A crash course in machine learning

Florent Krzakala & Antoine Baker

https://sphinxteam.github.io/mlcourse_2019/

florent.krzakala@gmail.com **antoine.baker@gmail.com**

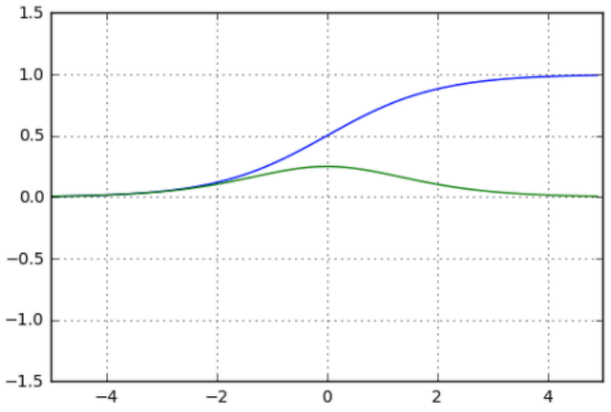
1: Tricks of the trade

1

Initialization of the weight

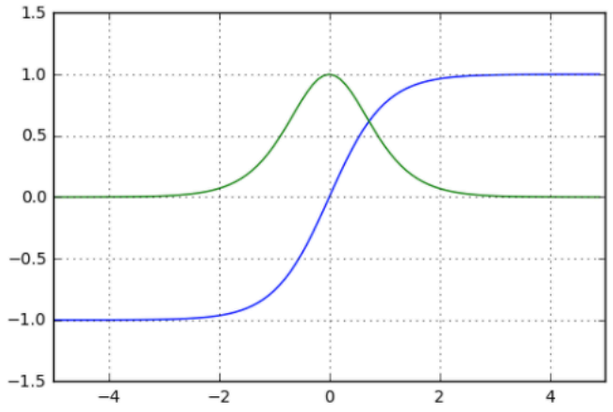
- Weights need to be small enough
  - around origin for symmetric activation functions (tanh, sigmoid)
    → stimulate activation functions near their linear regime
  - larger gradients → faster training
- Weights need to be large enough
  - otherwise signal is too weak for any serious learning

RELU prevent vanishing gradients
(but dead relus can exist! -> Leaky relu!)



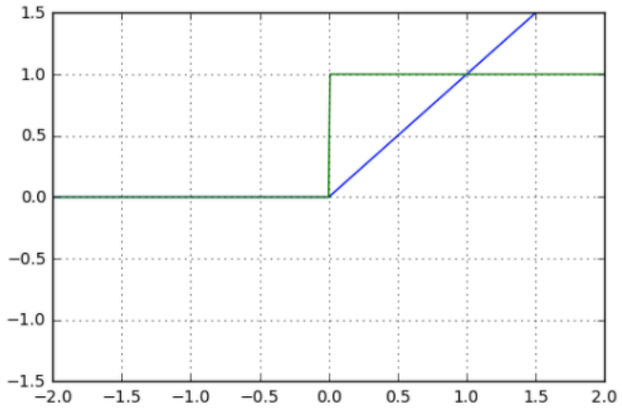
$$\text{sigm}(x) = \frac{1}{1+e^{-x}}$$

$$\text{sigm}'(x) = \text{sigm}(x)(1-\text{sigm}(x))$$



$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

$$\tanh'(x) = 1-\tanh(x)^2$$



$$\text{relu}(x) = \max(0,x)$$

$$\text{relu}'(x) = 1_{x>0}$$

1

Initialization of the weight

Xavier Initialization

$$\mathcal{N}(0, \frac{2}{N_{in} + N_{out}})$$

glorot_uniform

```
glorot_uniform(seed=None)
```

Glorot uniform initializer, also called Xavier uniform initializer.

It draws samples from a uniform distribution within [-limit, limit] where `limit` is `sqrt(6 / (fan_in + fan_out))` where `fan_in` is the number of input units in the weight tensor and `fan_out` is the number of output units in the weight tensor.

Arguments

- **seed**: A Python integer. Used to seed the random generator.

Returns

1

Initialization of the weight

Kaiming-He initialization $\mathcal{N}(0, \frac{2}{N_{in}})$

* Scale the incoming weight to have a O(1) variable
* The factor 2 depends on activation: ReLUs ground to 0 the linear activation about half the Time -> Double weight variance for Relu to adapt

he_normal

```
he_normal(seed=None)
```

He normal initializer.

It draws samples from a truncated normal distribution centered on 0 with `stddev = sqrt(2 / fan_in)` where `fan_in` is the number of input units in the weight tensor.

Arguments

- **seed**: A Python integer. Used to seed the random generator.

Returns

1

Initialization of the weight

Kaiming-He initialization $\mathcal{N}(0, \frac{2}{N_{in}})$

The same type of reasoning can be applied to other activation functions

From `torch/nn/init.py`:

```python
def calculate_gain(nonlinearity, param=None):
    linear_fns = ['linear', 'conv1d', 'conv2d', 'conv3d', 'conv_transpose1d', 'conv_transpose2d', 'conv_transpose3d']
    if nonlinearity in linear_fns or nonlinearity == 'sigmoid':
        return 1
    elif nonlinearity == 'tanh':
        return 5.0 / 3
    elif nonlinearity == 'relu':
        return math.sqrt(2.0)
    elif nonlinearity == 'leaky_relu':
        if param is None:
            negative_slope = 0.01
        elif not isinstance(param, bool) and isinstance(param, int) or isinstance(param, float):
            # True/False are instances of int, hence check above
            negative_slope = param
        else:
            raise ValueError("negative_slope {} not a valid number".format(param))
        return math.sqrt(2.0 / (1 + negative_slope ** 2))
    else:
        raise ValueError("Unsupported nonlinearity {}".format(nonlinearity))
```

Weight initialization

Does it actually matter that much?

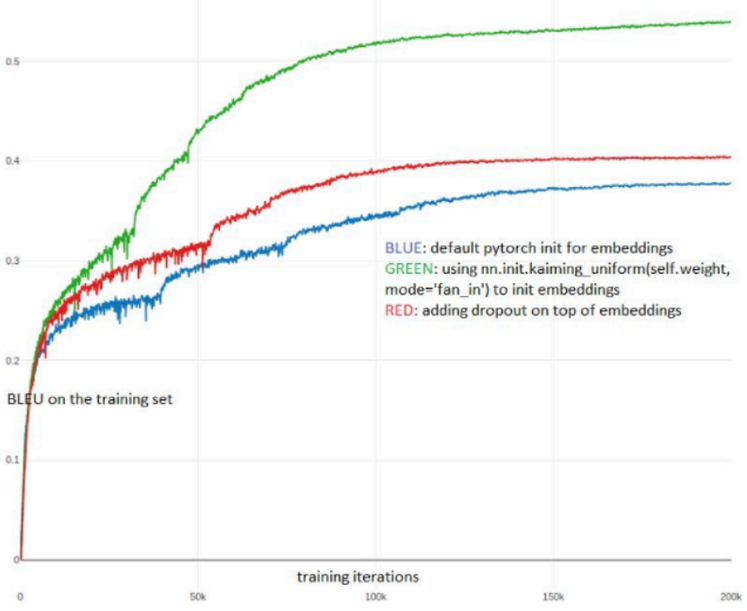
Weight initialization

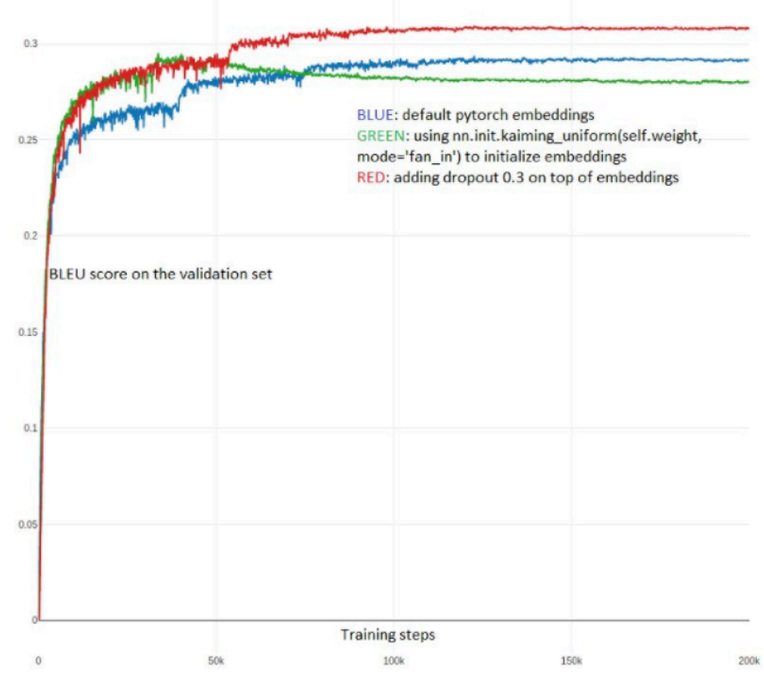
Does it actually matter that much?

Anton Osokin
@aosokin_ml

Following

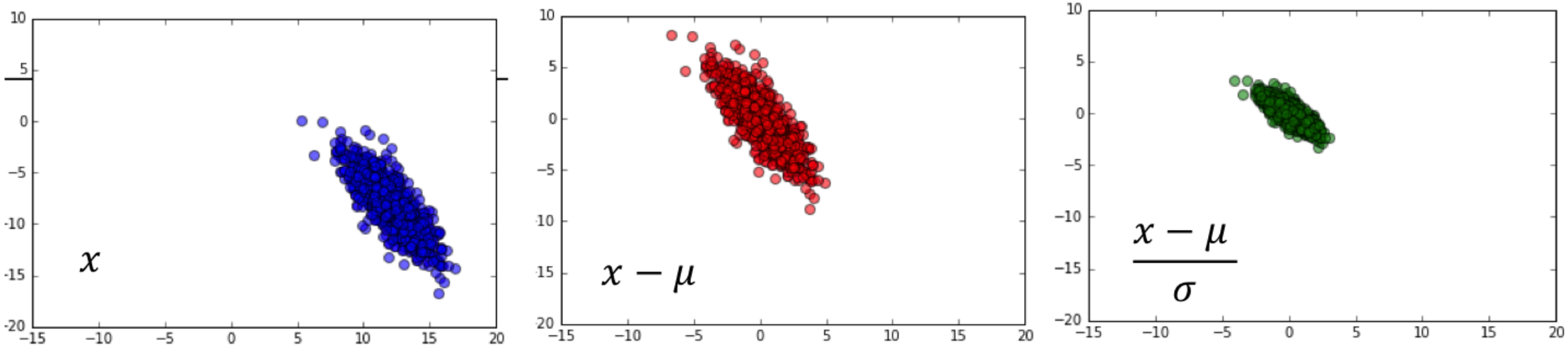
Initialization in deep learning matters a lot! In a simple @PyTorch code for seq2seq NMT, changing the init of embeddings from default to kaiming (Gaussian vs uniform is not important, but rescaling is!) and regularizing more boosts results by 2 BLEU. How to tune these things?





Data pre-processing

- Input variables should be as decorrelated as possible
  - Input variables are "more independent"
  - Network is forced to find non-trivial correlations between inputs
  - Decorrelated inputs → better optimization
- Input variables follow a more of less Gaussian distribution
- In practice:
  - compute mean and standard deviation
    - per pixel: (μ, σ^2)
    - per color channel:



2

Batch Normalization

```
from keras.layers.normalization import BatchNormalization
model = Sequential()
# think of this as the input layer
model.add(Dense(64, input_dim=16, init='uniform'))
model.add(BatchNormalization())
model.add(Activation('tanh'))
model.add(Dropout(0.5))
# think of this as the hidden layer
model.add(Dense(64, init='uniform'))
model.add(BatchNormalization())
model.add(Activation('tanh'))
model.add(Dropout(0.5))
# think of this as the output layer
model.add(Dense(2, init='uniform'))
model.add(BatchNormalization())
model.add(Activation('softmax'))
# optimiser and loss function
model.compile(loss='binary_crossentropy', optimizer=sgd)
```

During training, we normalise the activations of the previous layer for each batch:

We normalise in order to maintains the mean activation close to 0 and the activation standard deviation close to 1 befo

the activation function

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

3

What do you do when do not have enough data?

You create more!

Data augmentation

- Changing the pixels without changing the label
- Train on transformed data
- Widely used

Original

Flip

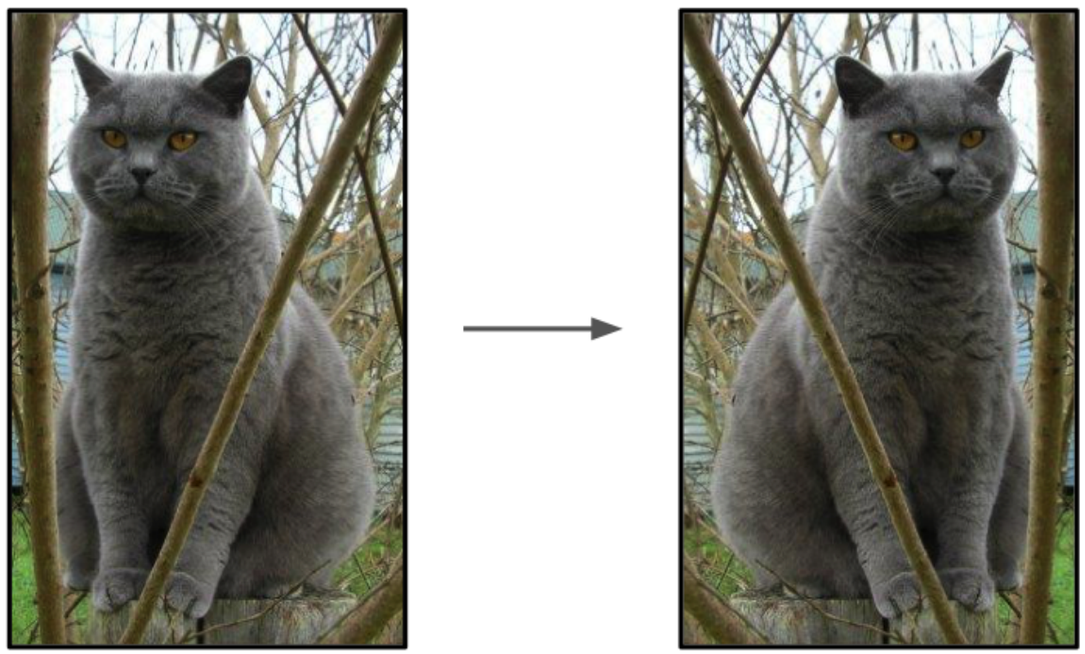
Random crop

Contrast

Tint

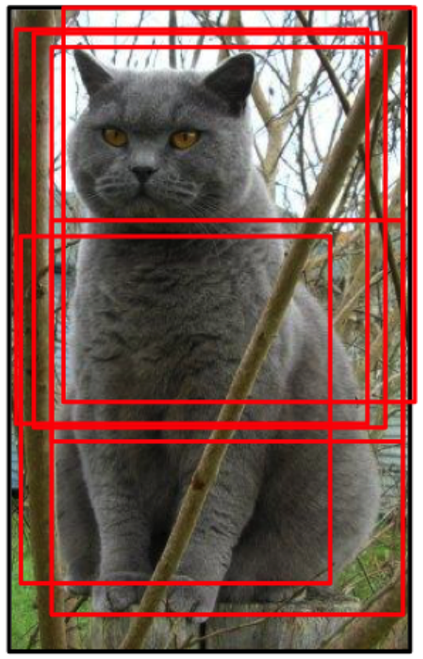
Data augmentation

Horizontal flips



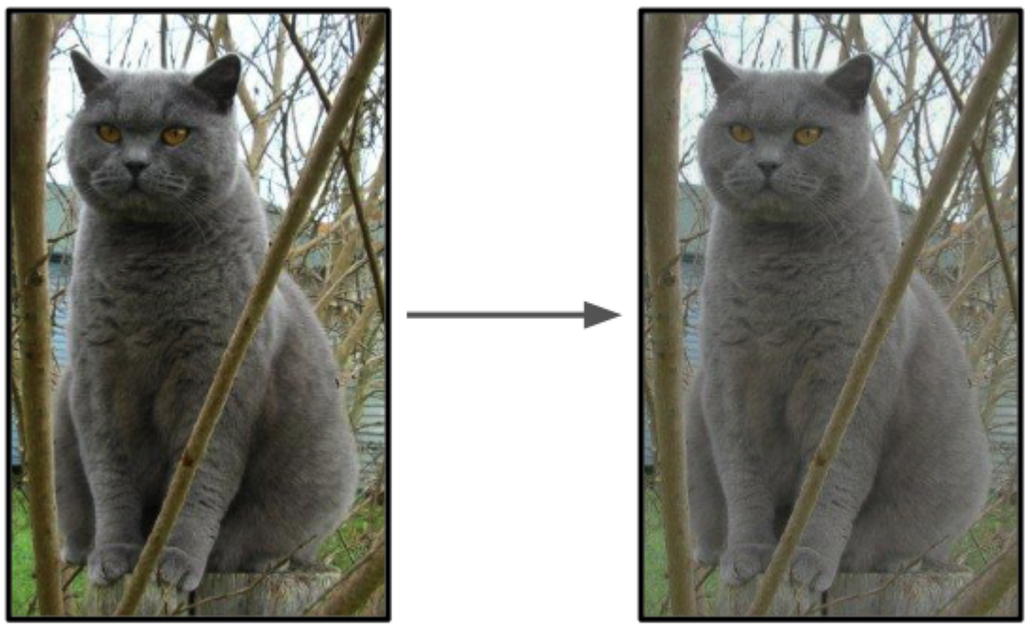
Data augmentation

Random crops/scales



Data augmentation

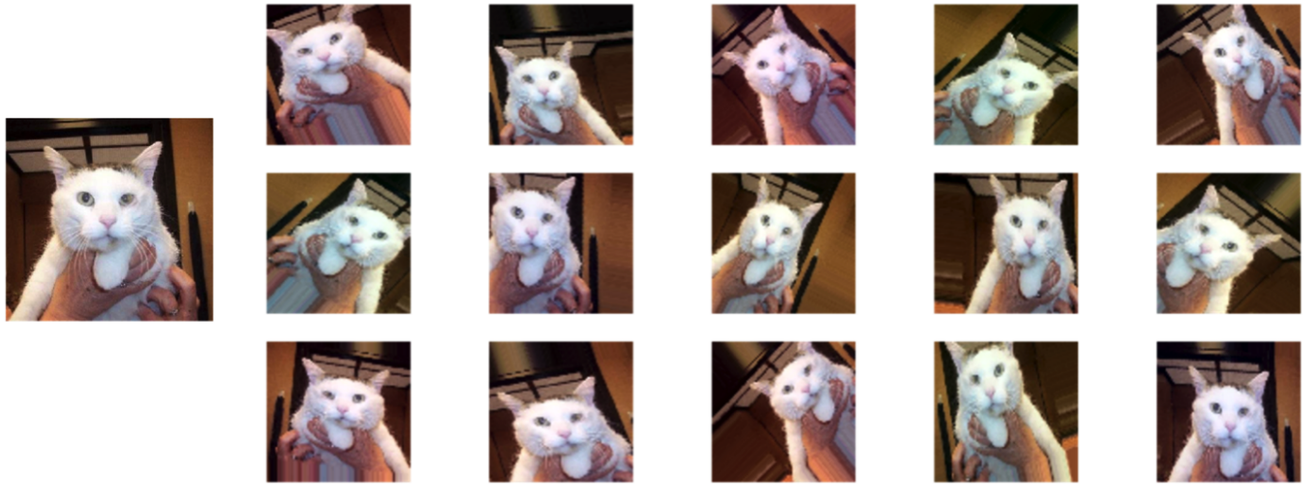
Color jitter



- randomly jitter color, brightness, contrast, etc.

Data augmentation

- Various techniques can be mixed
- Domain knowledge helps in finding new data augmentation techniques
- Very useful for small datasets



3

Data augmentation

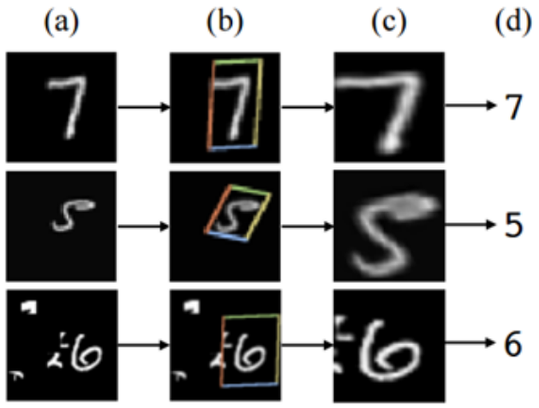
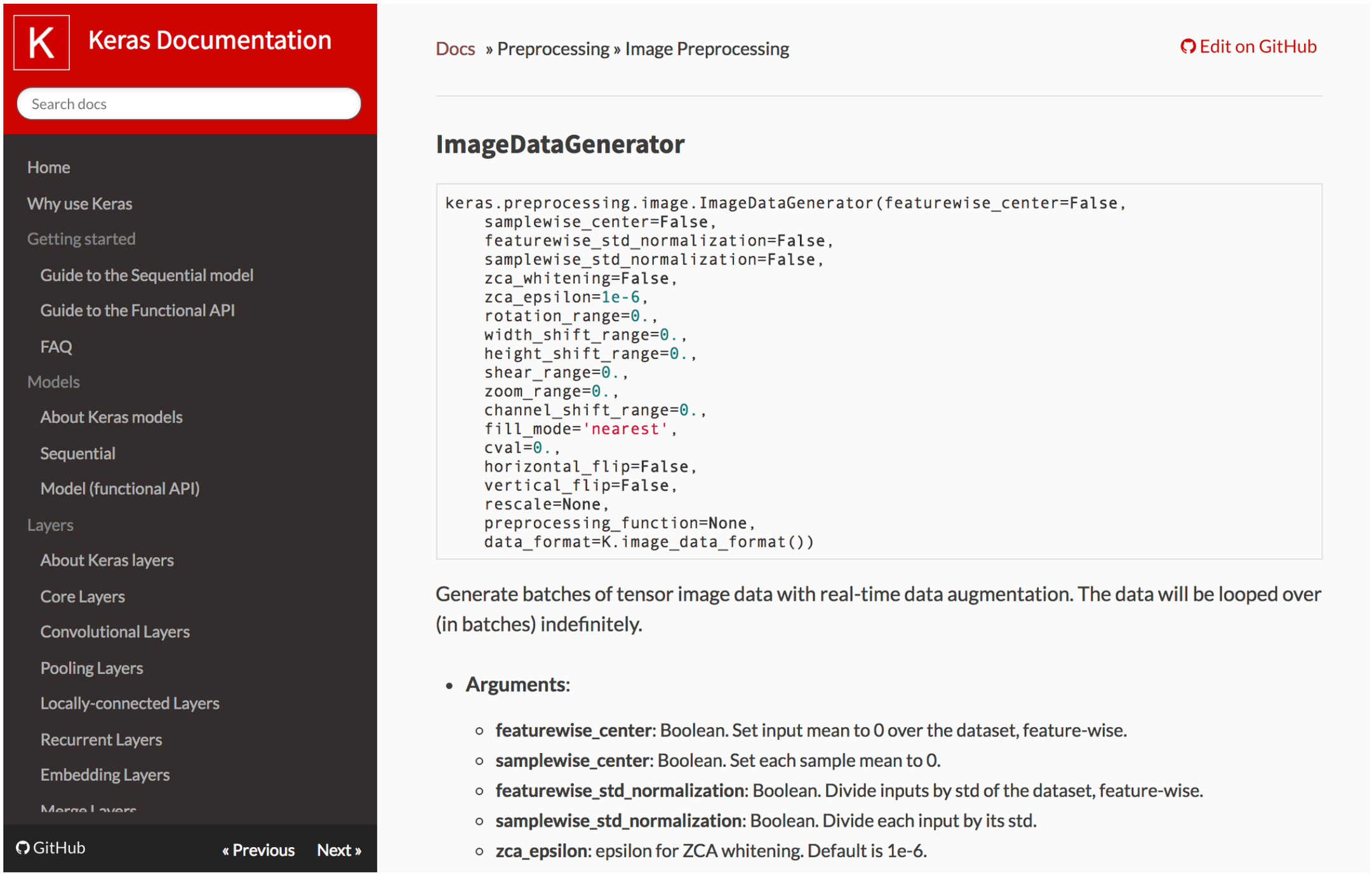


Figure 1: The result of using a spatial transformer as the first layer of a fully-connected network trained for distorted MNIST digit classification. (a) The input to the spatial transformer network is an image of an MNIST digit that is distorted with random translation, scale, rotation, and clutter. (b) The localisation network of the spatial transformer predicts a transformation to apply to the input image. (c) The output of the spatial transformer, after applying the transformation. (d) The classification prediction produced by the subsequent fully-connected network on the output of the spatial transformer. The spatial transformer network (a CNN including a spatial transformer module) is trained end-to-end with only class labels – no knowledge of the groundtruth transformations is given to the system.

