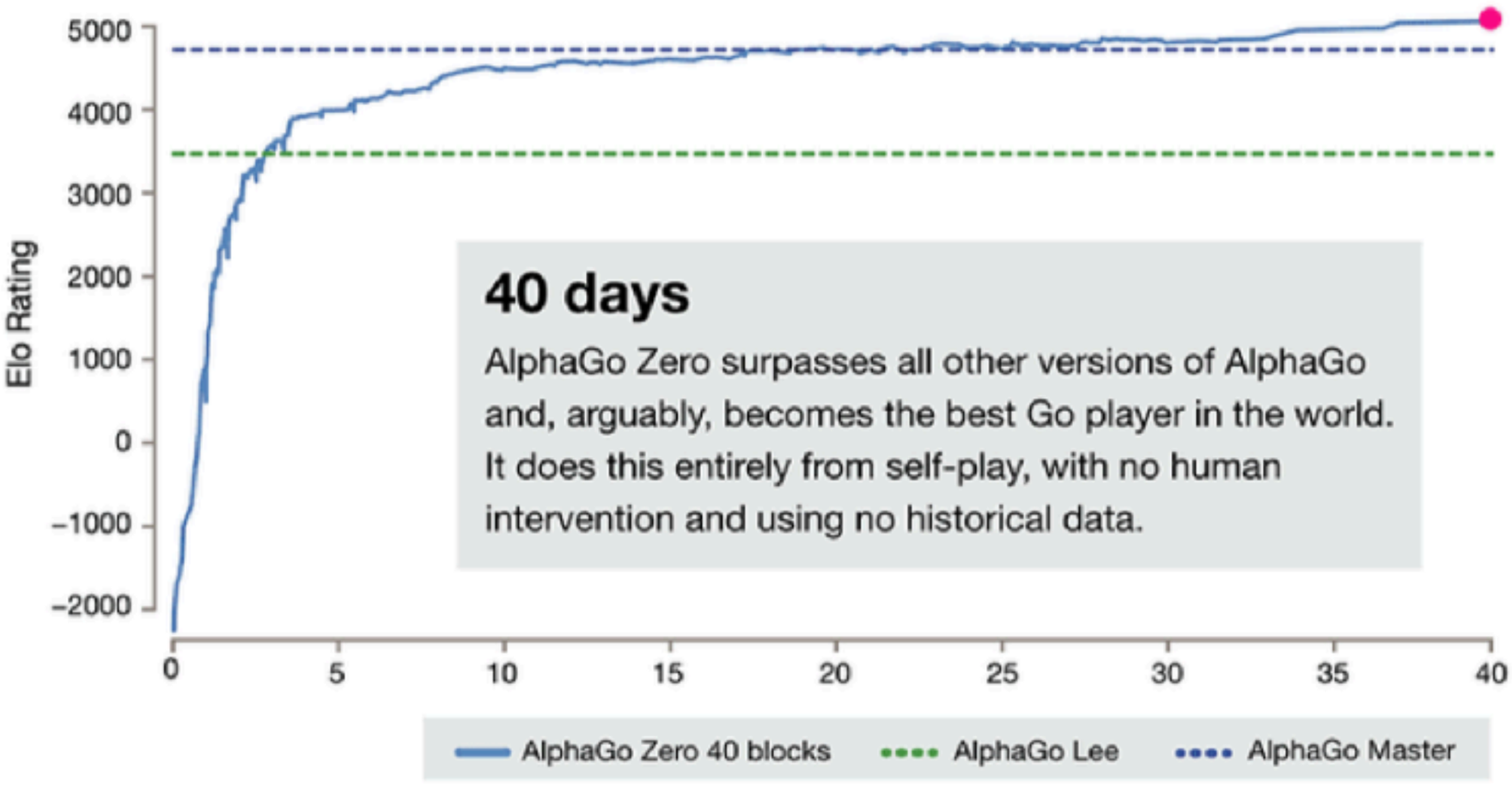
Our new method uses a deep neural network f_θ with parameters θ. This neural network takes as an input the raw board representation s of the position and its history, and outputs both move probabilities and a value, $(\mathbf{p}, v) = f_\theta(s)$. The vector of move probabilities $\mathbf{p}$ represents the probability of selecting each move (including pass), $p_a = Pr(a|s)$. The value v is a scalar evaluation, estimating the probability of the current player winning from position s. This neural network combines the roles of both policy network and value network [12] into a single architecture. The neural network consists of many residual blocks [4] of convolutional layers [16,17] with batch normalisation [18] and rectifier non-linearities [19] (see Methods).