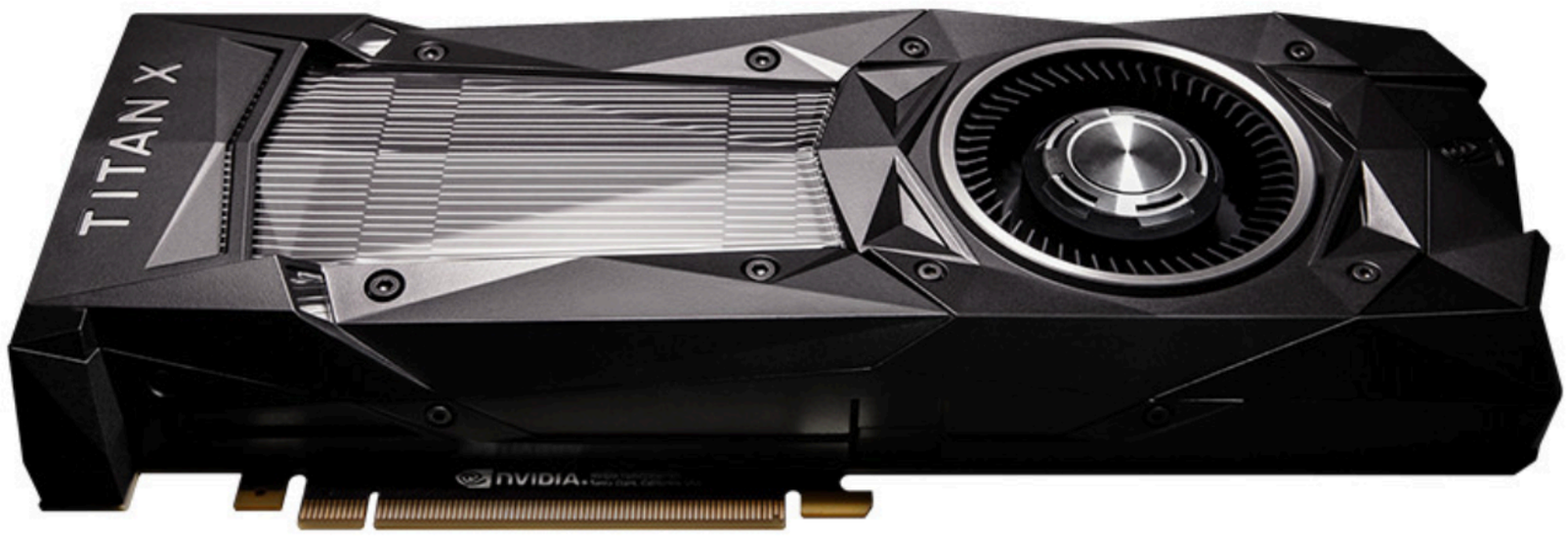
Great overview of the function of a Spatial Tranfomer module

3

Data augmentation



GPUs



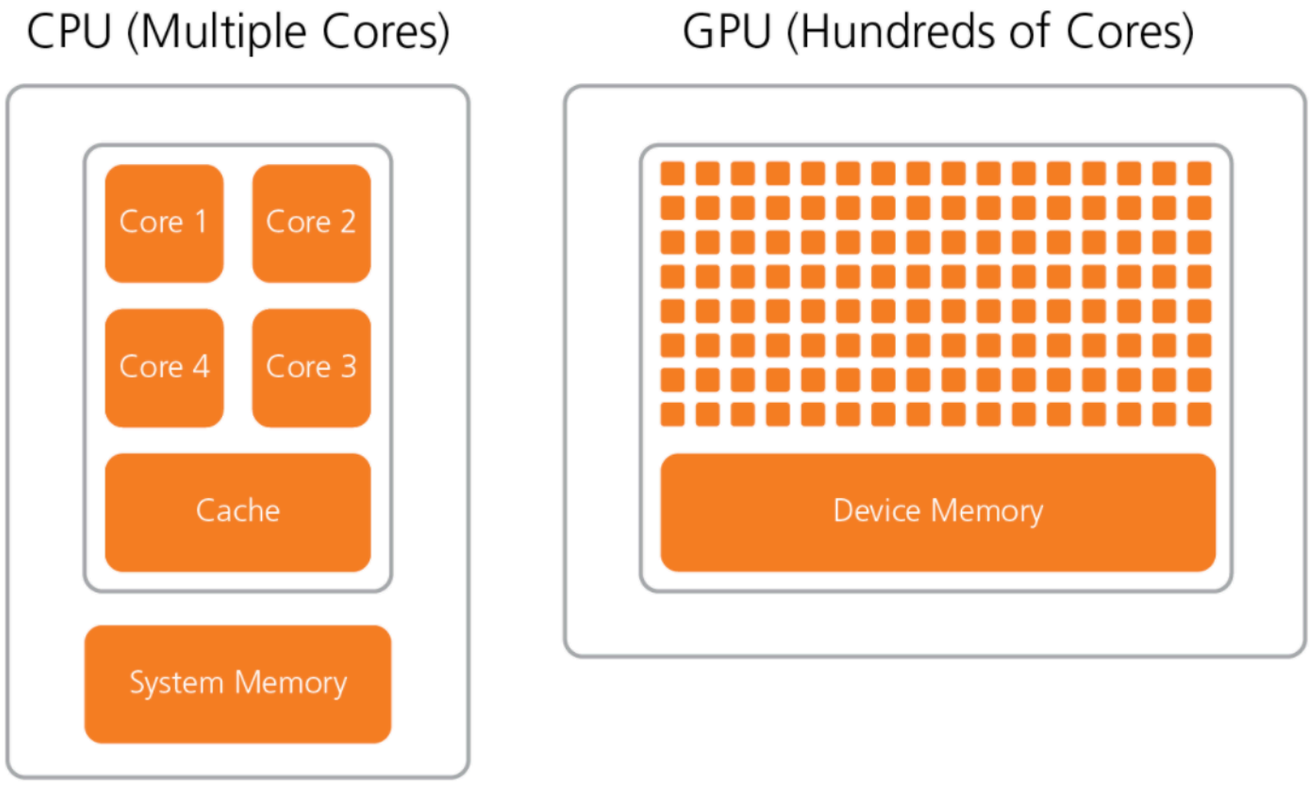
CPU vs GPU

CPU

GPU

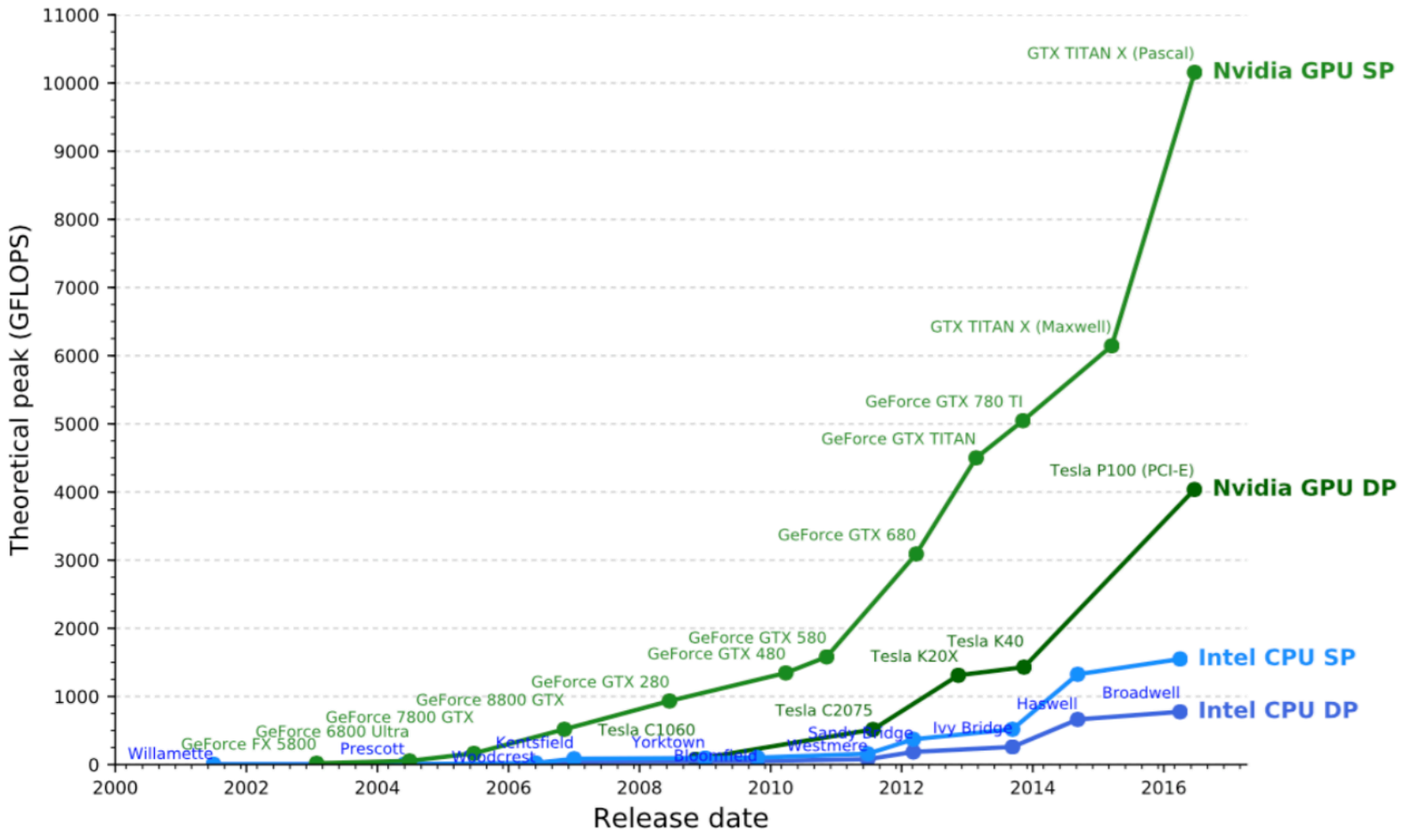
CPU vs GPU

- CPU:
  - fewer cores; each core is faster and more powerful
  - useful for sequential tasks
- GPU:
  - more cores; each core is slower and weaker
  - great for parallel tasks



CPU vs GPU

- SP = single precision, 32 bits / 4 bytes
- DP = double precision, 64 bits / 8 bytes



4

NVIDIA Corporation

NASDAQ: NVDA - Dec 19, 5:25 AM EST

197.90 USD ↑6.34 (3.31%)

After-hours: 198.95 ↑0.53%

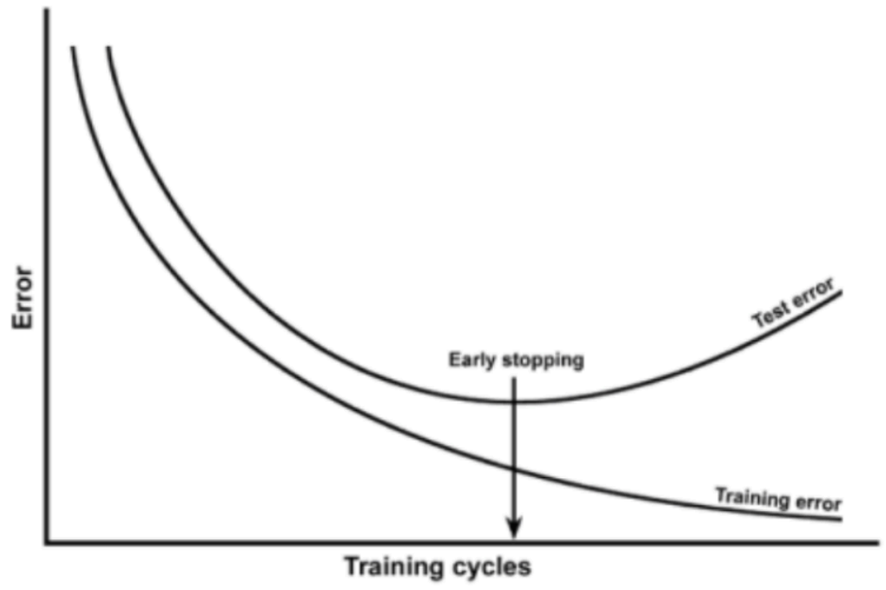
1 day | 5 day | 1 month | 3 months | 1 year | 5 years | max

Open	193.20	Mkt cap	119.93B
High	198.02	P/E ratio	49.51
Low	192.00	Div yield	0.30%

2: Regularization

Early stopping

- To avoid overfitting another popular technique is early stopping
- Monitor performance on validation set
- Training the network will decrease training error, as well validation error (although with a slower rate usually)
- Stop when validation error starts increasing
  - most likely the network starts to overfit
  - use a *patience* term to let it degrade for a while and then stop



Remember this?

The linear model revisited: regularisation

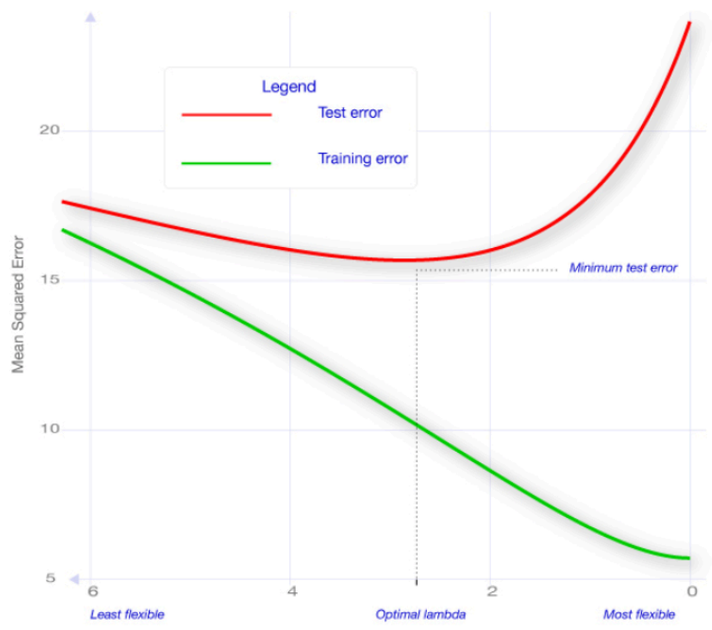
Replace $\hat{\theta} = \text{argmin}(||\mathbf{Y} - \mathrm{A}\theta||_2^2)$

By $\hat{\theta} = \text{argmin}(||\mathbf{Y} - \mathrm{A}\theta||_2^2) + \mathrm{g}(\theta)$

L2-regularization aka Tikhonov regularization
aka Ridge regression aka Weight decay

$$\hat{\theta} = \text{argmin}(||\mathbf{Y} - \mathrm{A}\theta||_2^2) + \Gamma||\theta||_2^2$$

Find the best Γ
using cross-validation



Weight Decay

Regularization

L2 regularization:

$$\|\theta\|_2 = (w_1)^2 + (w_2)^2 + \dots$$

- New loss function to be minimized

$$L'(\theta) = L(\theta) + \lambda \frac{1}{2}\|\theta\|_2$$

Gradient: $\frac{\partial L'}{\partial w} = \frac{\partial L}{\partial w} + \lambda w$

Update:

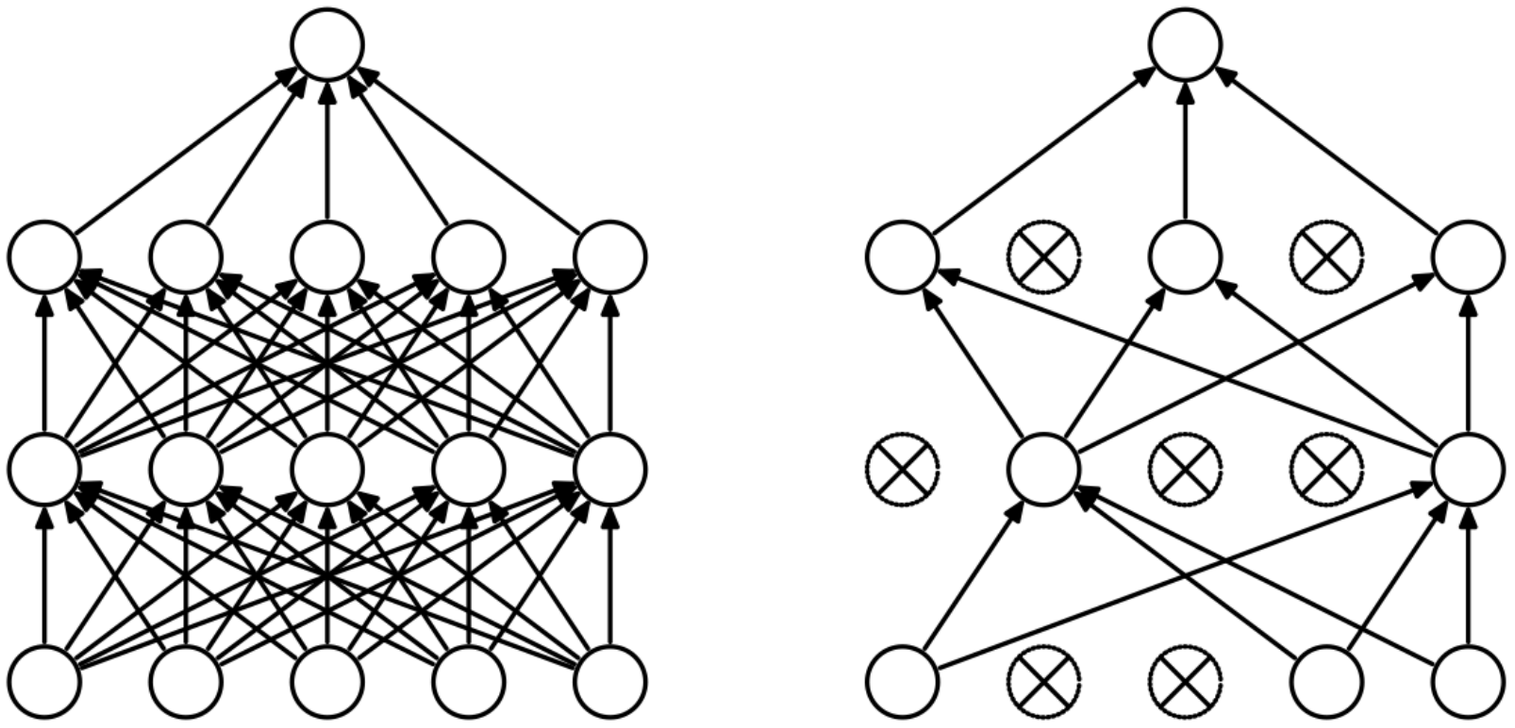
$$w^{t+1} \rightarrow w^t - \eta \frac{\partial L'}{\partial w} = w^t - \eta\left(\frac{\partial L}{\partial w} + \lambda w^t\right)$$

$$= (1 - \eta\lambda) w^t - \eta \frac{\partial L}{\partial w}$$

Weight Decay

$(1 - \eta\lambda) w^t$: Closer to zero

Dropout



(a) Standard Neural Net

(b) After applying dropout.

Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network. The mask for each unit is sampled independently from all of the others

Dropout

Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.

Editor: Yoshua Bengio

Abstract

Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different "thinned" networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning

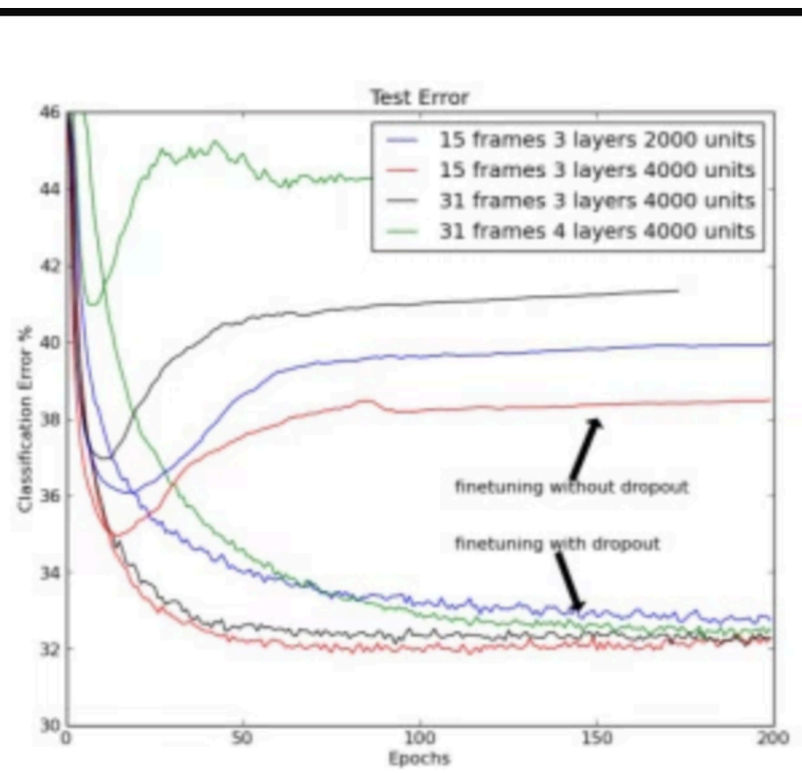
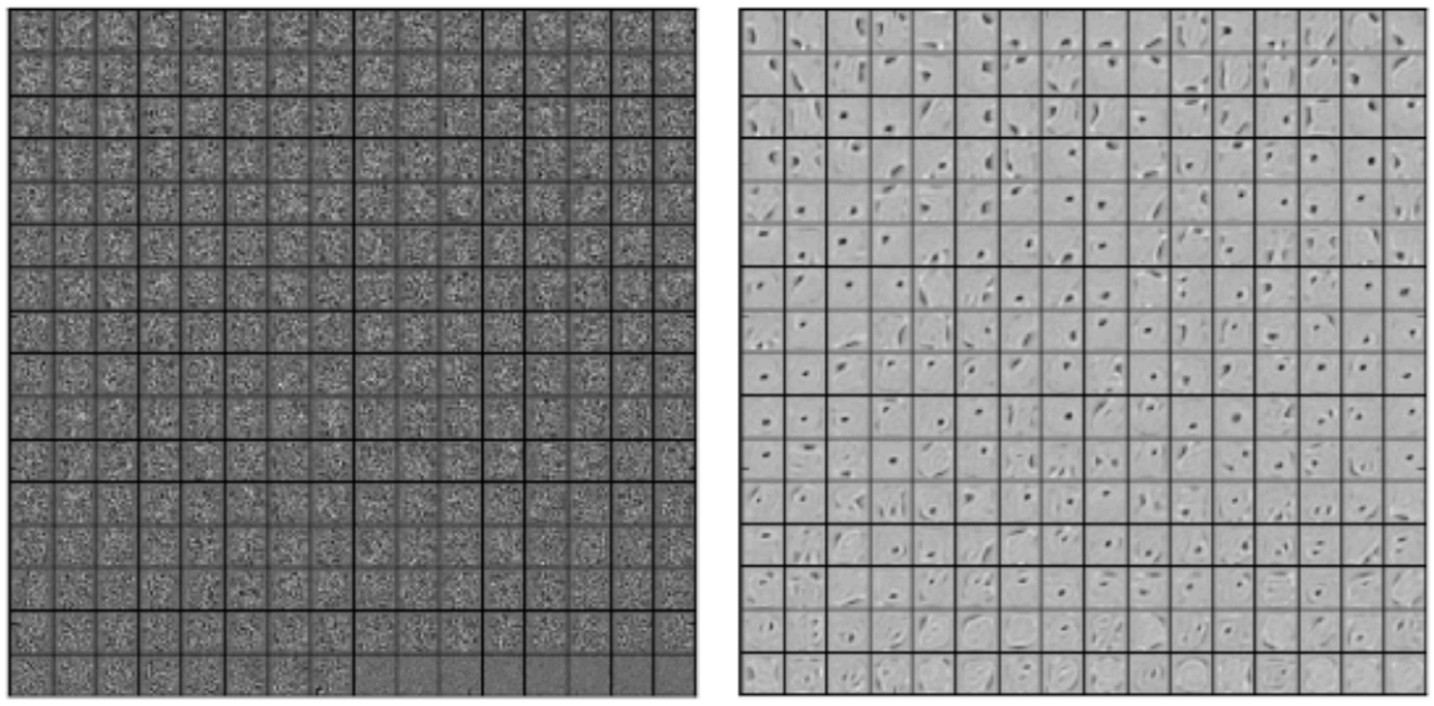


Fig. 2: The frame *classification* error rate on the core test set of the TIMIT benchmark. Comparison of standard and dropout finetuning for different network architectures. Dropout of 50% of the hidden units and 20% of the input units improves classification.

Dropout

Features learned on MNIST with one hidded layer autoencoders having 256 rectified linear units

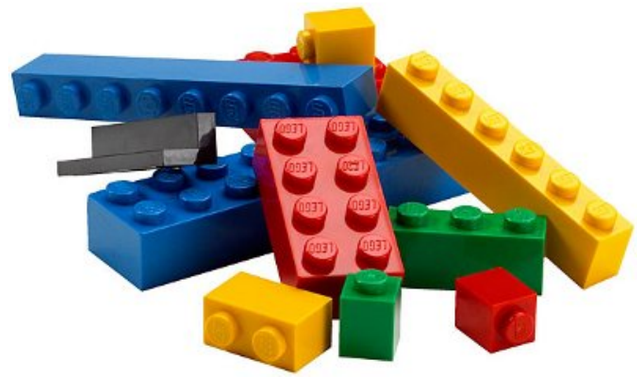


(a) Without dropout

(b) Dropout with $p = 0.5$.

3: Special Layers

Playing Lego



Example

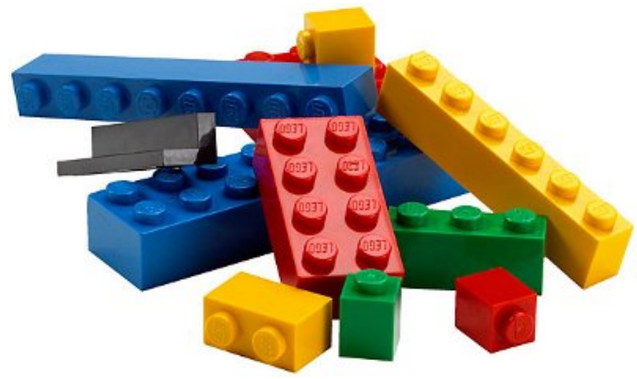
```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
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]]

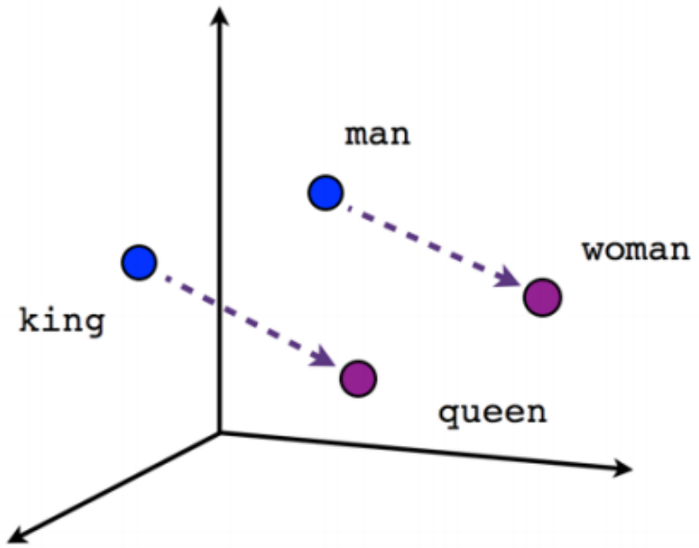
This layer can only be used as the first layer in a model.

Example

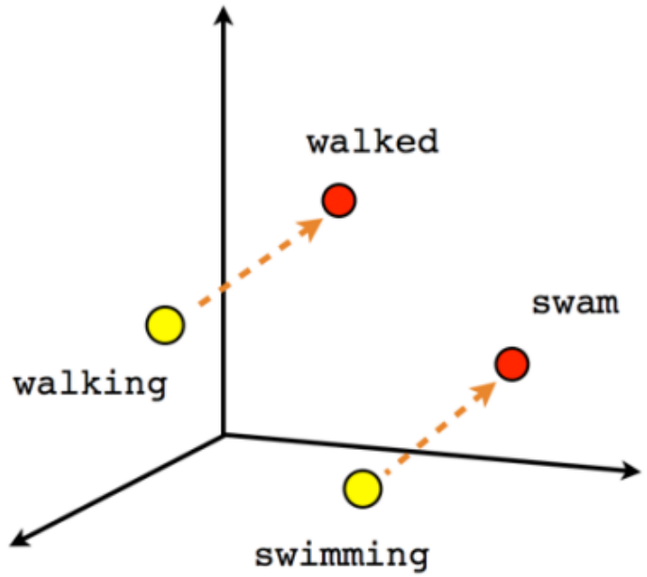
```
model = Sequential()
model.add(Embedding(1000, 64, input_length=10))
# the model will take as input an integer matrix of size (batch, input_length).
# the largest integer (i.e. word index) in the input should be no larger than 999 (vocabulary
# now model.output_shape == (None, 10, 64), where None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

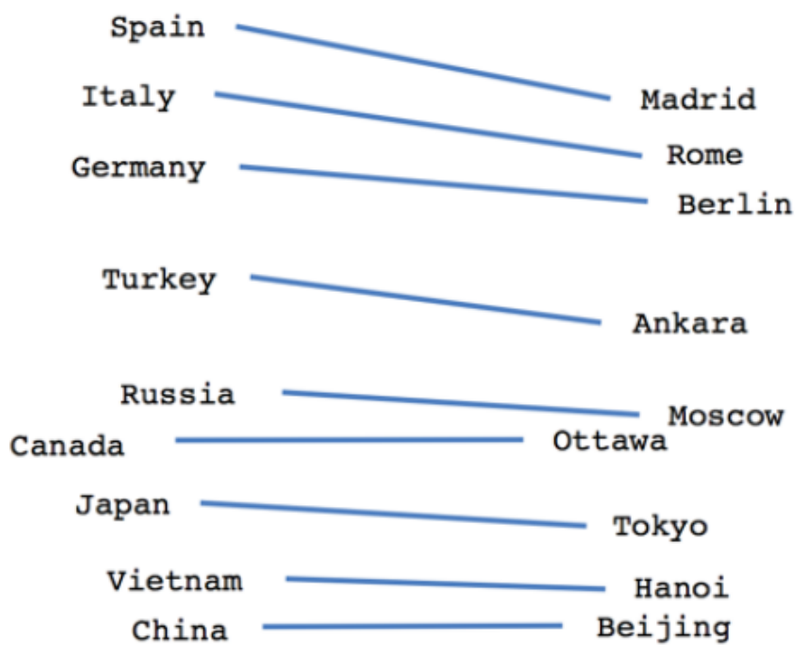
Example: word2vec



Male-Female



Verb tense



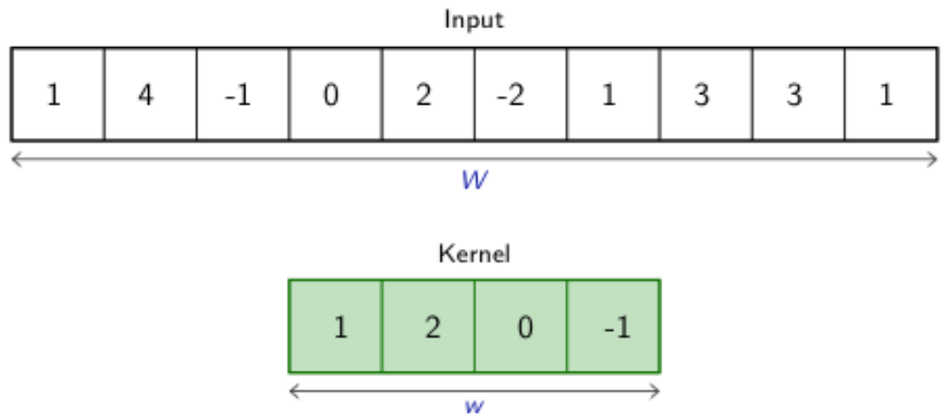
Country-Capital

Preservation of semantic and syntactic relationships

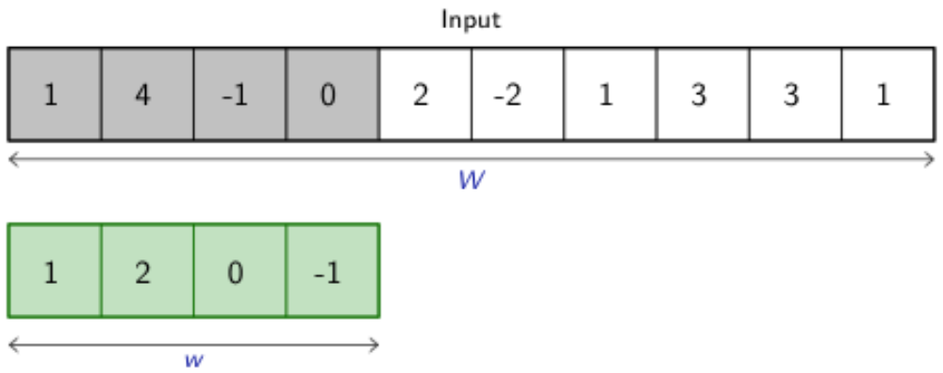
Convolutional and pooling layers

Fondamental for images and sounds!