AlphaGo Zero

- Beats AlphaGo by 100:0

AlphaGo Zero

- Beats AlphaGo by 100:0



Computer Science > Artificial Intelligence

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis

(Submitted on 5 Dec 2017)

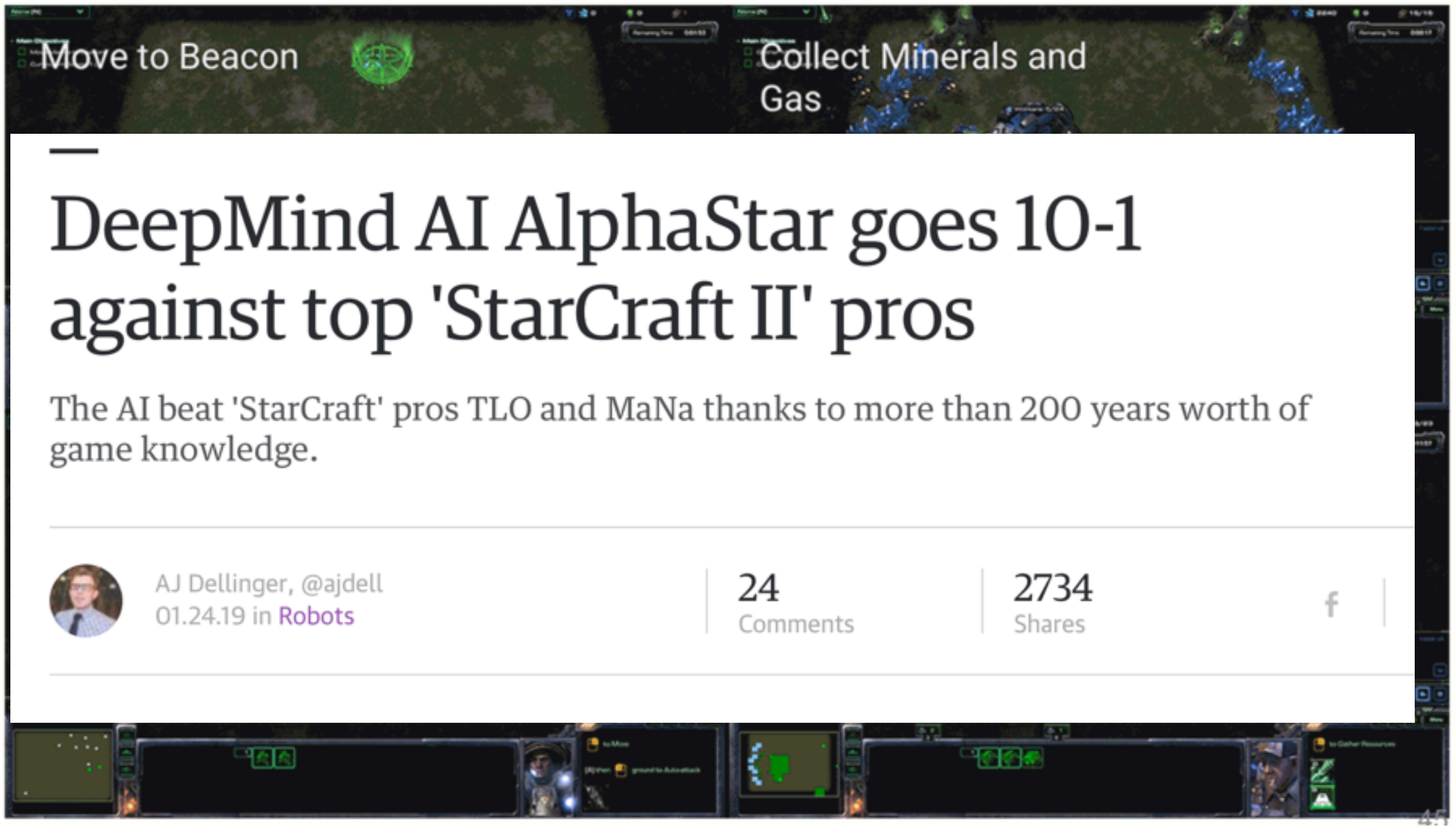
The game of chess is the most widely-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. In contrast, the AlphaGo Zero program recently achieved superhuman performance in the game of Go, by tabula rasa reinforcement learning from games of self-play. In this paper, we generalise this approach into a single AlphaZero algorithm that can achieve, tabula rasa, superhuman performance in many challenging domains. Starting from random play, and given no domain knowledge except the game rules, AlphaZero achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case.

What's next for AI?

DeepMind's AI is Struggling to Beat Starcraft II - Bloomberg
https://www.bloomberg.com/.../deepmind-master-of-go-struggles-to-crack-its-next-mi...

What's next for AI?

DeepMind's AI is Struggling to Beat Starcraft II - Bloomberg
https://www.bloomberg.com/.../deepmind-master-of-go-struggles-to-crack-its-next-mi...



—

DeepMind AI AlphaStar goes 10-1 against top 'StarCraft II' pros

The AI beat 'StarCraft' pros TLO and MaNa thanks to more than 200 years worth of game knowledge.

AJ Dellinger, @ajdell
01.24.19 in Robots

24
Comments

2734
Shares