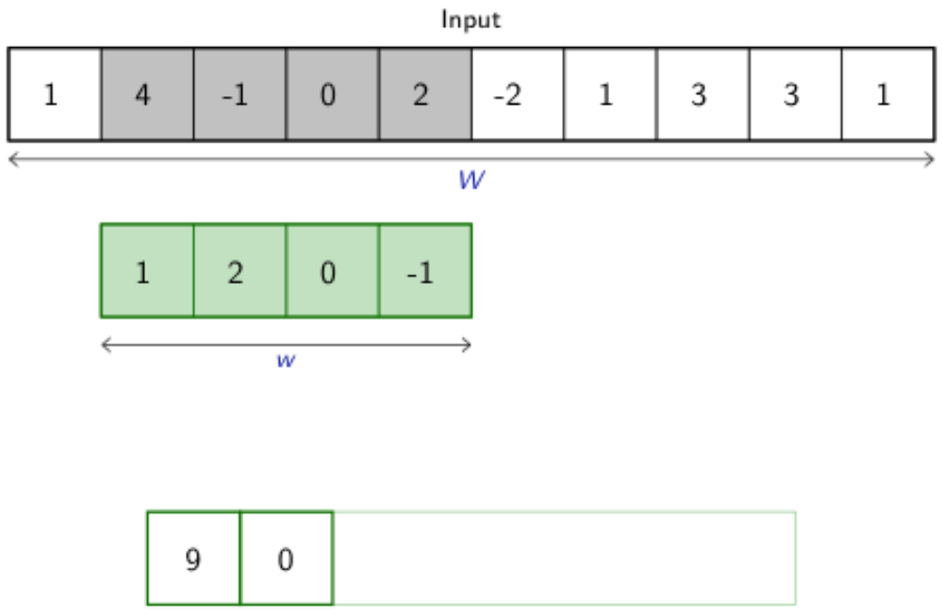
Convolution 1d



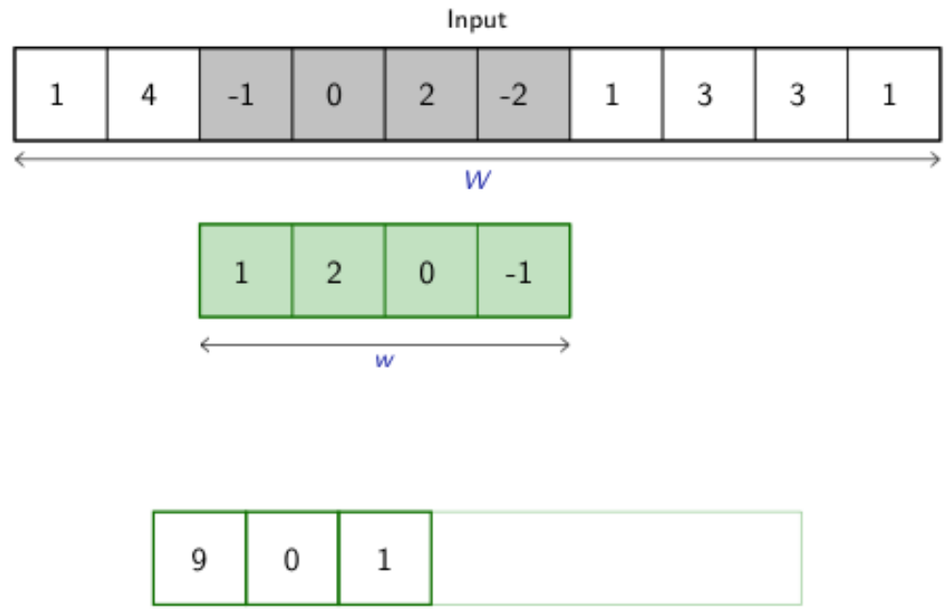
Convolution 1d



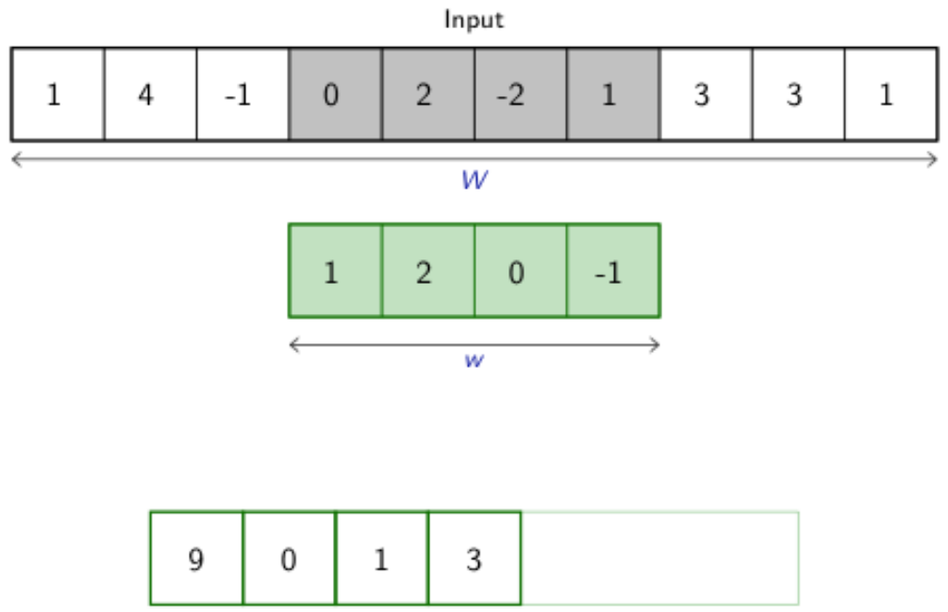
Convolution 1d



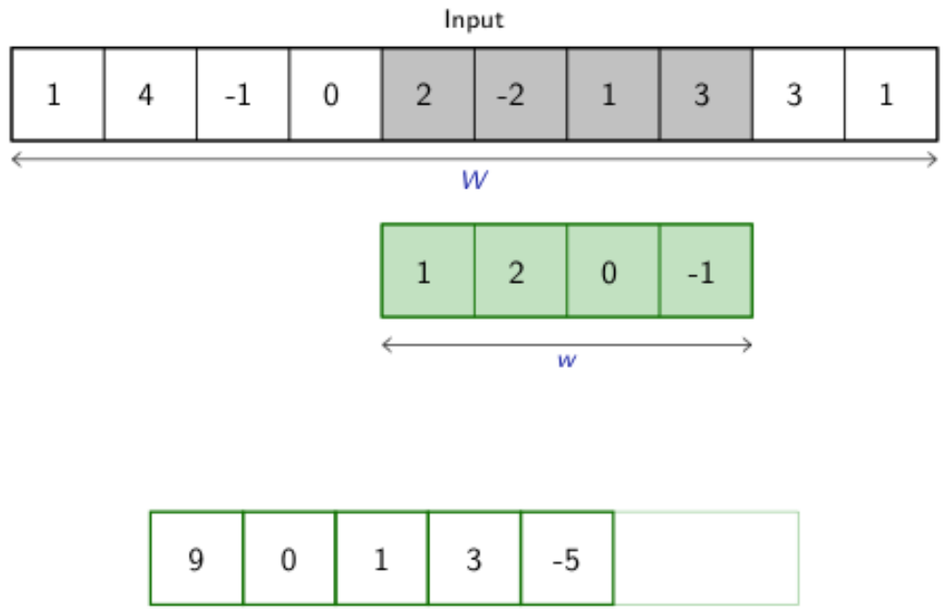
Convolution 1d



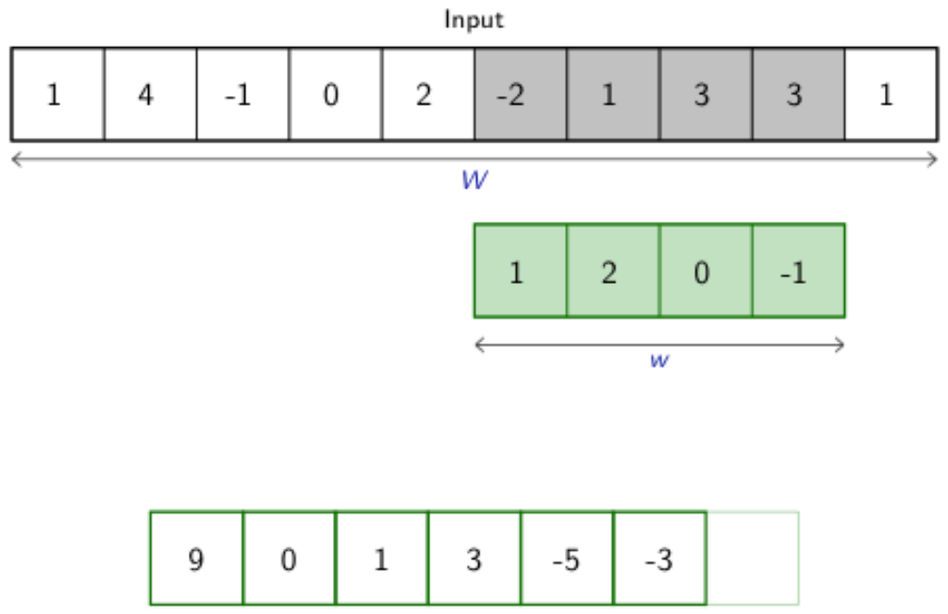
Convolution 1d



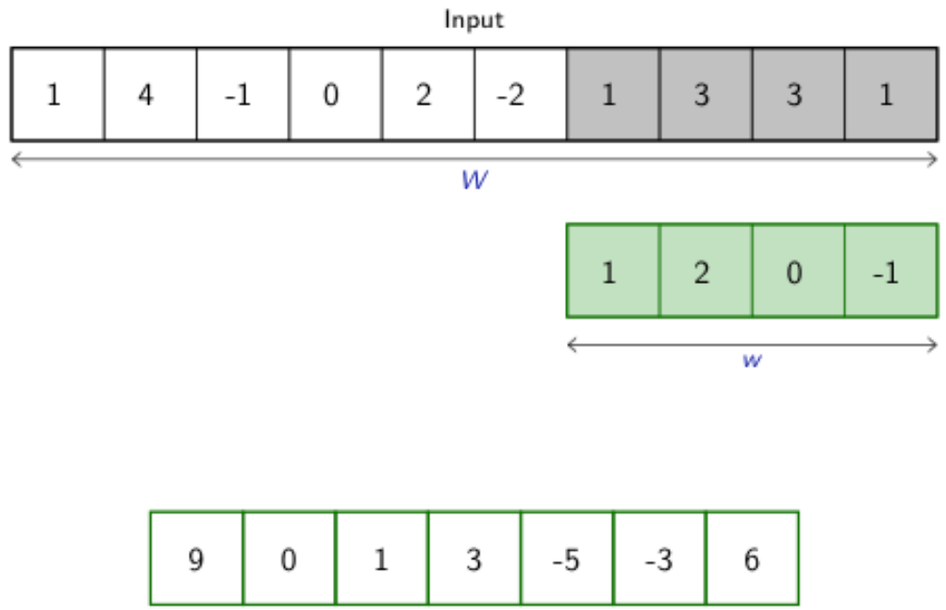
Convolution 1d



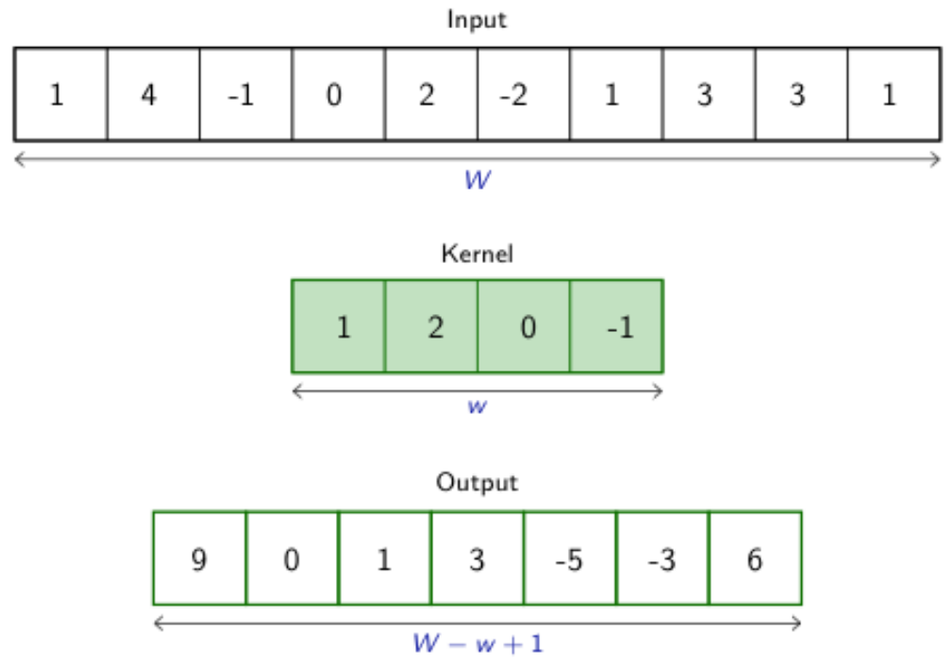
Convolution 1d



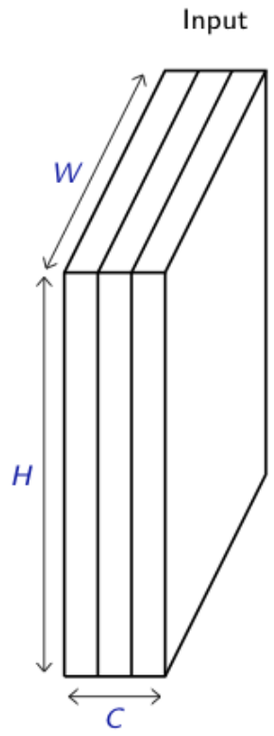
Convolution 1d



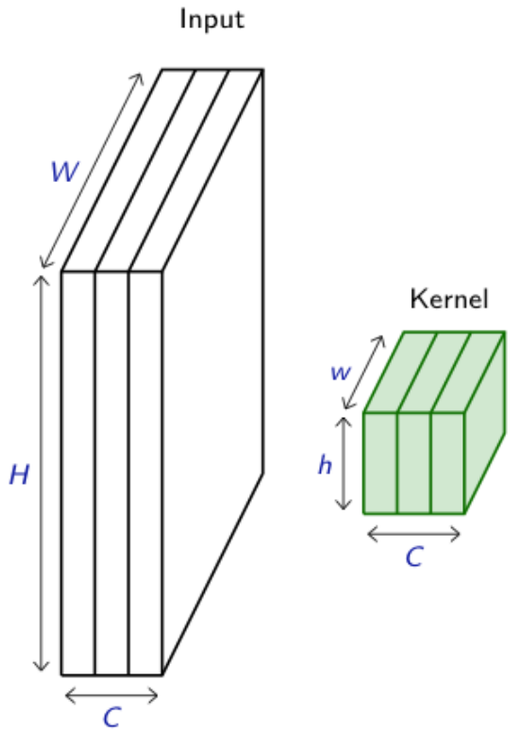
Convolution 1d



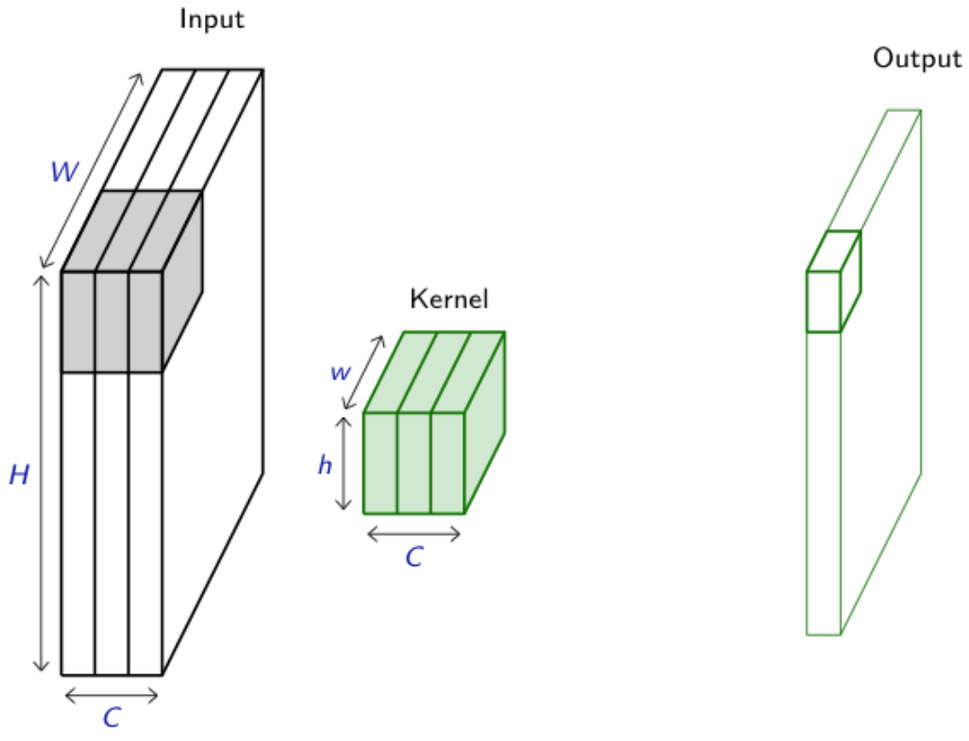
Convolution 2d



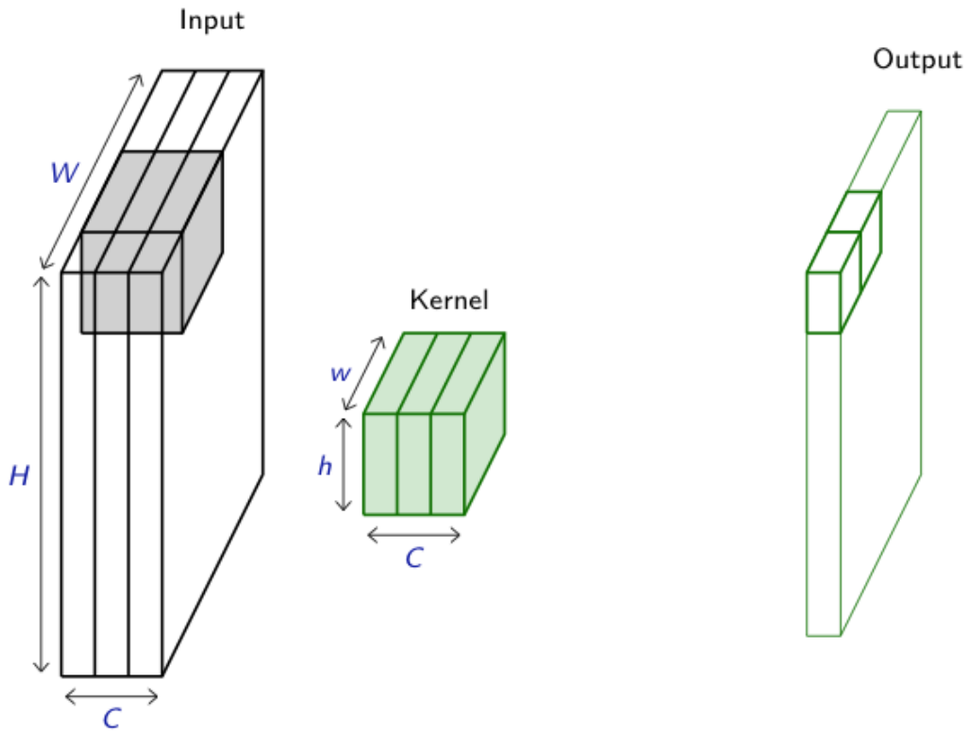
Convolution 2d



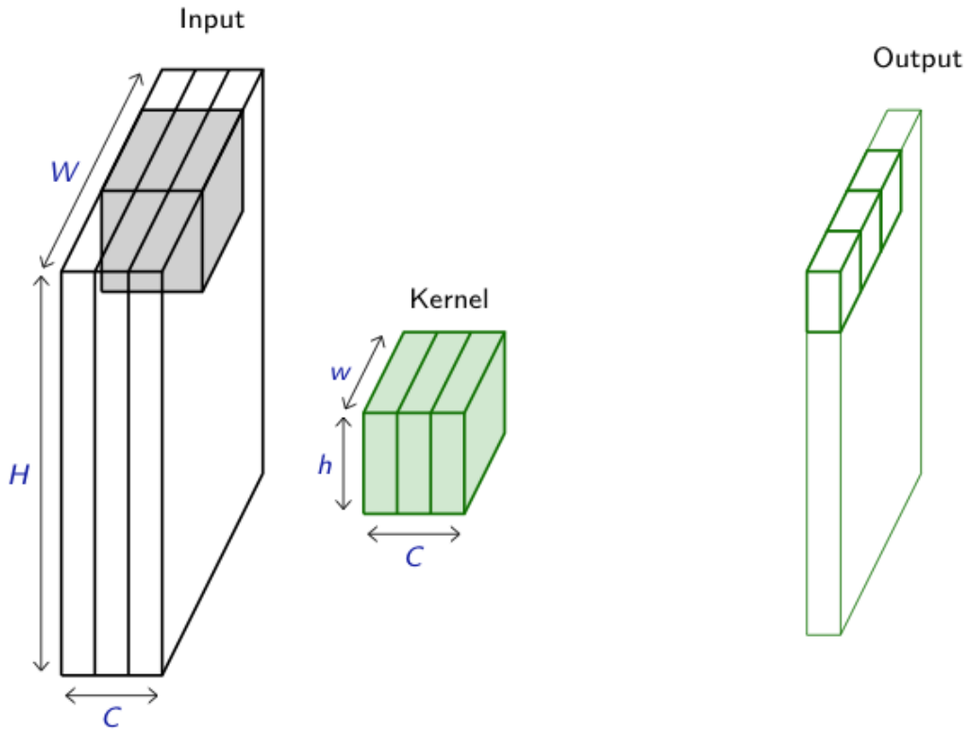
Convolution 2d



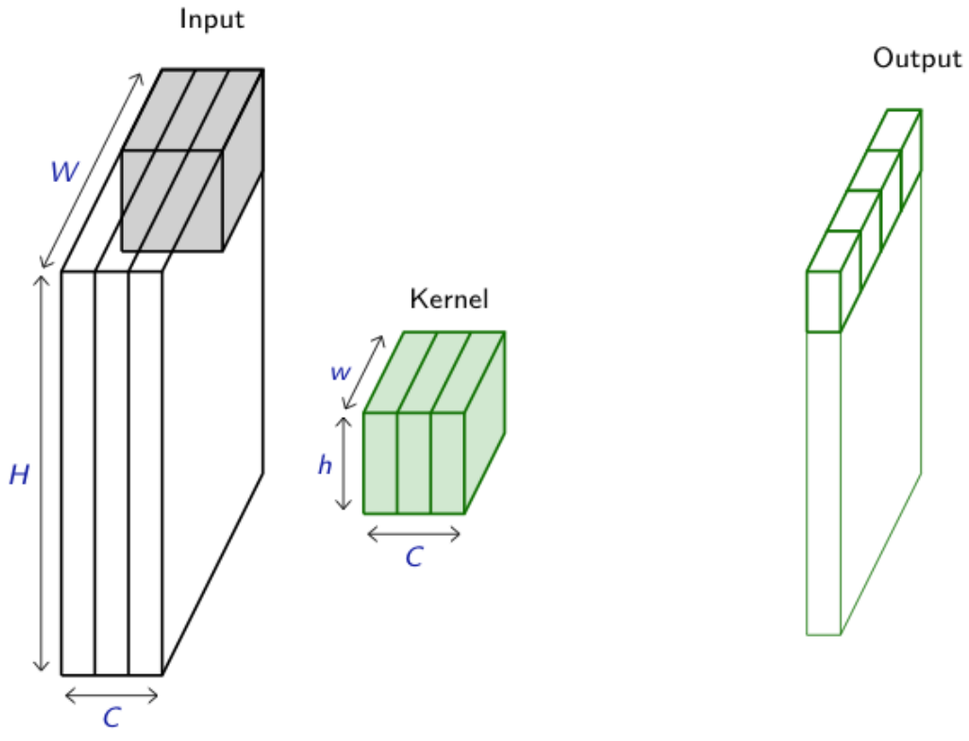
Convolution 2d



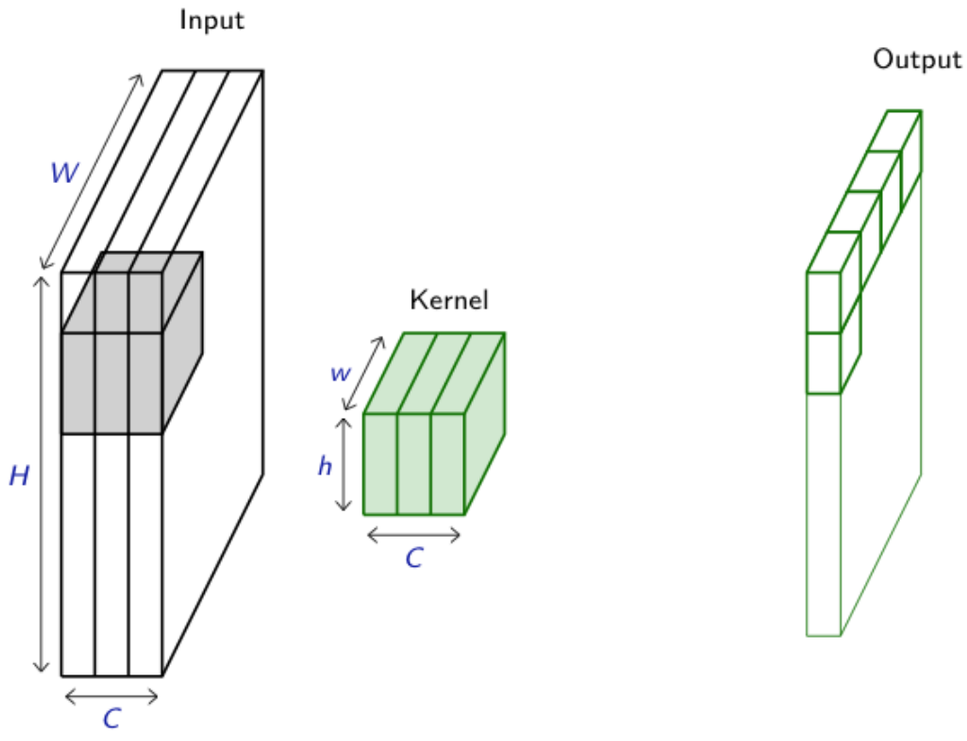
Convolution 2d



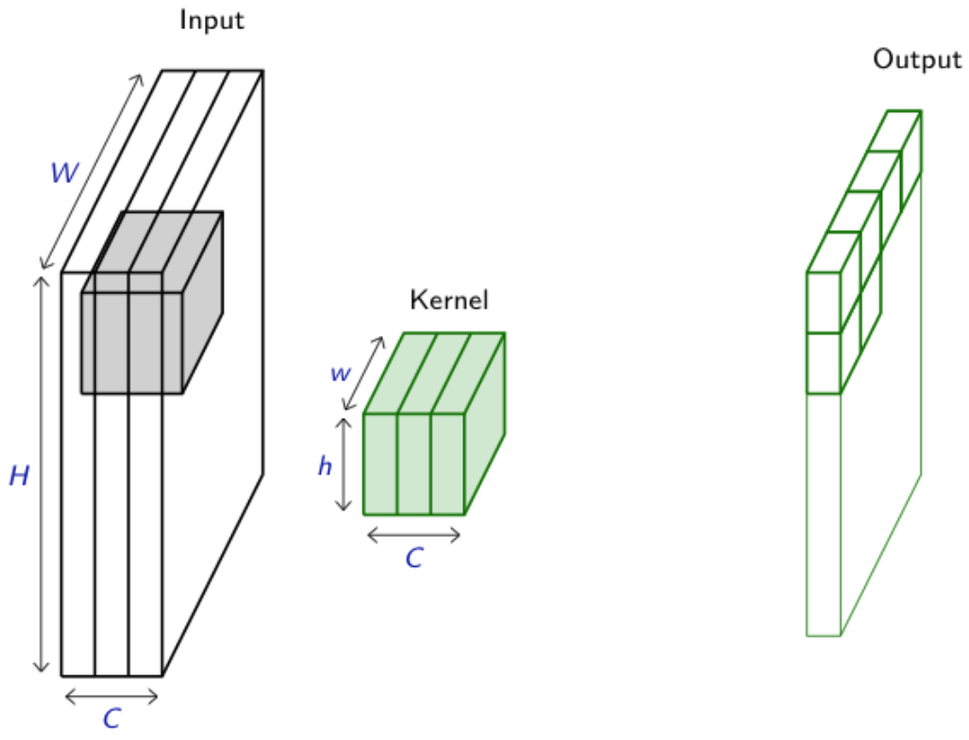
Convolution 2d



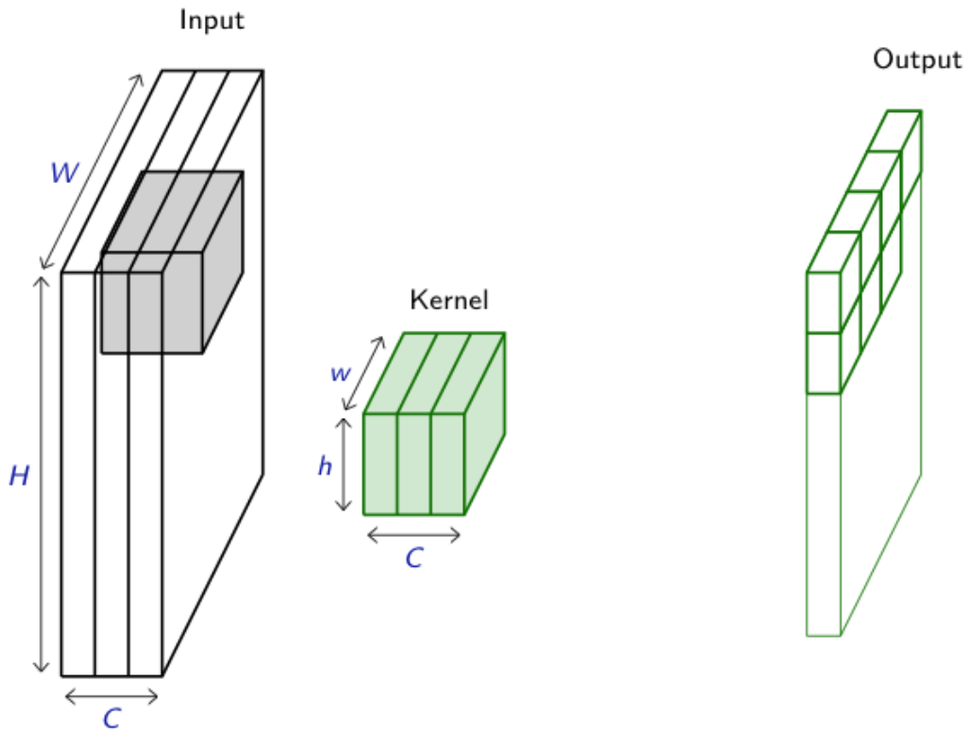
Convolution 2d



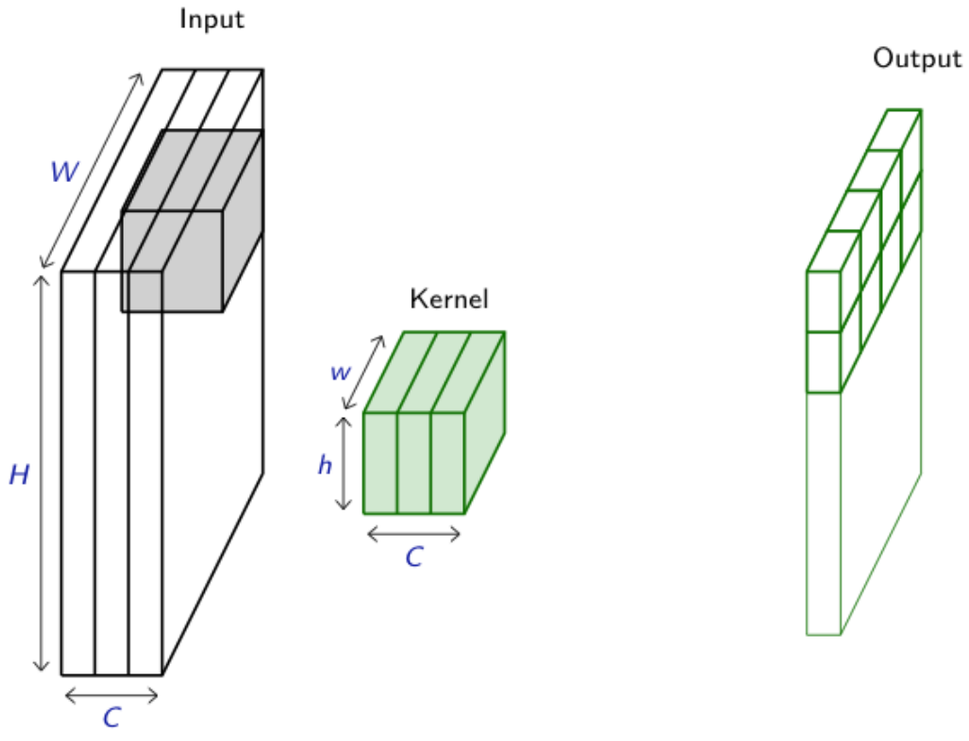
Convolution 2d



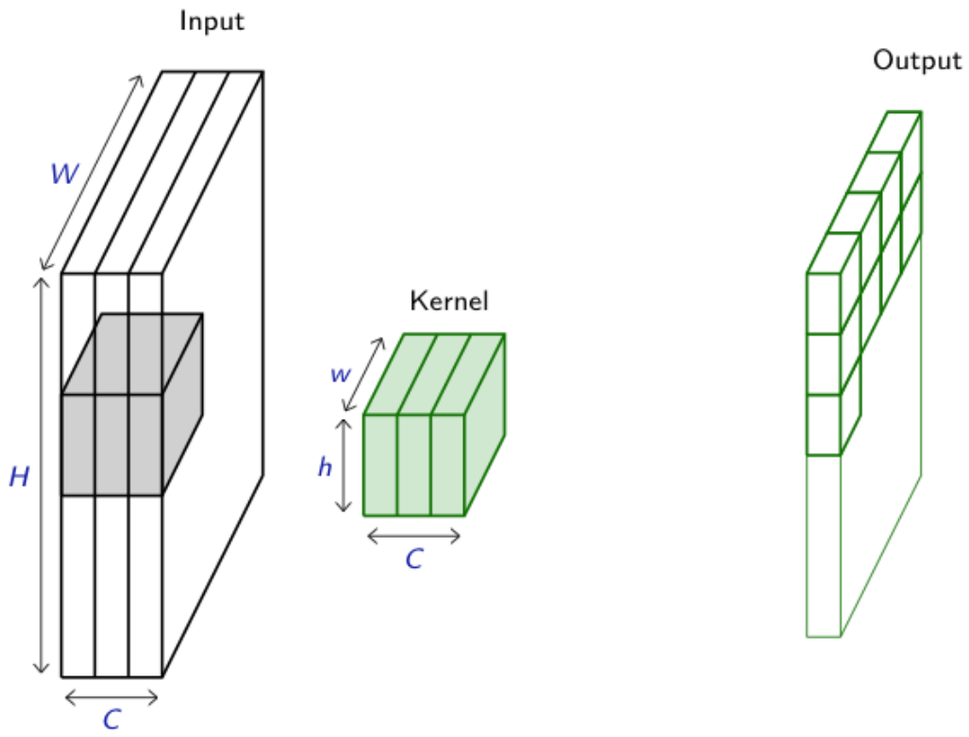
Convolution 2d



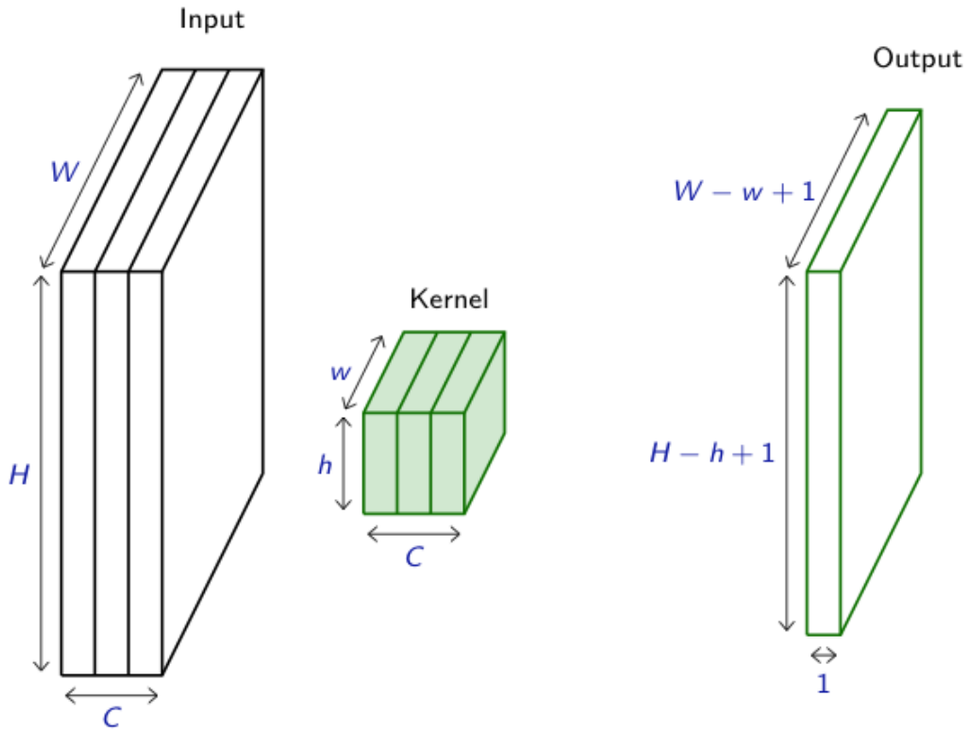
Convolution 2d



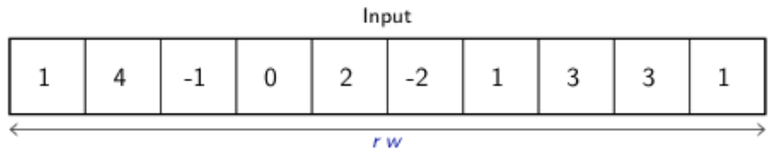
Convolution 2d



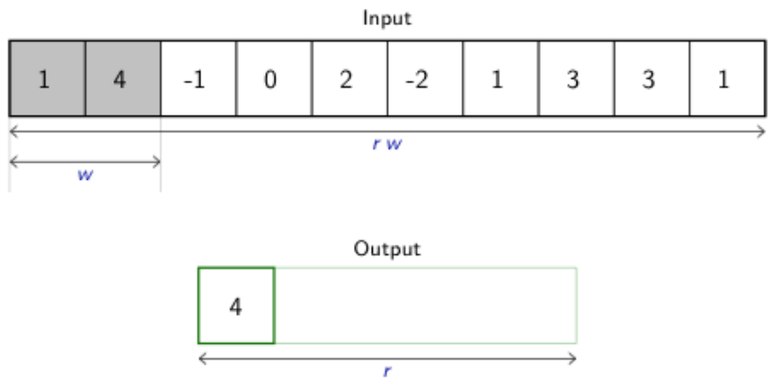
Convolution 2d



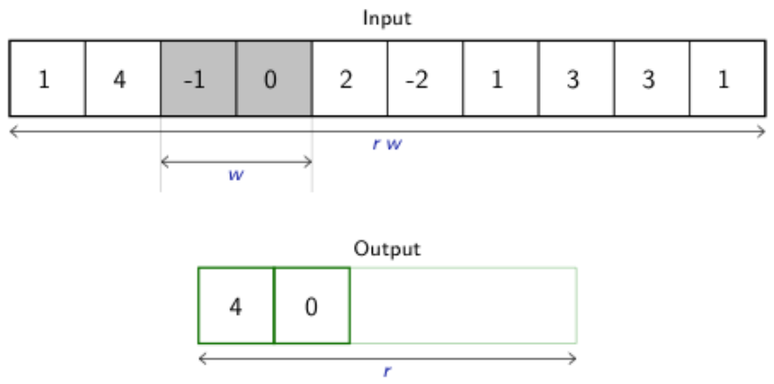
Max-Pooling 1d



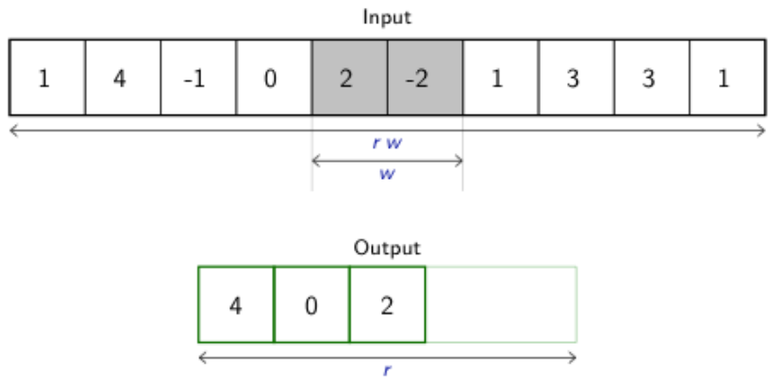
Max-Pooling 1d



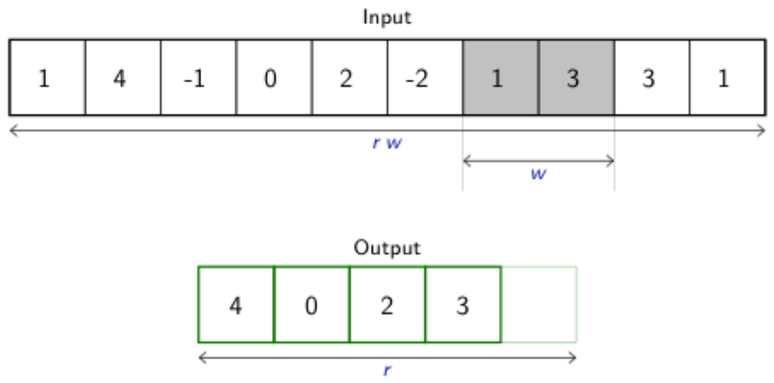
Max-Pooling 1d



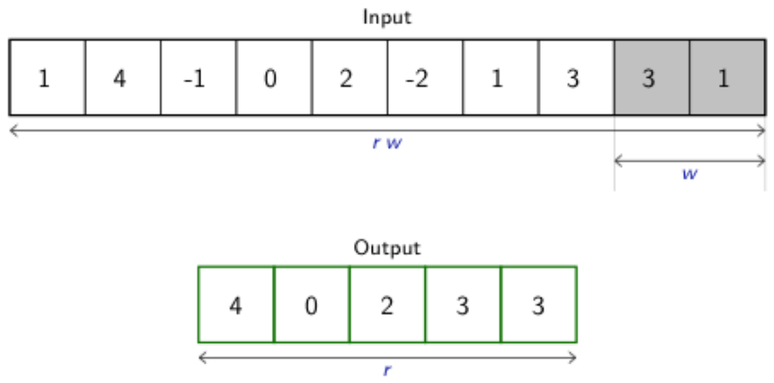
Max-Pooling 1d



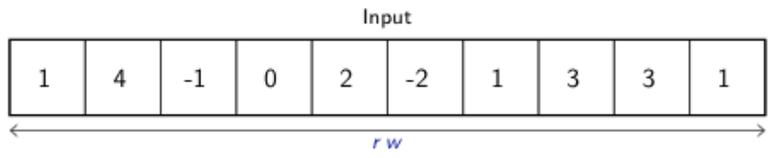
Max-Pooling 1d

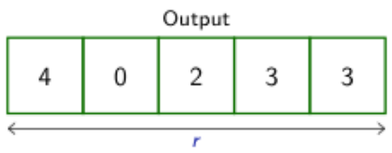


Max-Pooling 1d

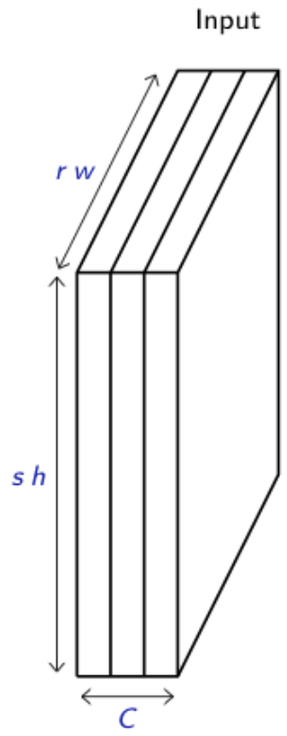


Max-Pooling 1d

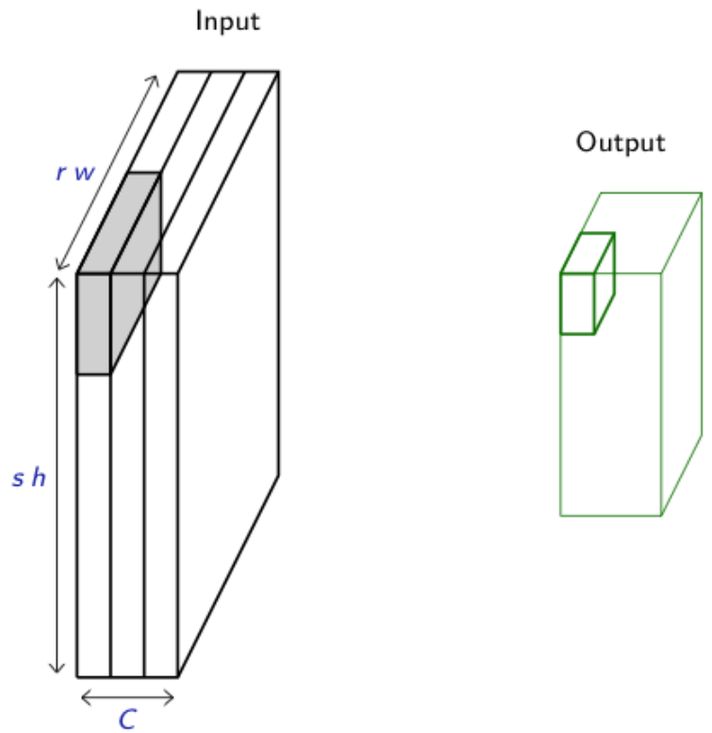




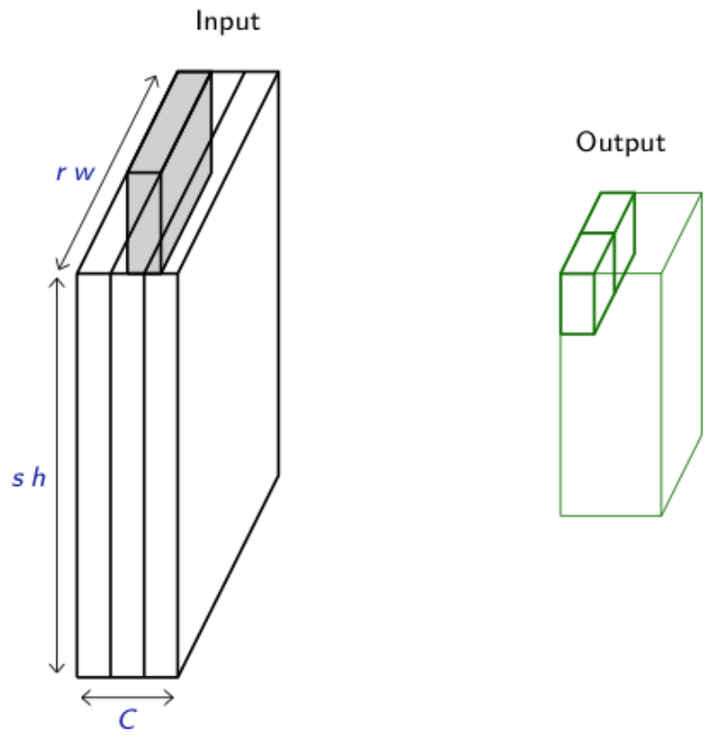
Max-Pooling 2d



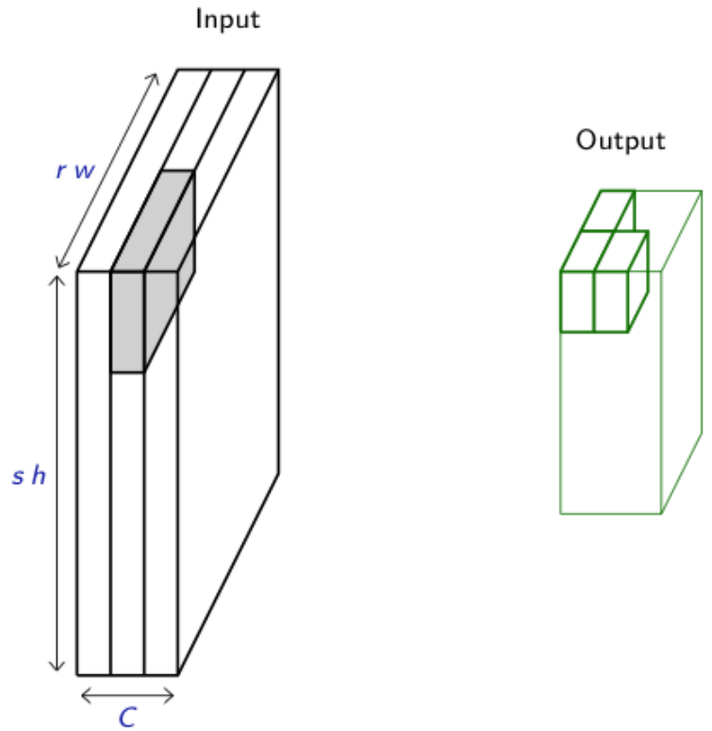
Max-Pooling 2d



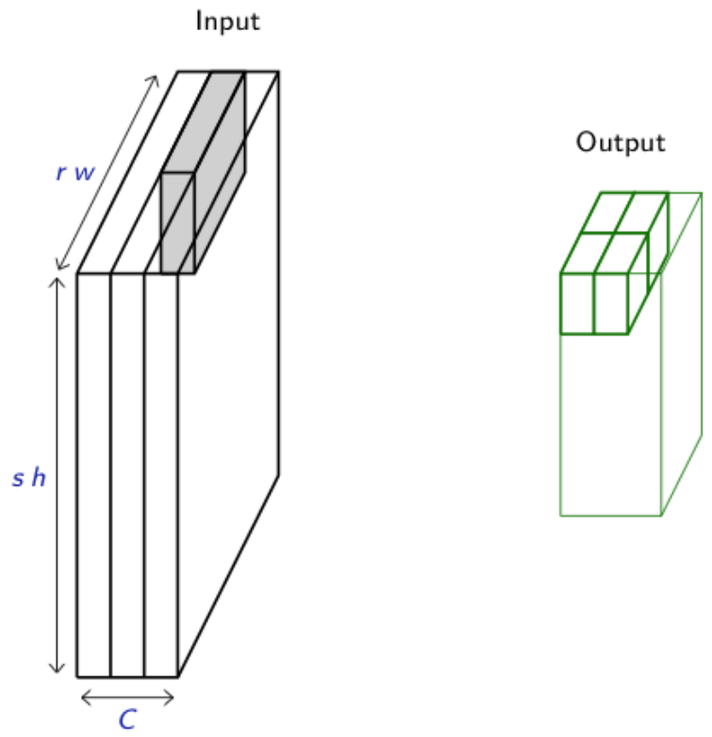
Max-Pooling 2d



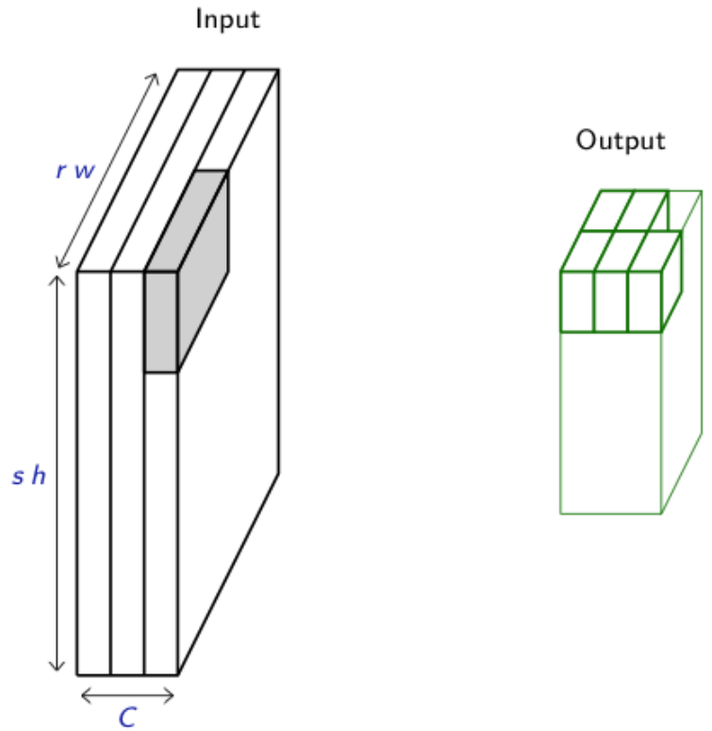
Max-Pooling 2d



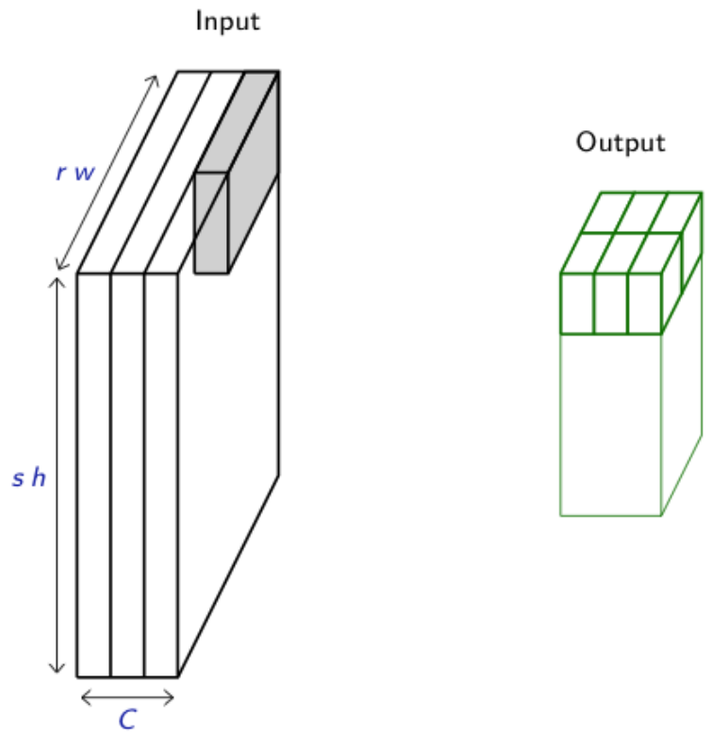
Max-Pooling 2d



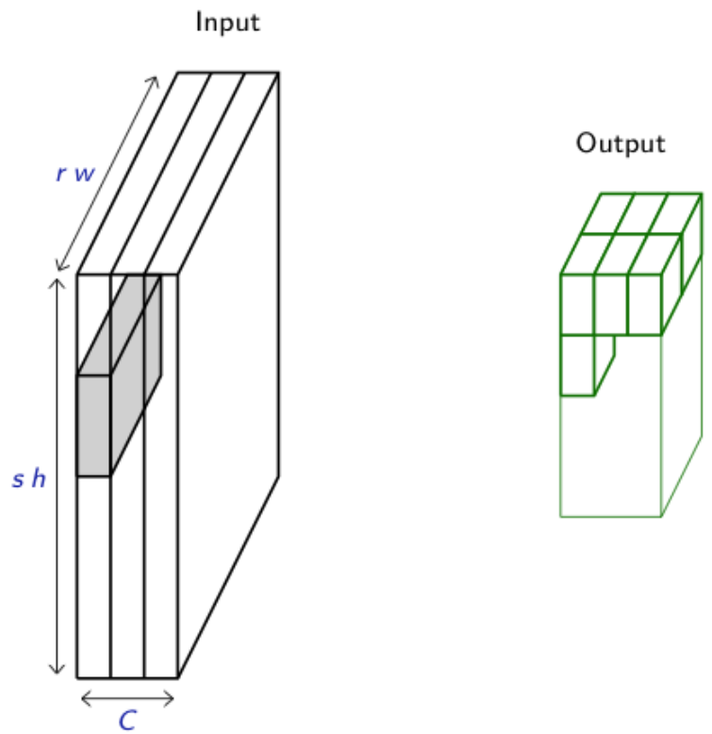
Max-Pooling 2d



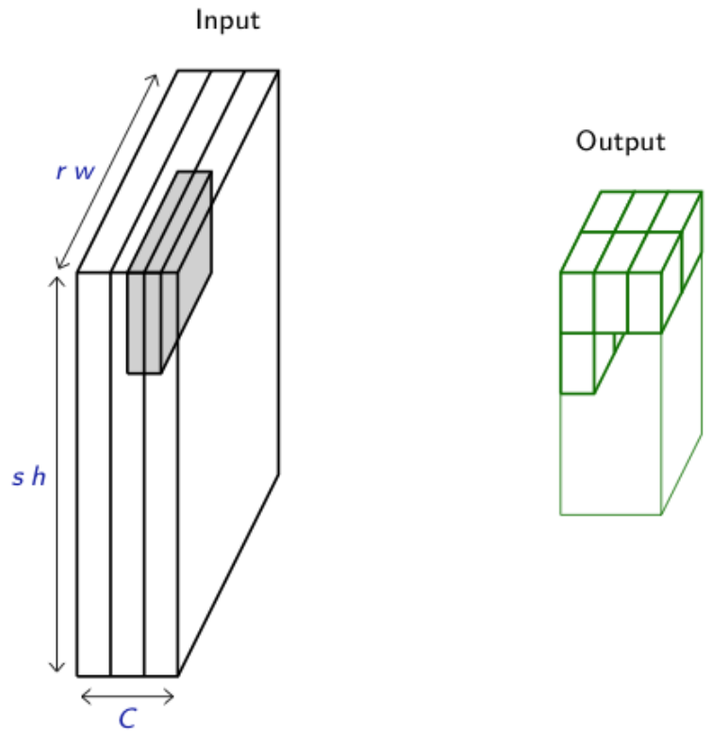
Max-Pooling 2d



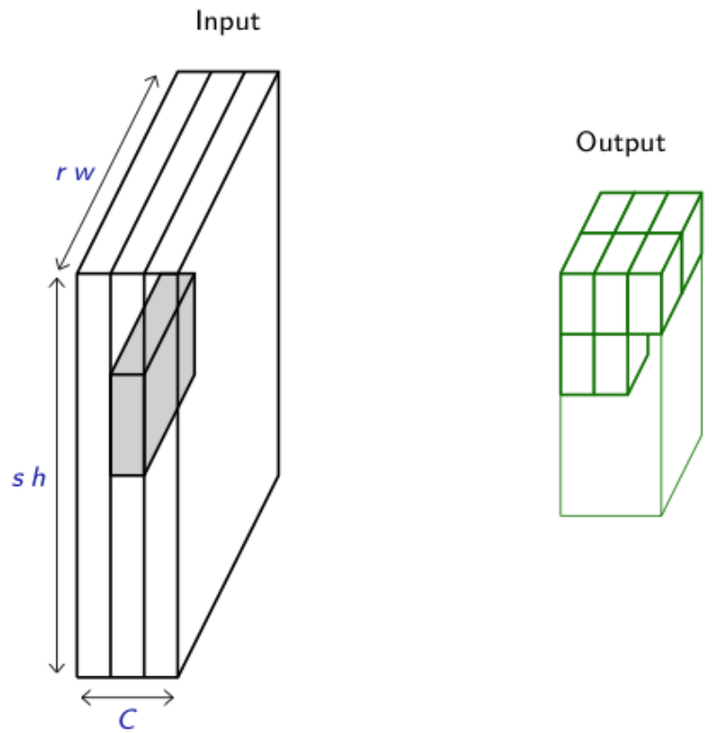
Max-Pooling 2d



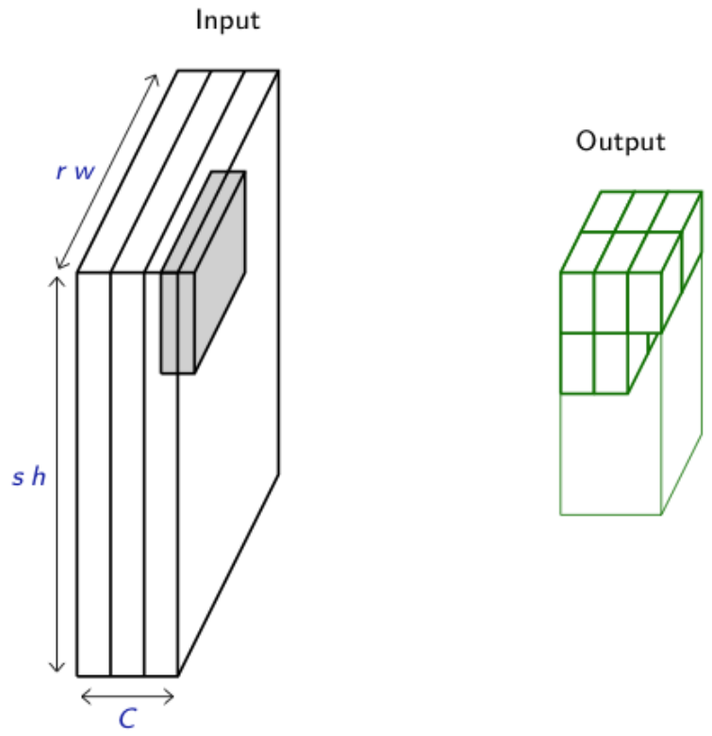
Max-Pooling 2d



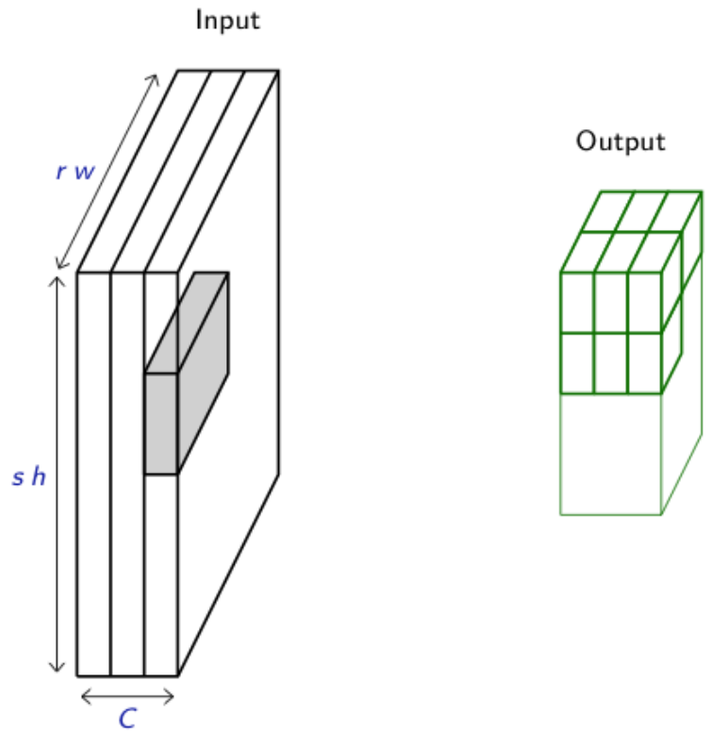
Max-Pooling 2d



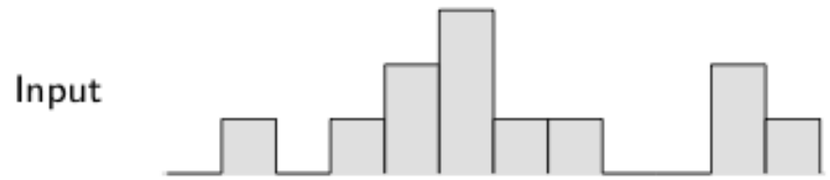
Max-Pooling 2d



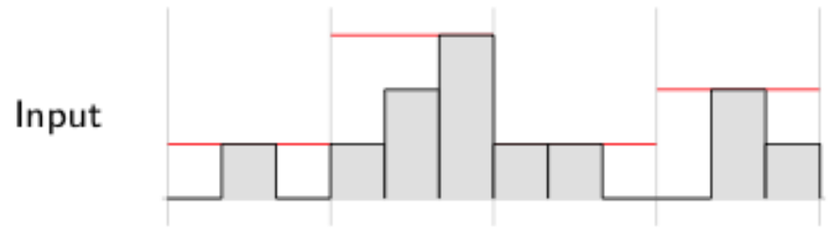
Max-Pooling 2d



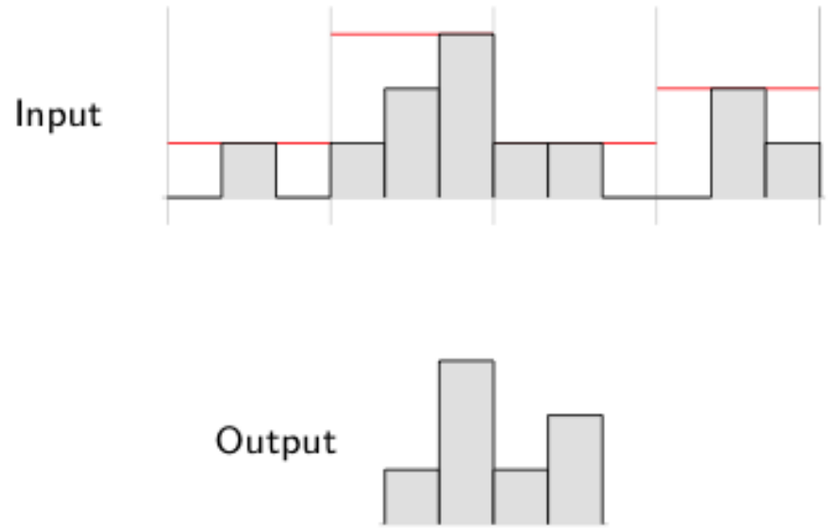
Translation invariance from pooling



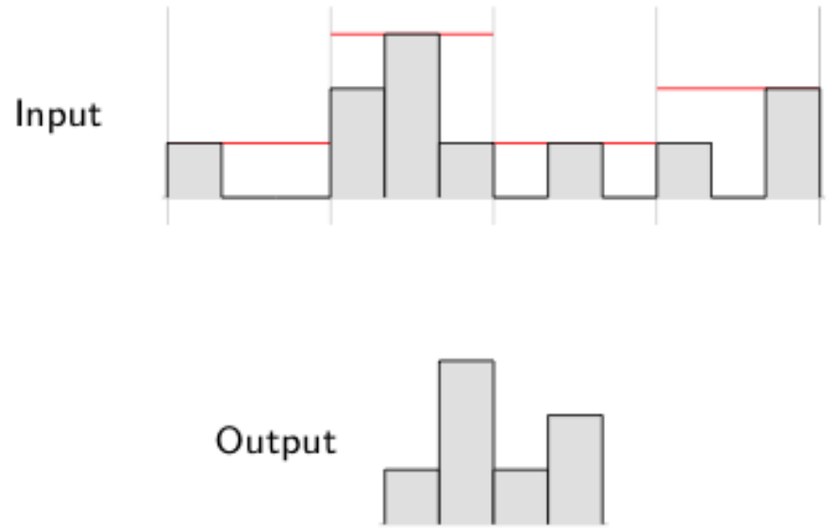
Translation invariance from pooling



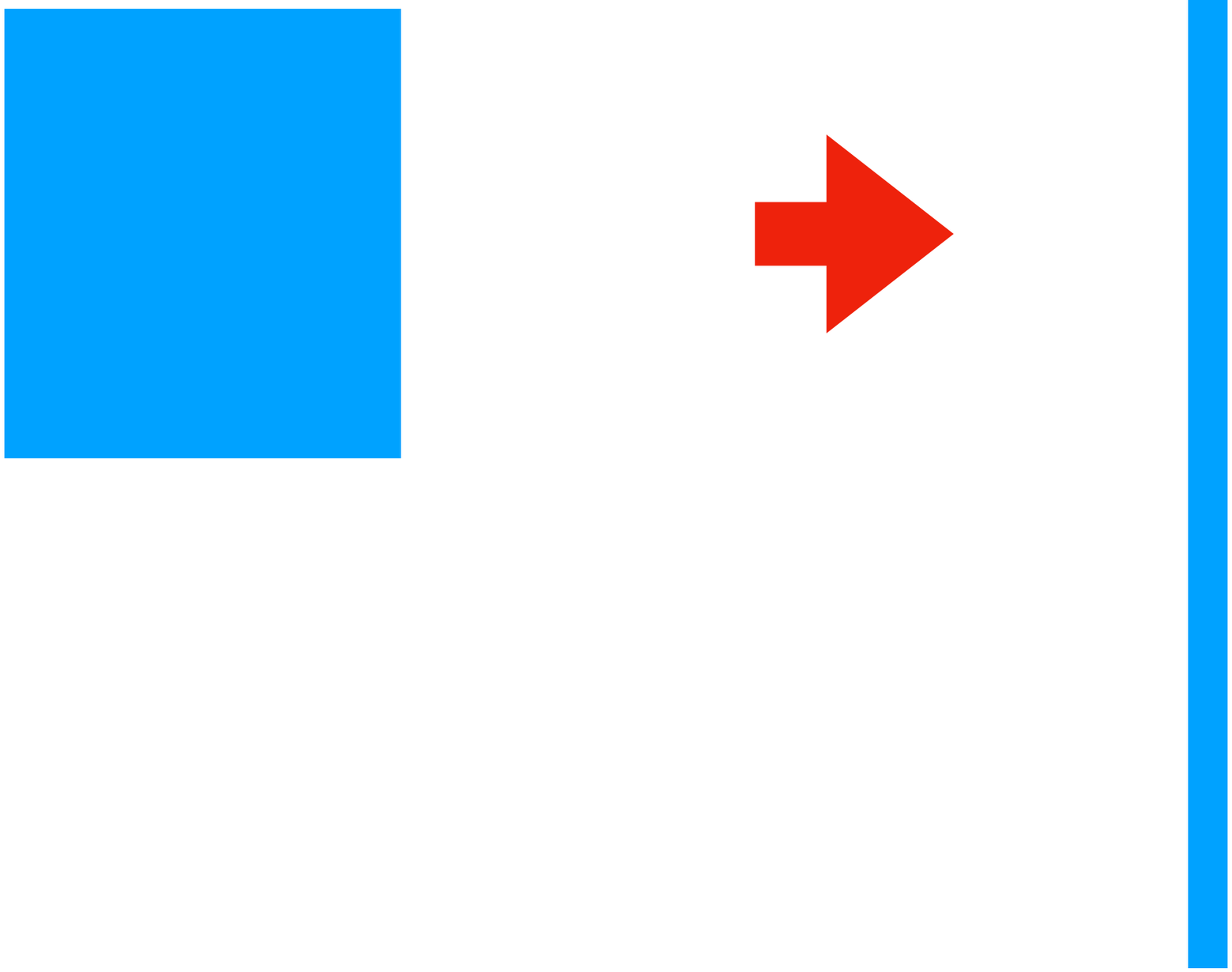
Translation invariance from pooling



Translation invariance from pooling



Flatten
2D->1D



Achitectures

4: conv-nets

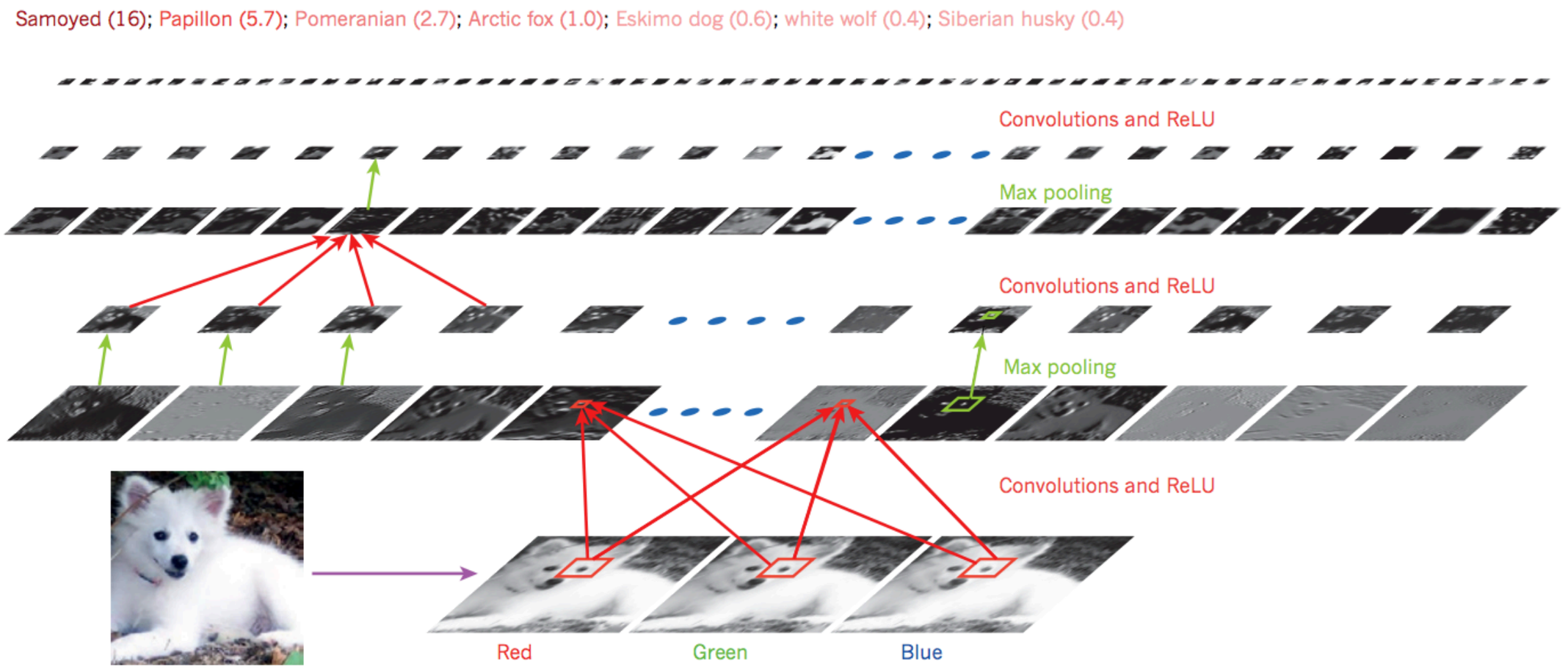
REVIEW

doi:10.1038/nature14539

Deep learning

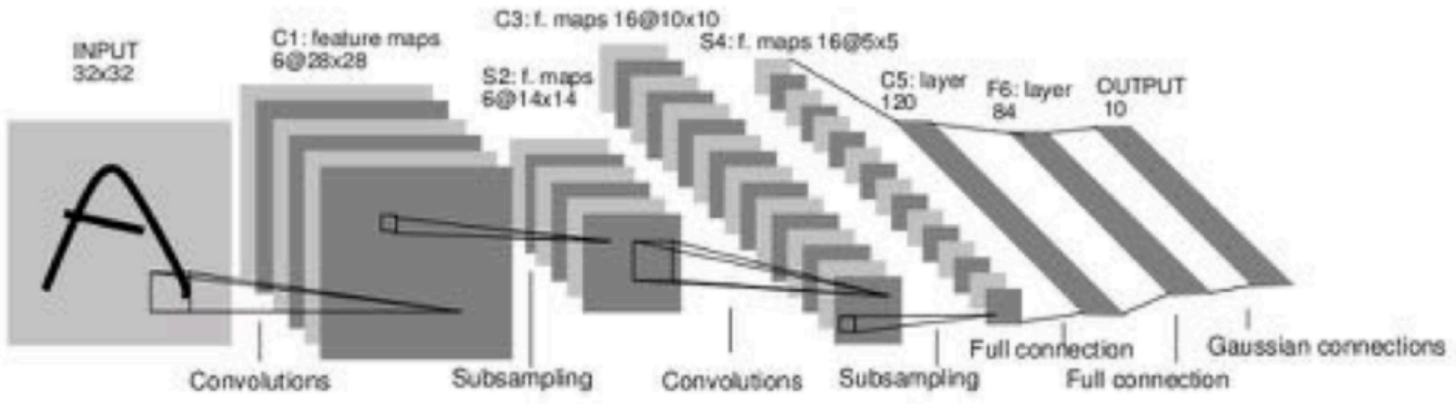
Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

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

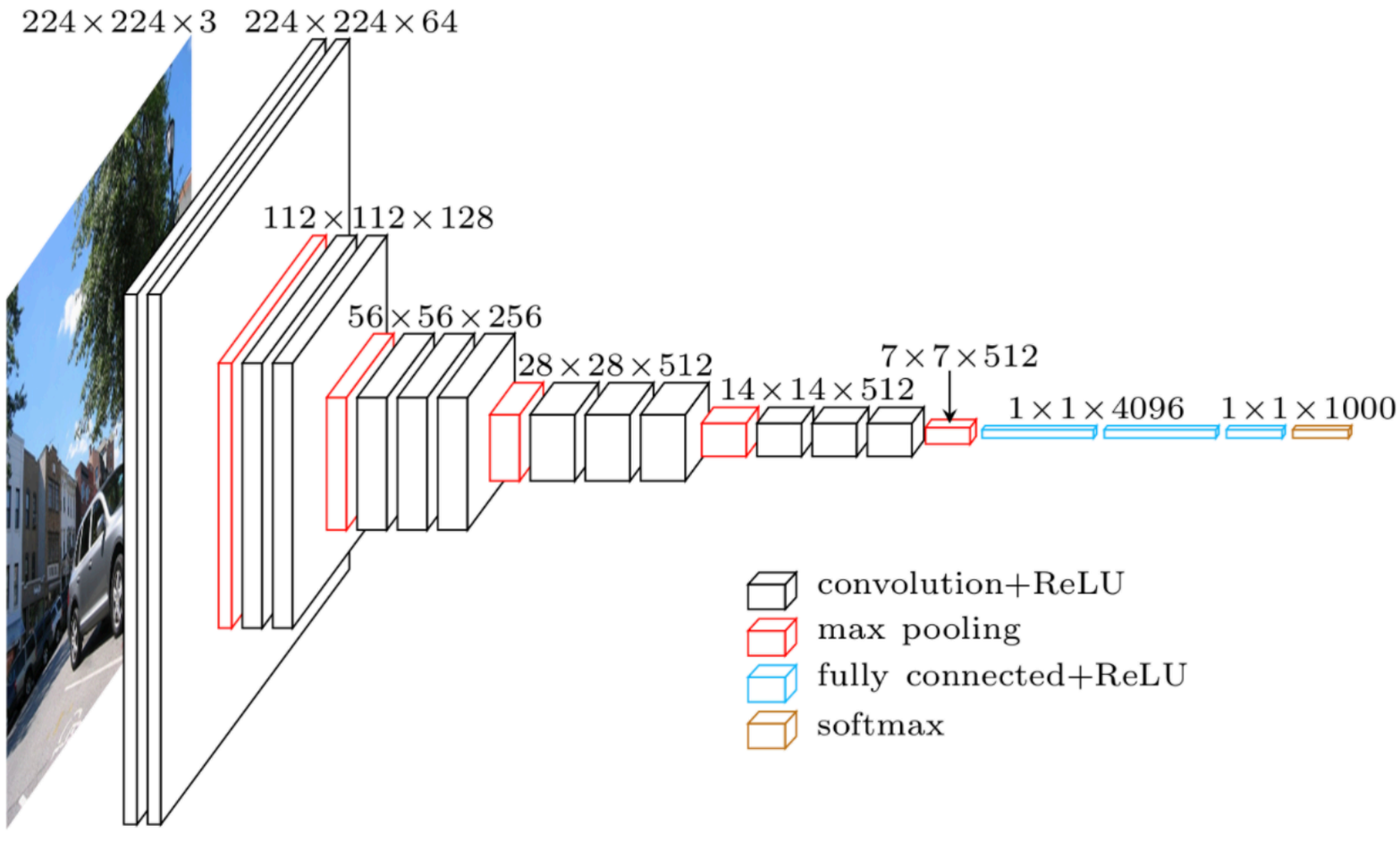
10 classes, input 1 x 28 x 28

```
(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

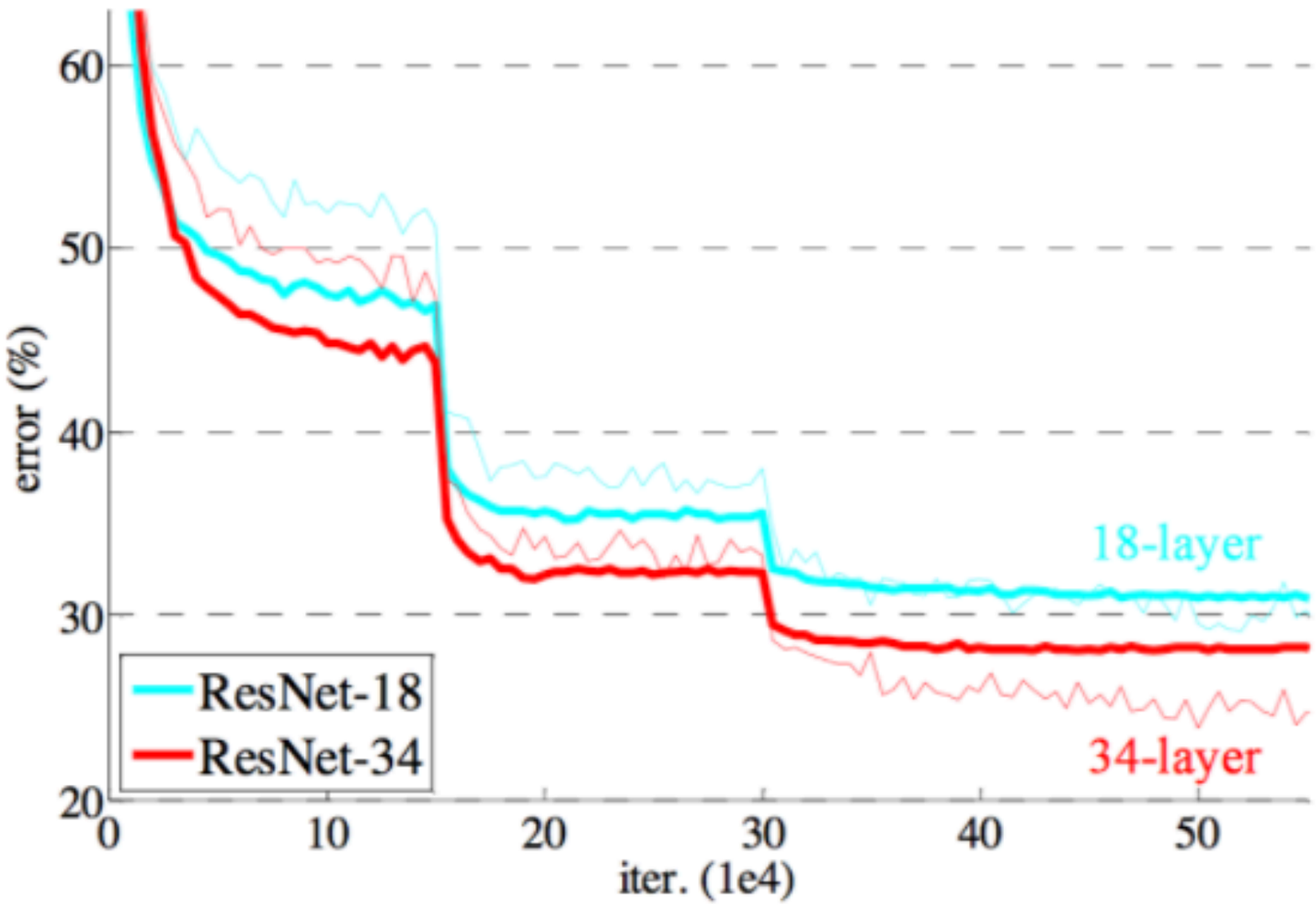
```
(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU (inplace)
(2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(3): ReLU (inplace)
(4): MaxPool2d (size=(2, 2), stride=(2, 2), dilation=(1, 1))
(5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), 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

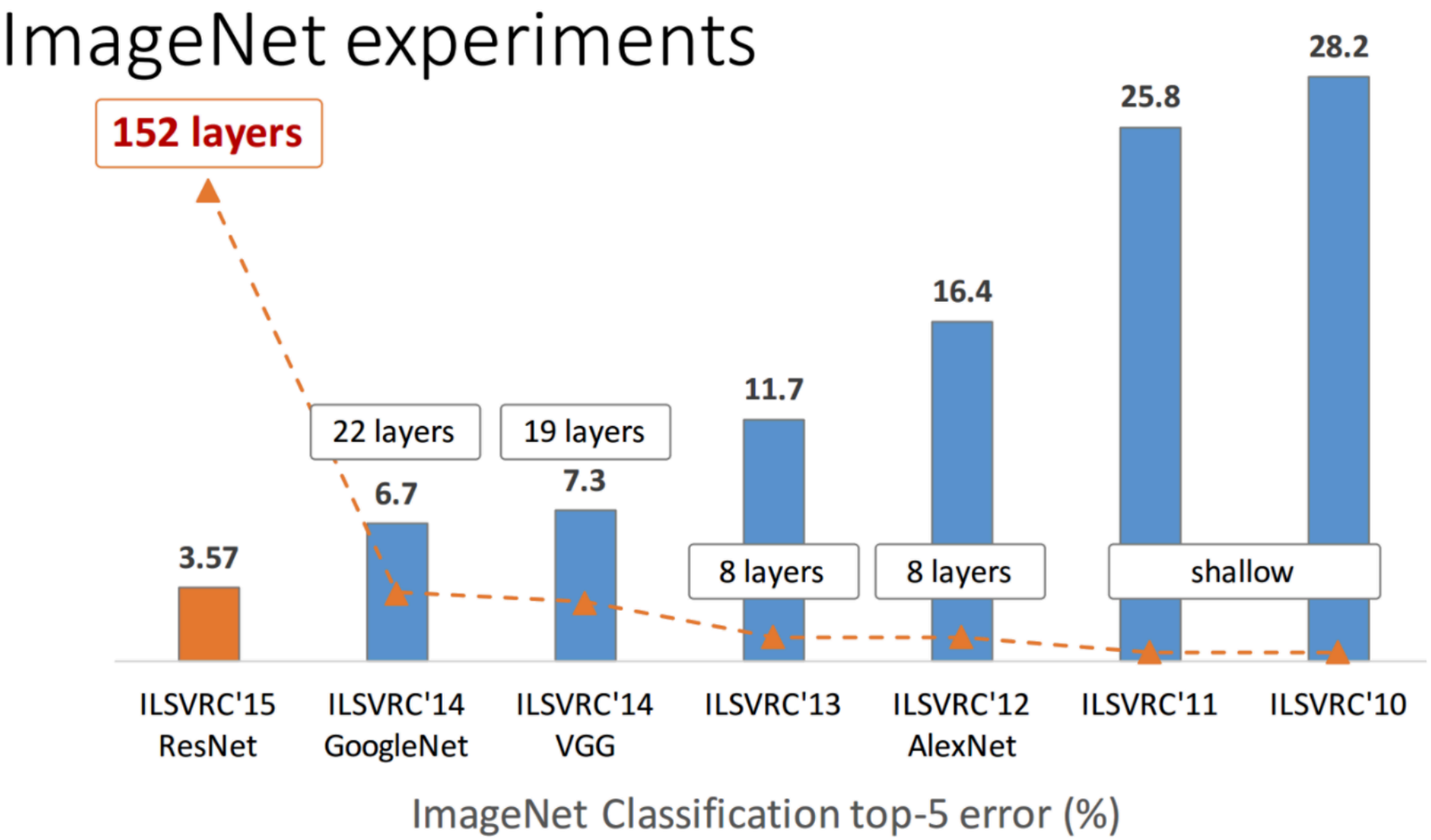
If we continue stacking more layers on a CNN:

Deeper models are harder to optimize

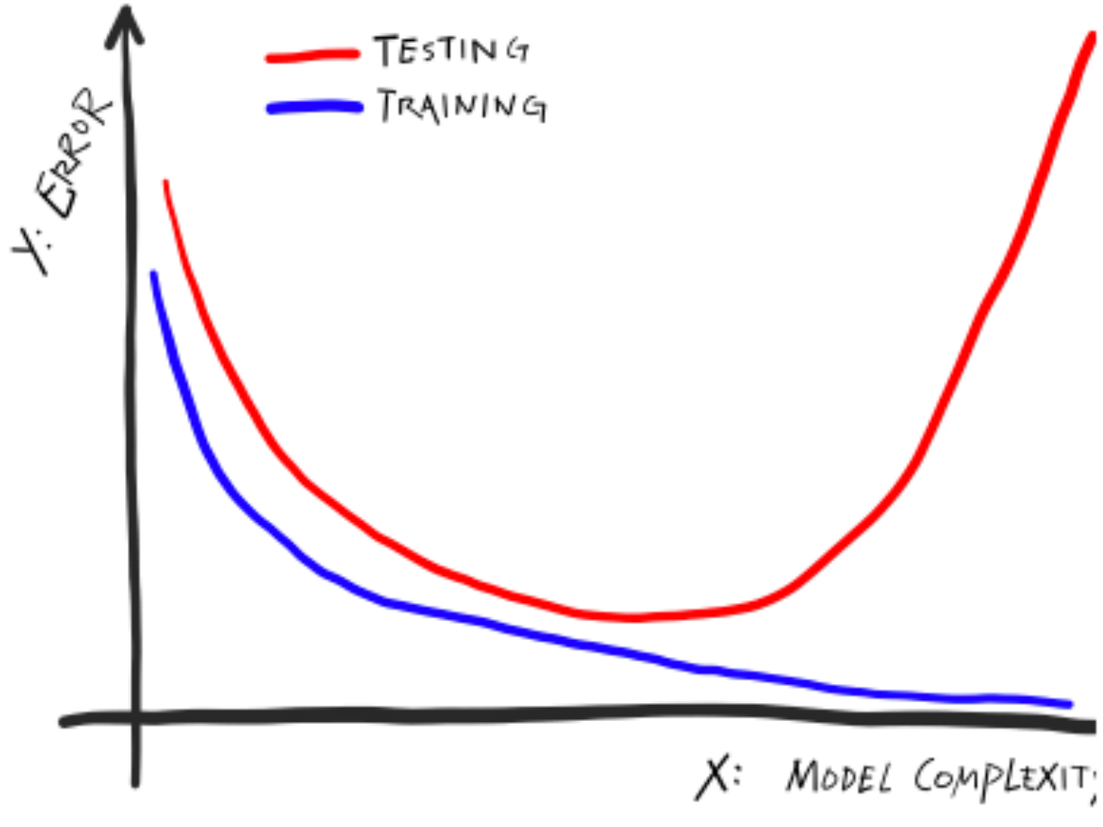
Resnets: Skiped-connections



Deeper is better



Regularization



Deep Learning

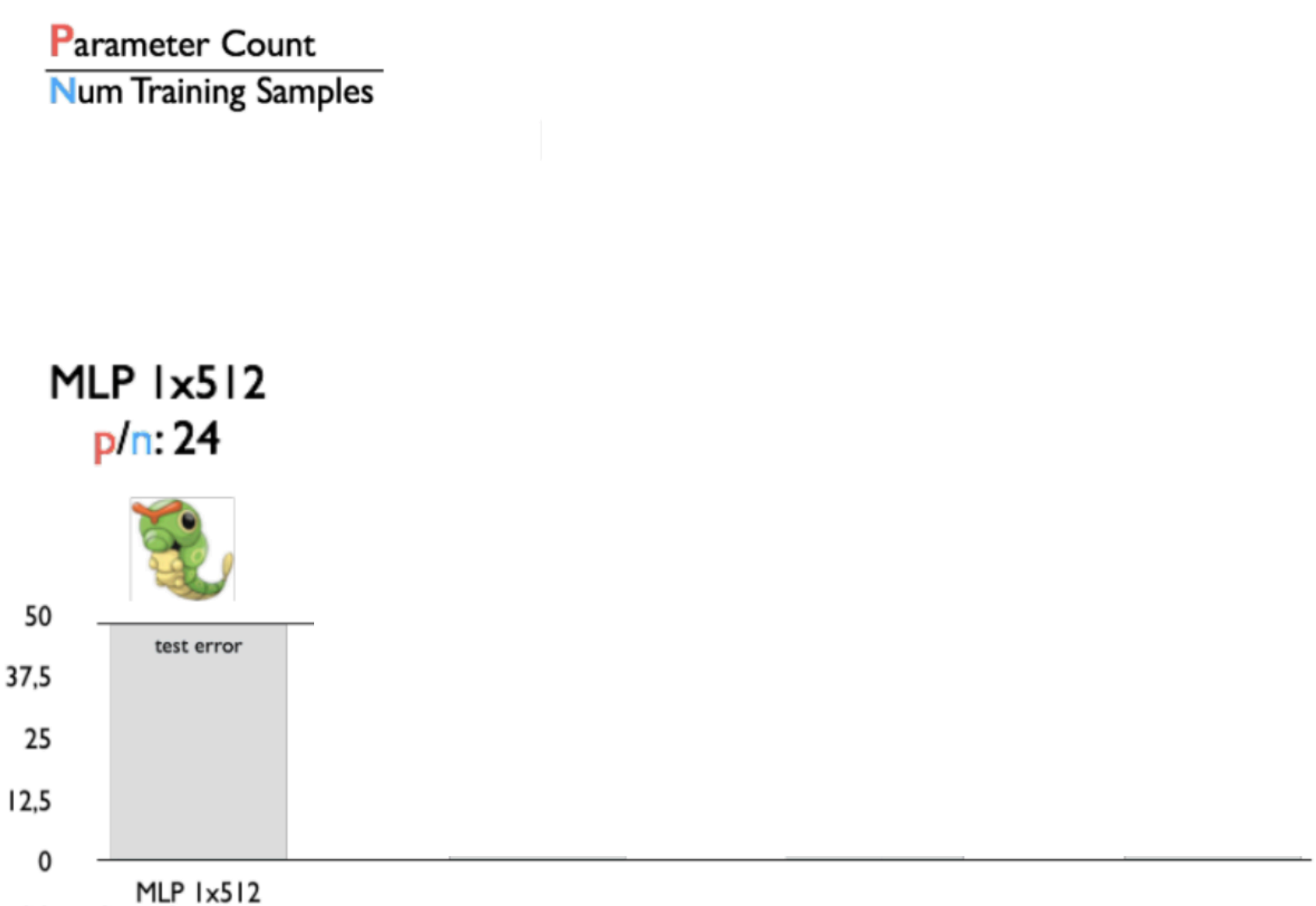
Slide credit: C. Zhang

Regularization

Parameter Count
Num Training Samples

MLP 1x512

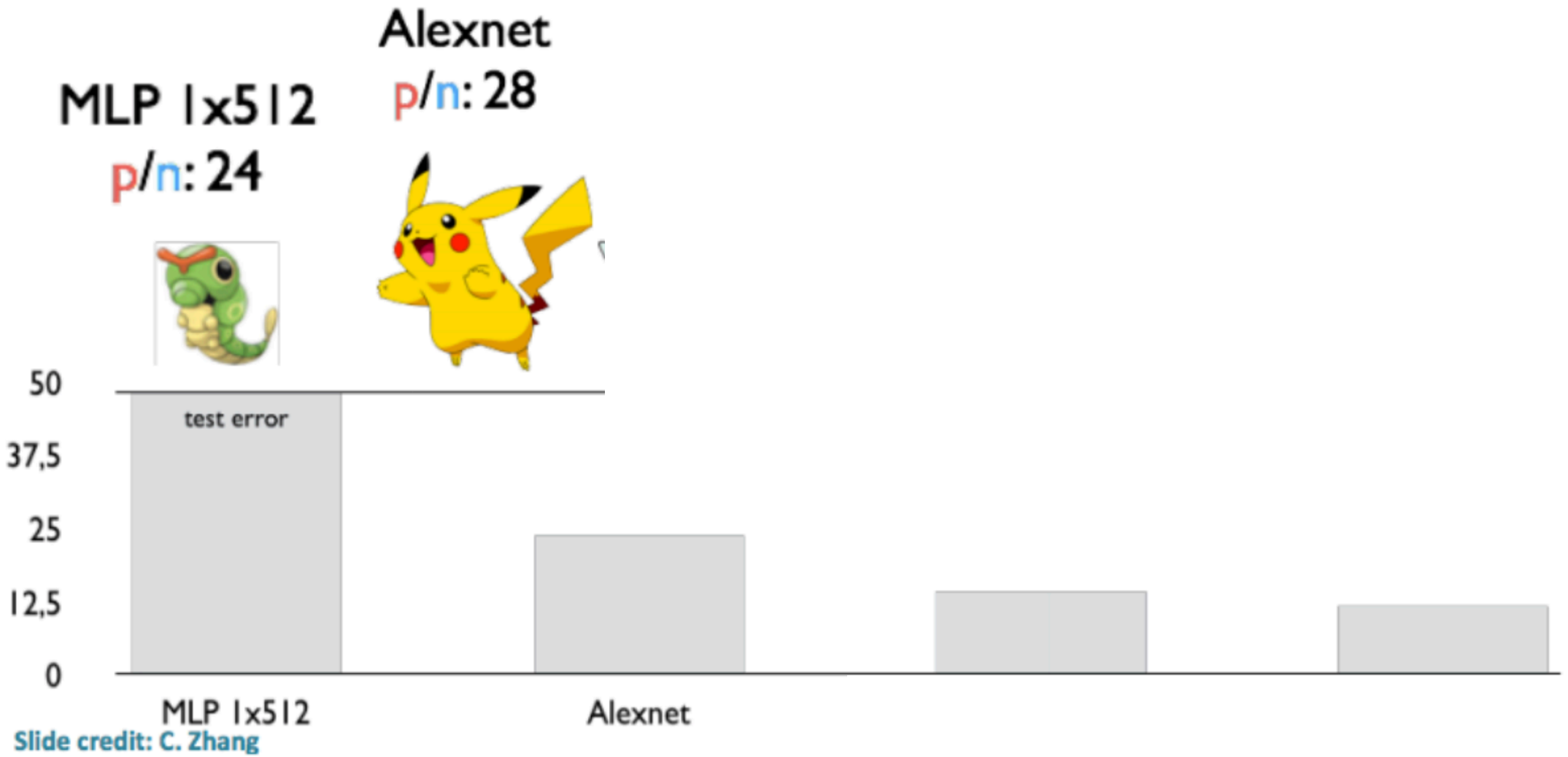
p/n: 24



Slide credit: C. Zhang

Regularization

$$\frac{\text{Parameter Count}}{\text{Num Training Samples}}$$

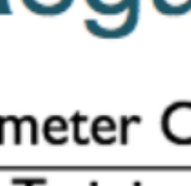


Slide credit: C. Zhang

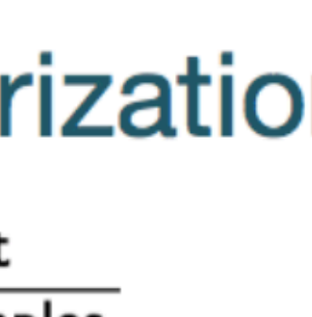
Regularization

Parameter Count
Num Training Samples

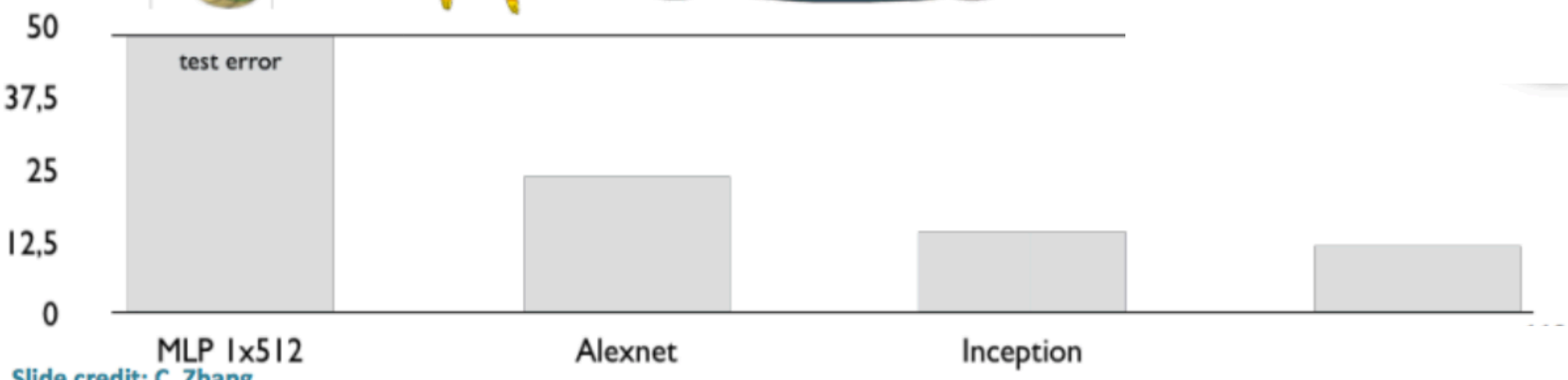
MLP 1x512
p/n: 24



Alexnet
p/n: 28



Inceptior
p/n: 33

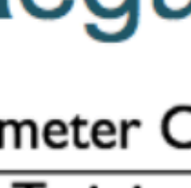


Slide credit: C. Zhang

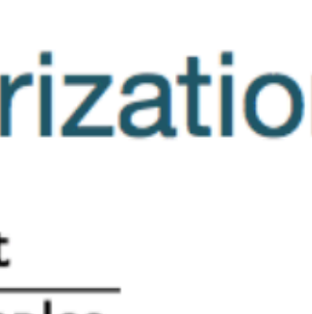
Regularization

Parameter Count
Num Training Samples

MLP 1x512
p/n: 24

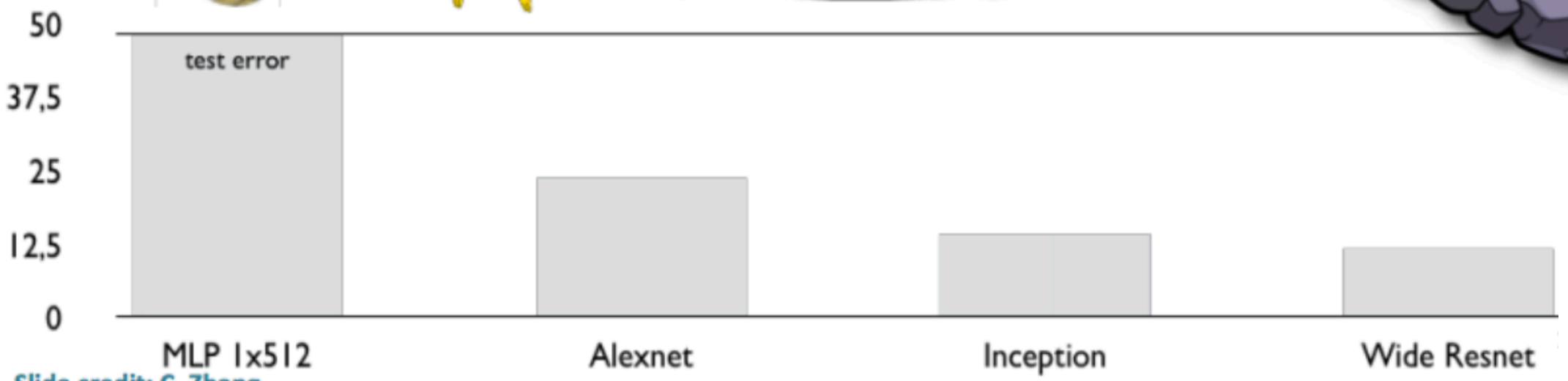


Alexnet
p/n: 28



Inception
p/n: 33

Wide Resnet
p/n: 179

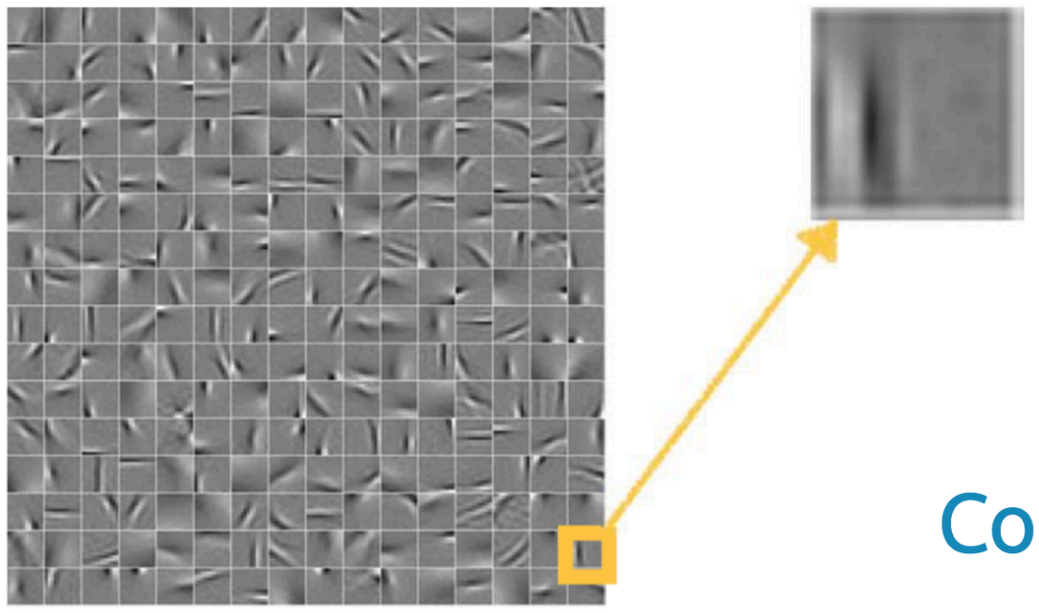


Slide credit: C. Zhang

What is learned in conv-nets?

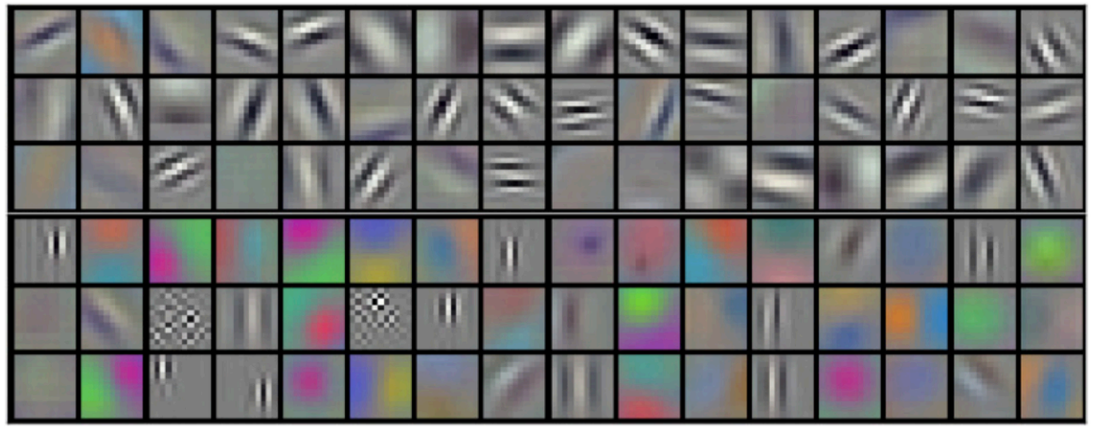
Convolutions

- 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

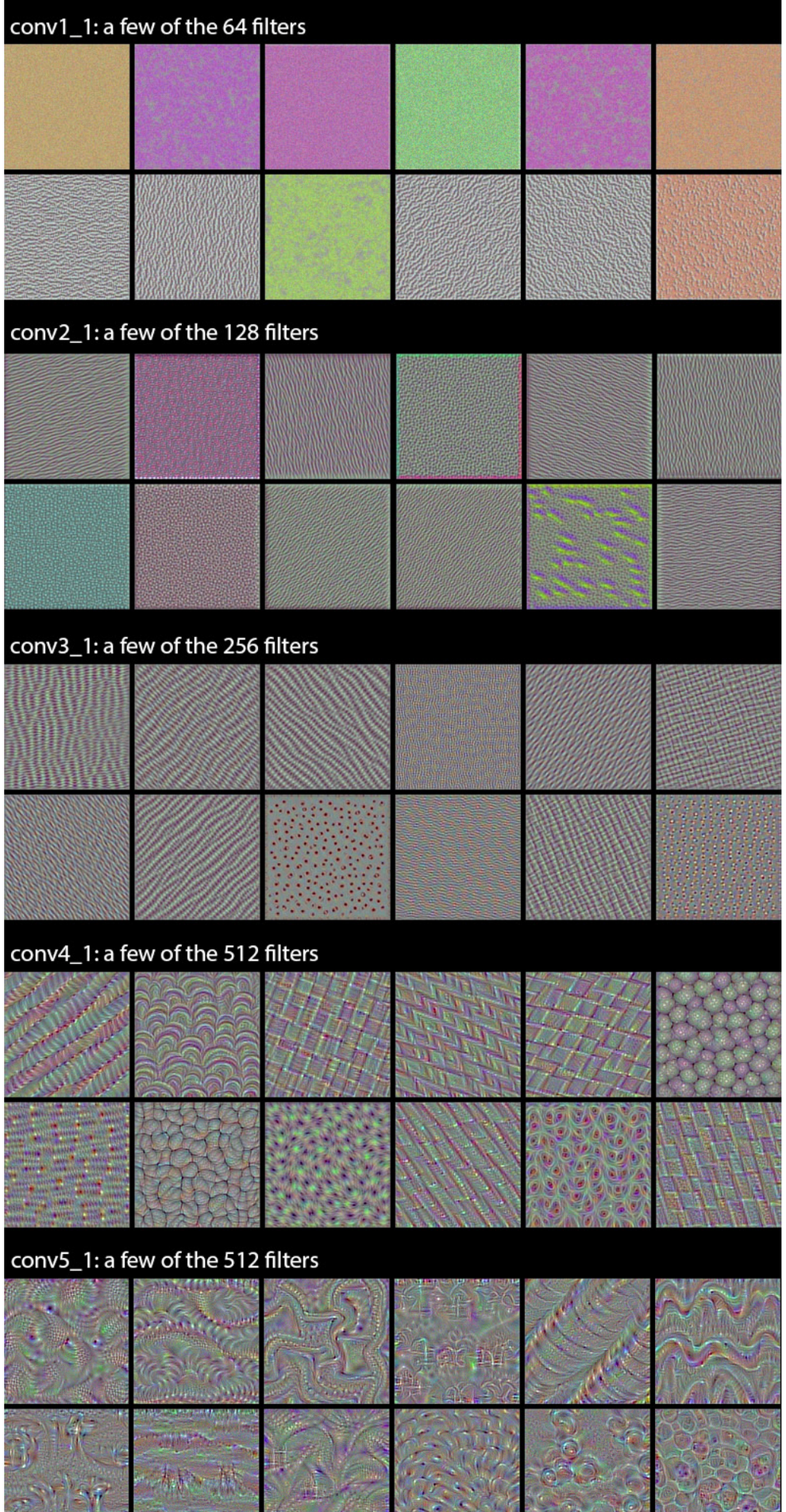


Convolutions

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

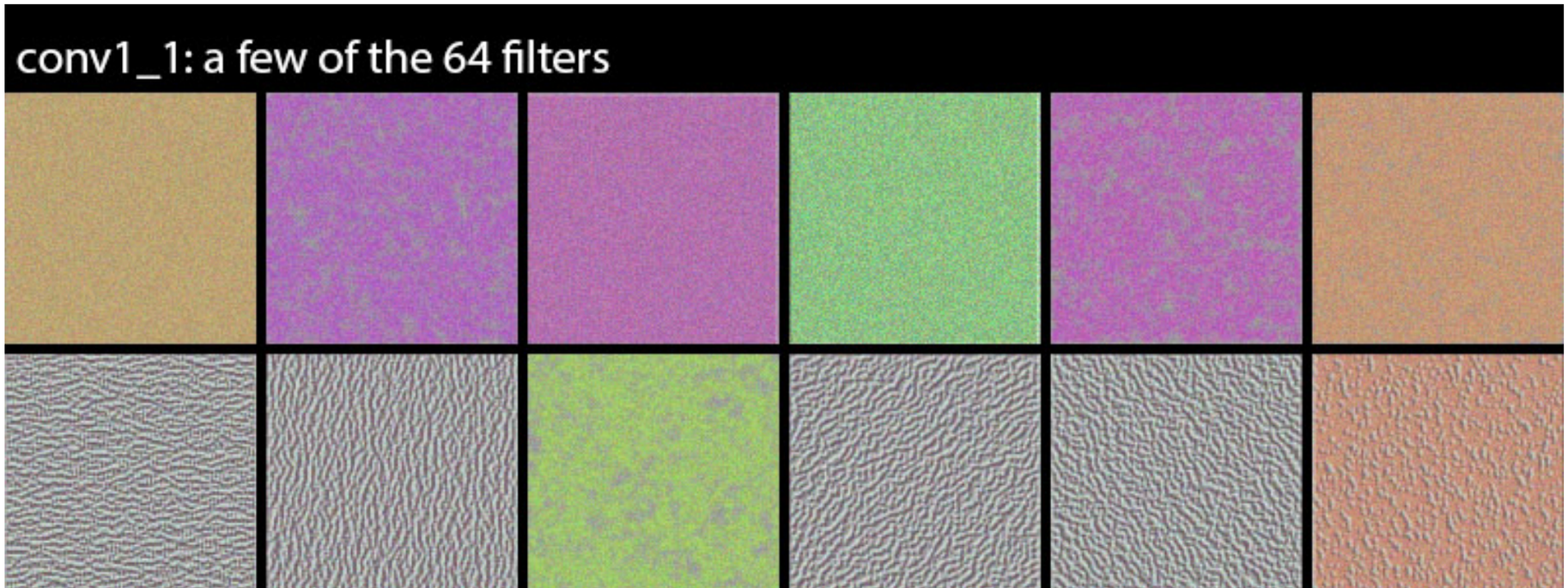


What sort of images maximise the activity for a given neutron in each layers?



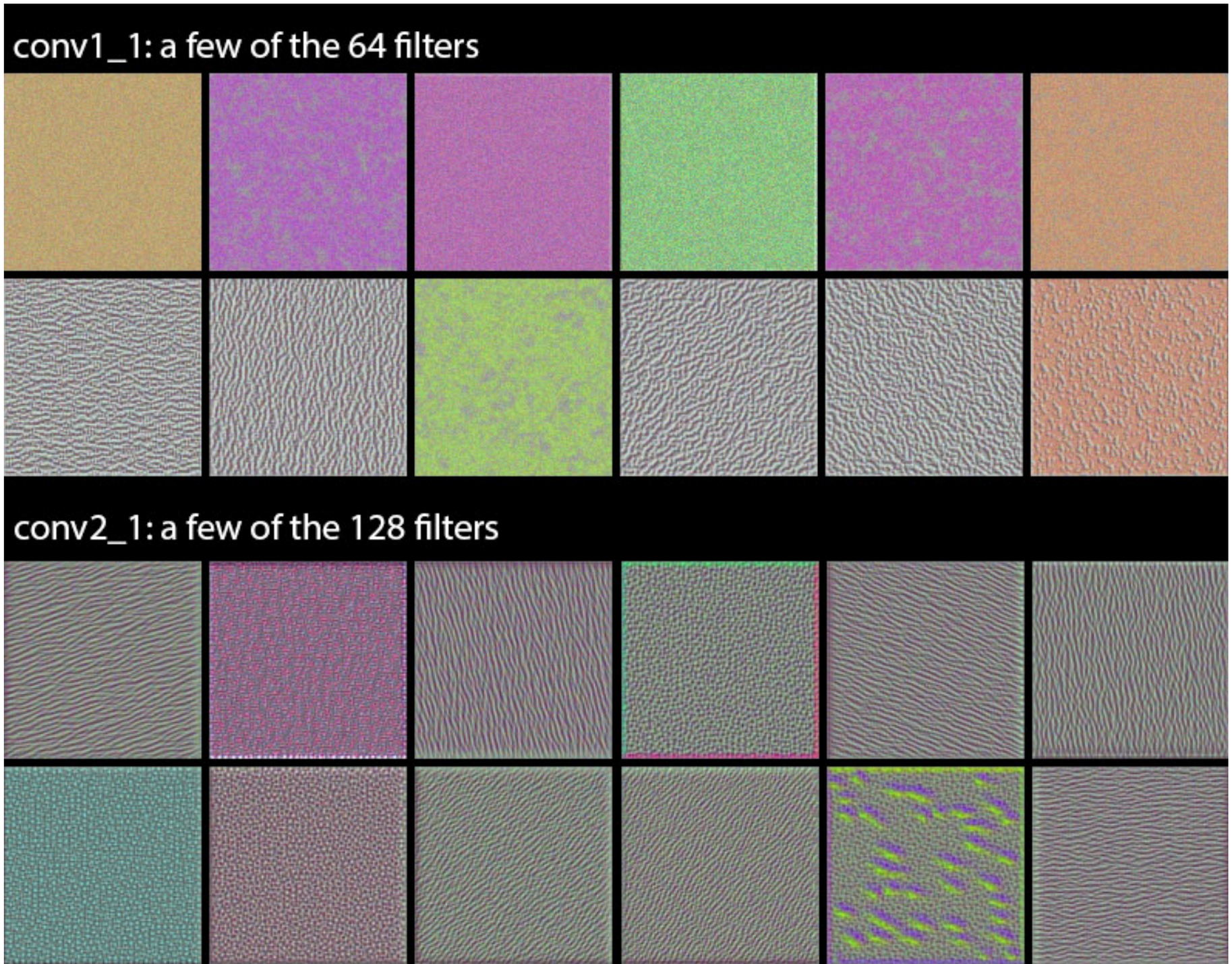
Slide credit: Francois Chollet

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



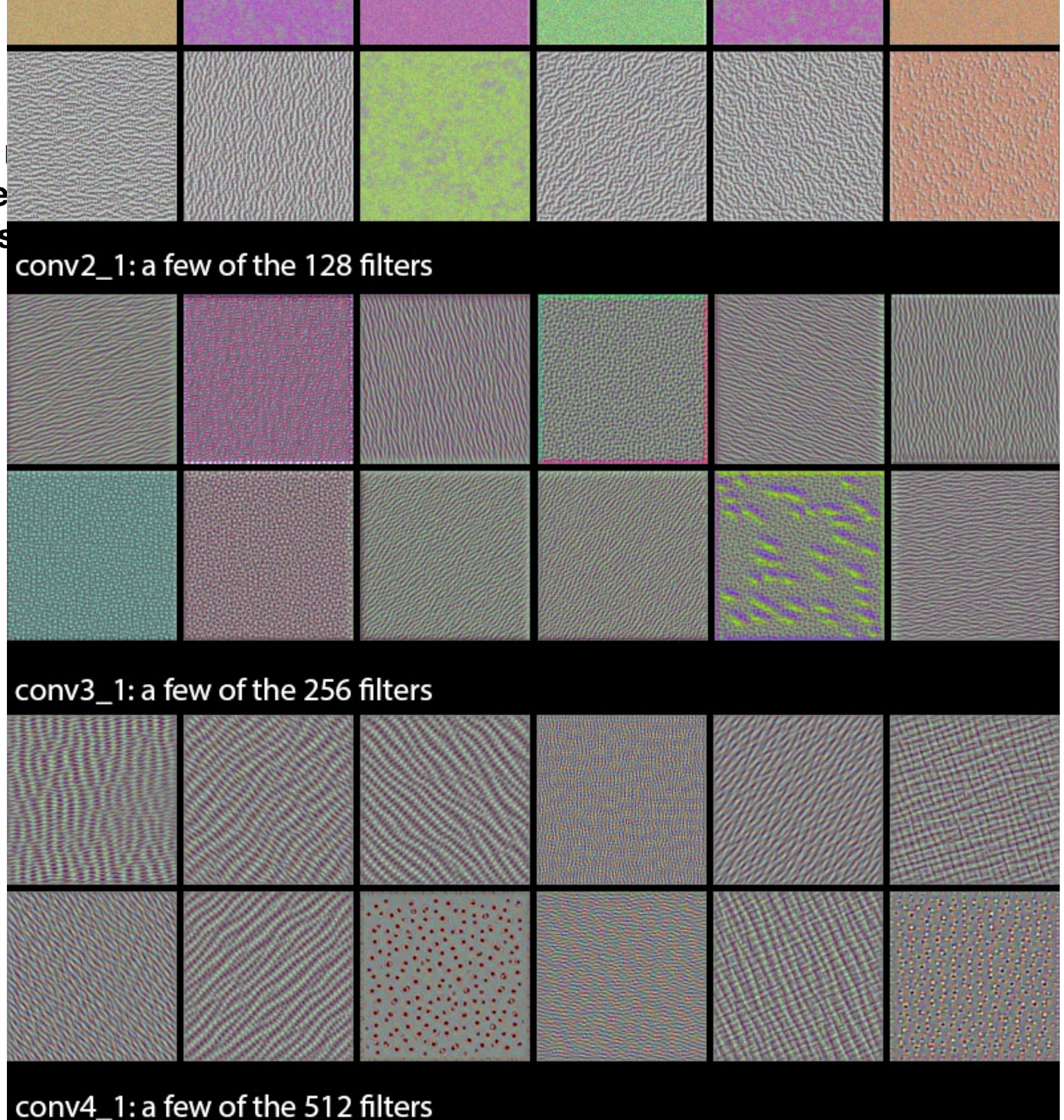
Slide credit: Francois Chollet

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



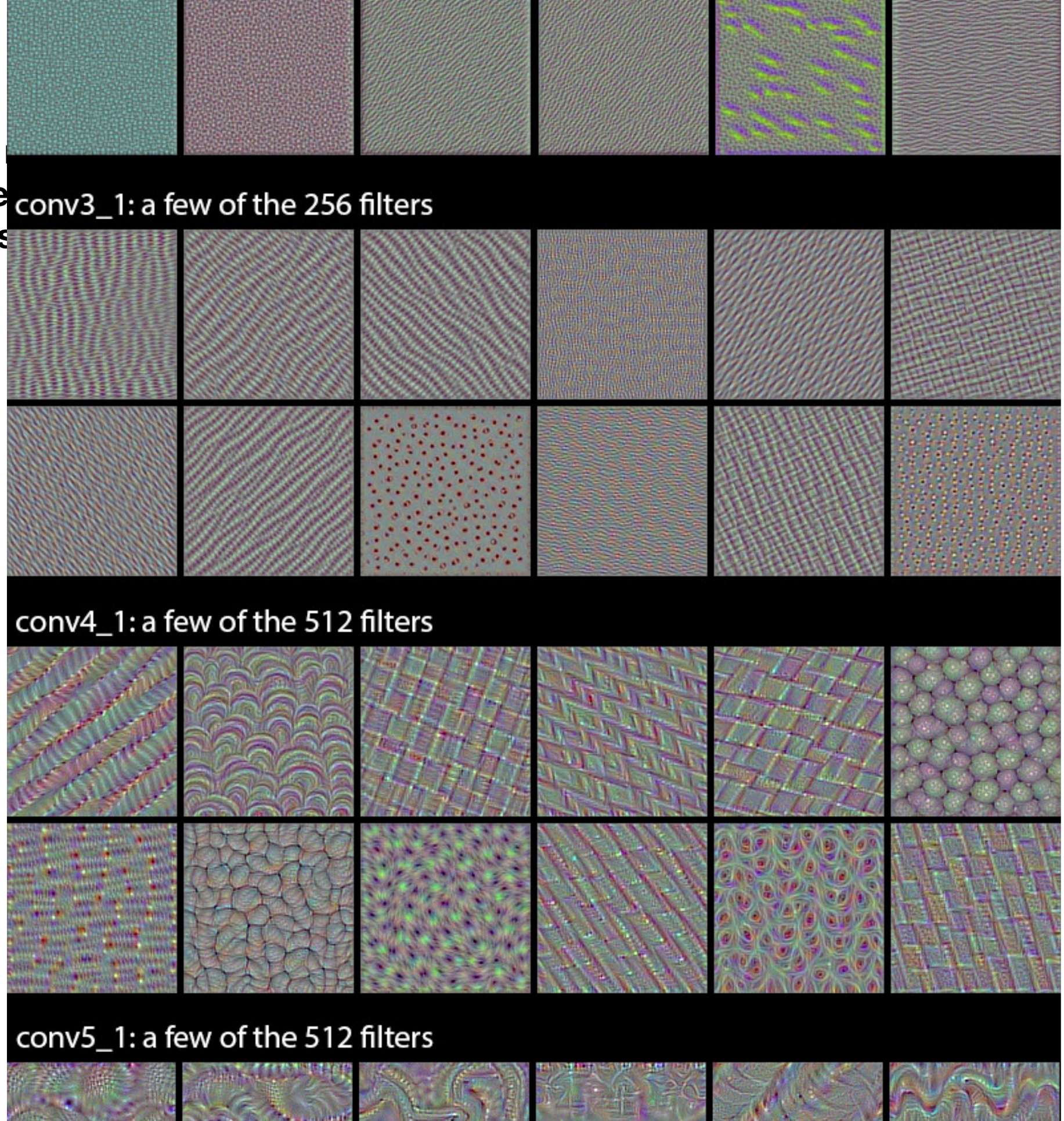
Slide credit: Francois Chollet

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



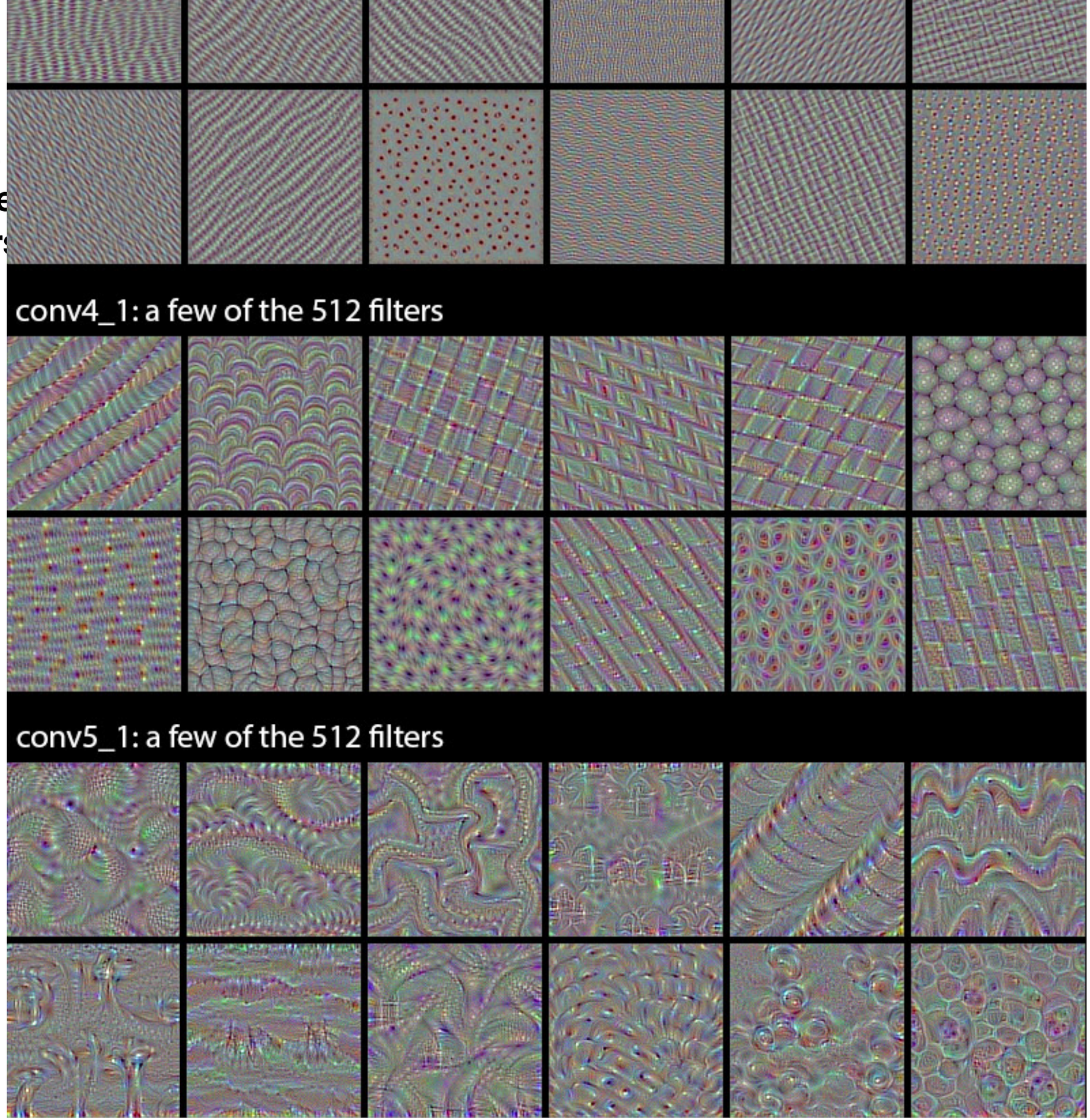
Slide credit: Francois Chollet

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



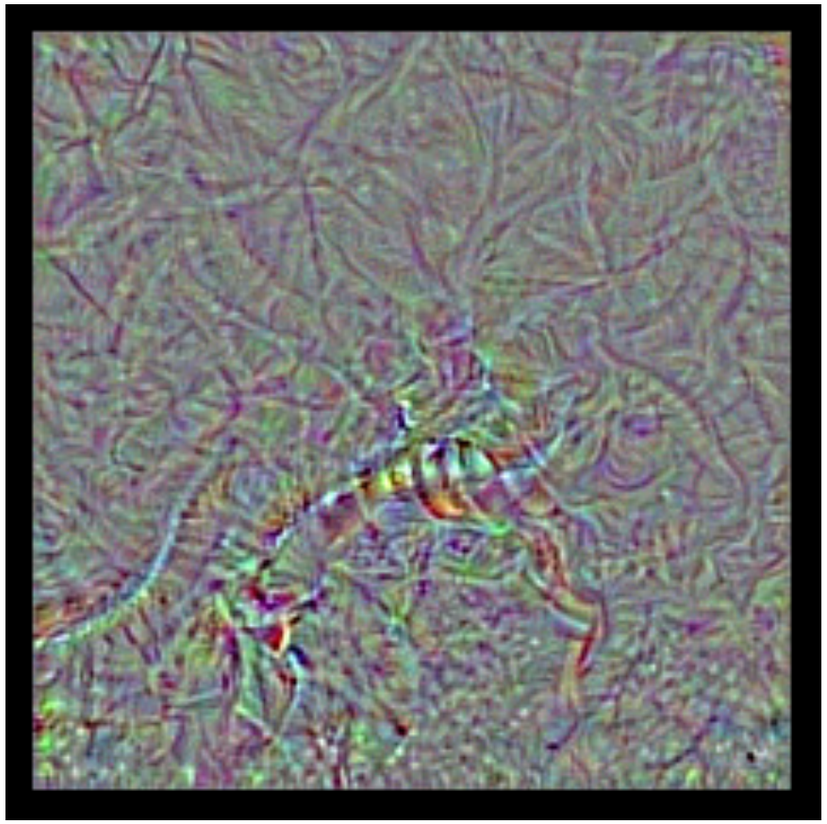
Slide credit: Francois Chollet

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! »



Slide credit: Francois Chollet

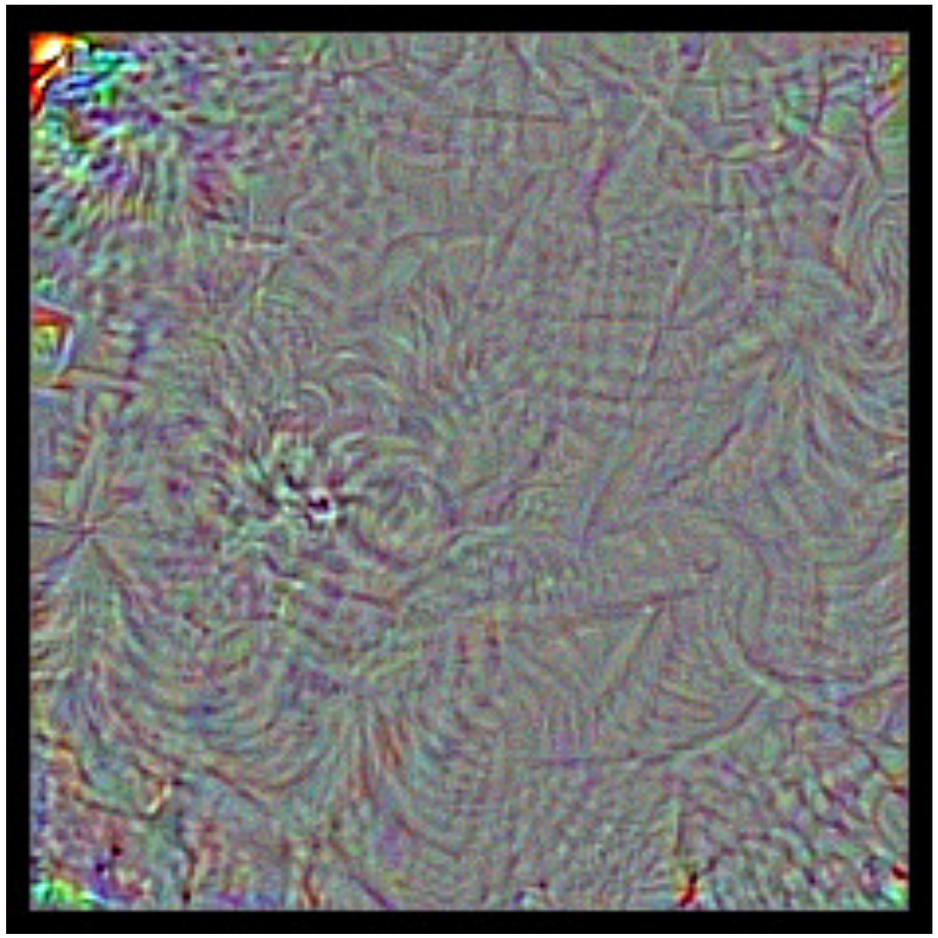
What sort of images maximise the activity for a given neutron in each layers?

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

Let's try with a 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:

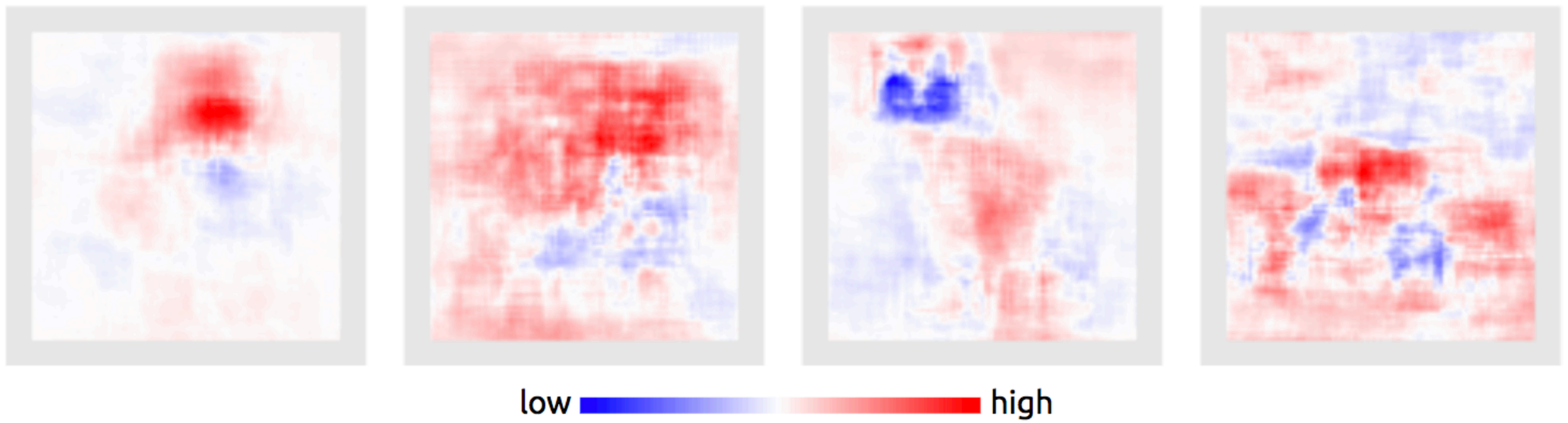
1. the value of the maximally responding output unit on the image, and
2. the value of the same unit with that part occluded.

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

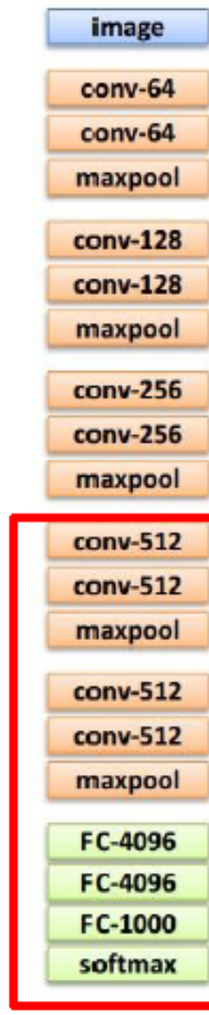
(state of art result in few minutes)

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

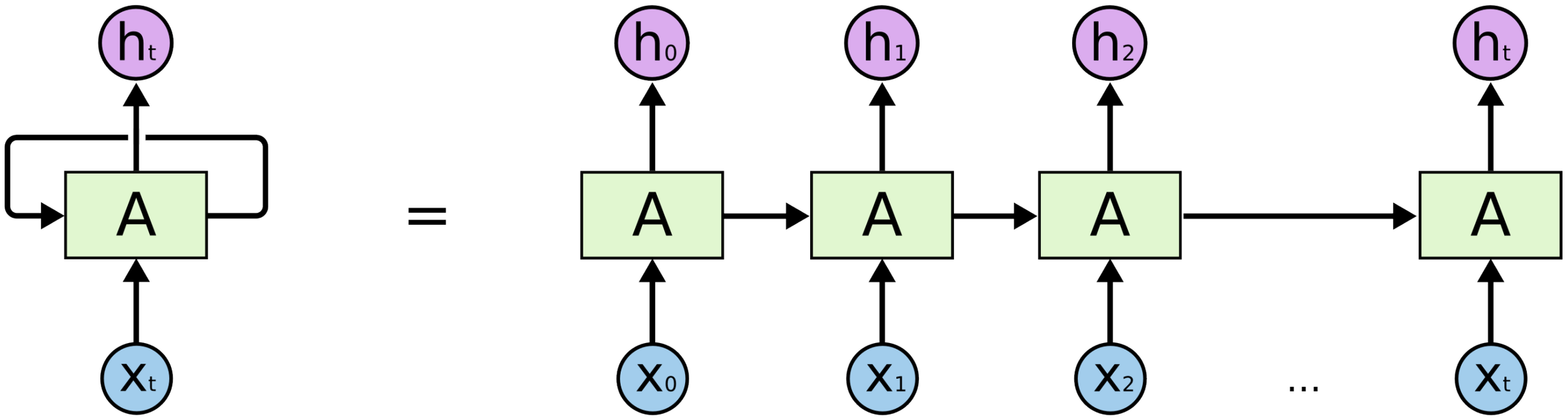
retrain bigger portion of the network, or even all of it.

Achitectures

5: Recurrent Neural Networks

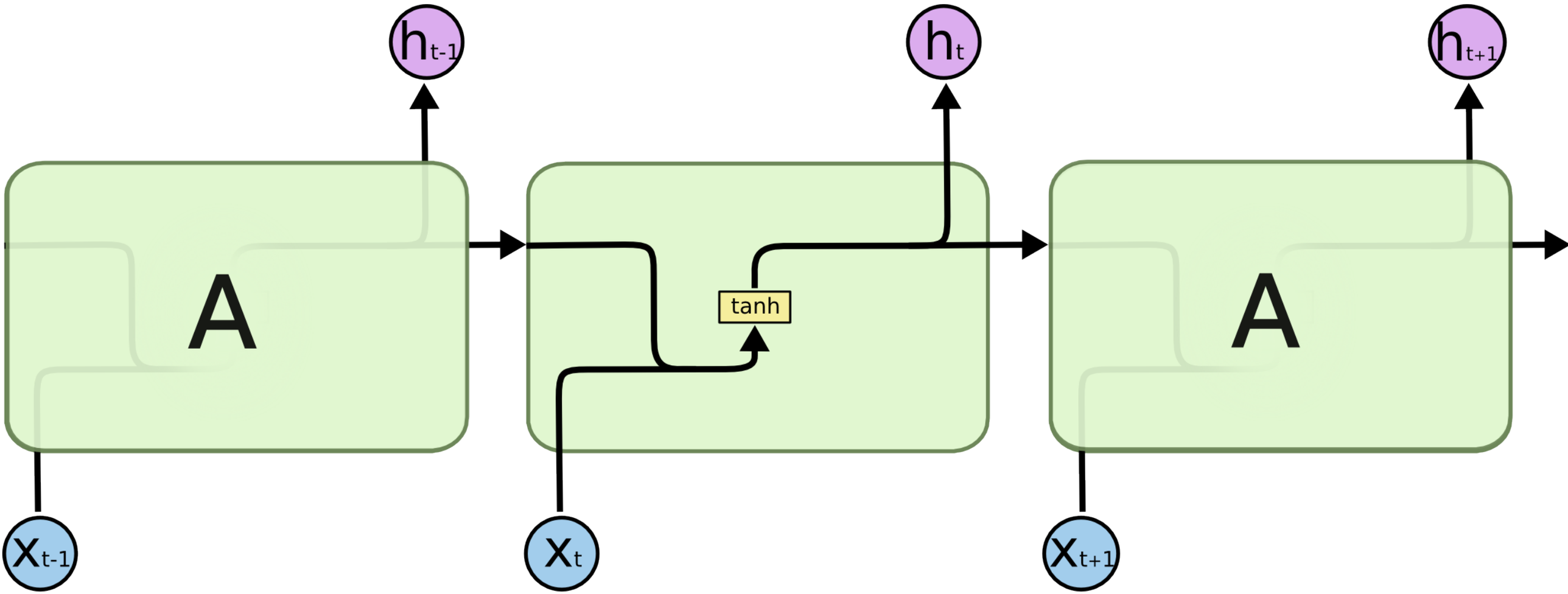
Times series

Recurent Neural Networks



Slide credit: Christopher Olah

Recurent Neural Networks



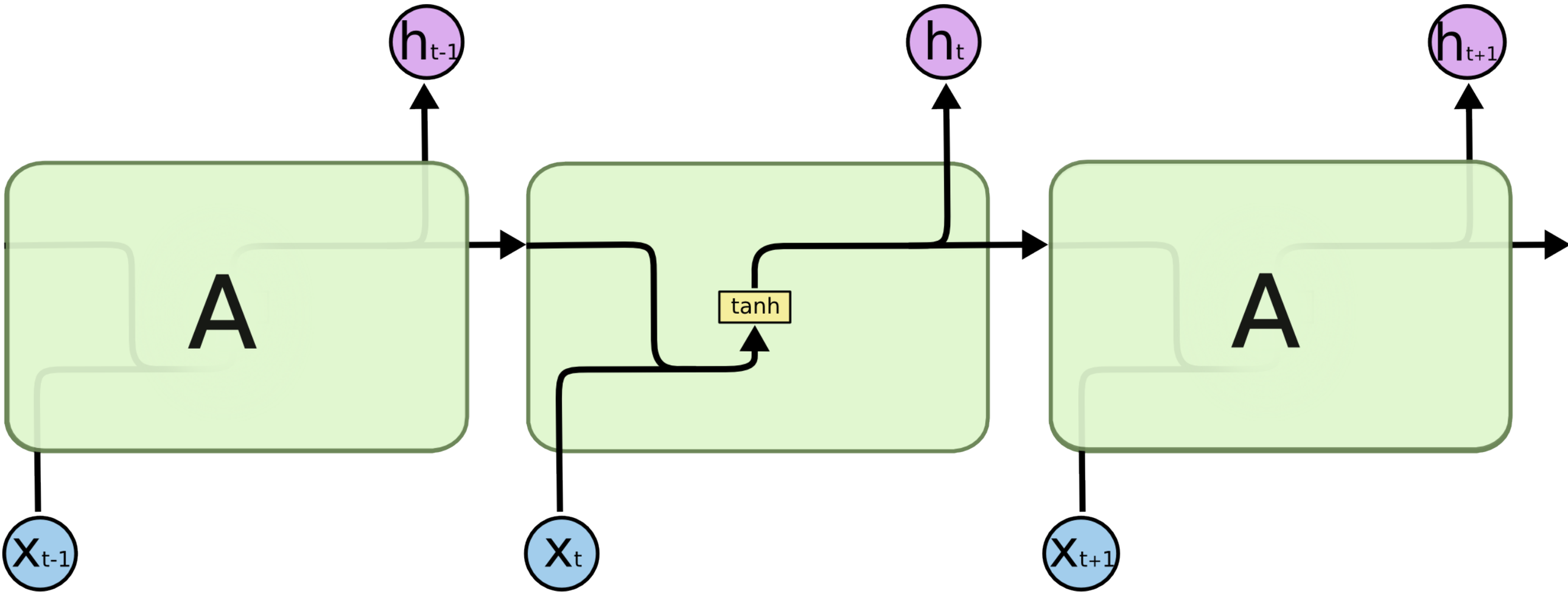
$$h_t = \tanh(Wx_t + Uh_{t-1} + b)$$

Slide credit: Christopher Olah

Echo State Networks

Use random matrices, fit on h!

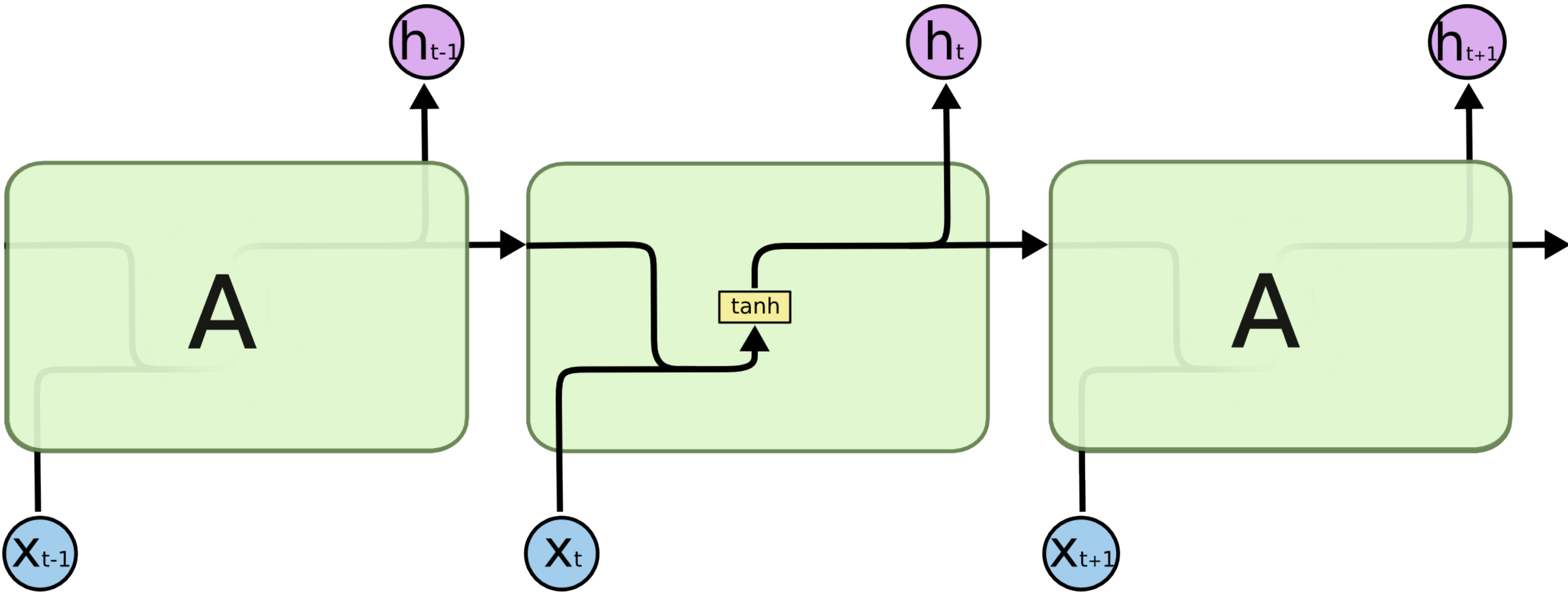
« Reccurent » equivalent to the random projection of lecture II



$$h_t = \tanh(Wx_t + Uh_{t-1} + b)$$

Slide credit: Christopher Olah

Recurent Neural Networks

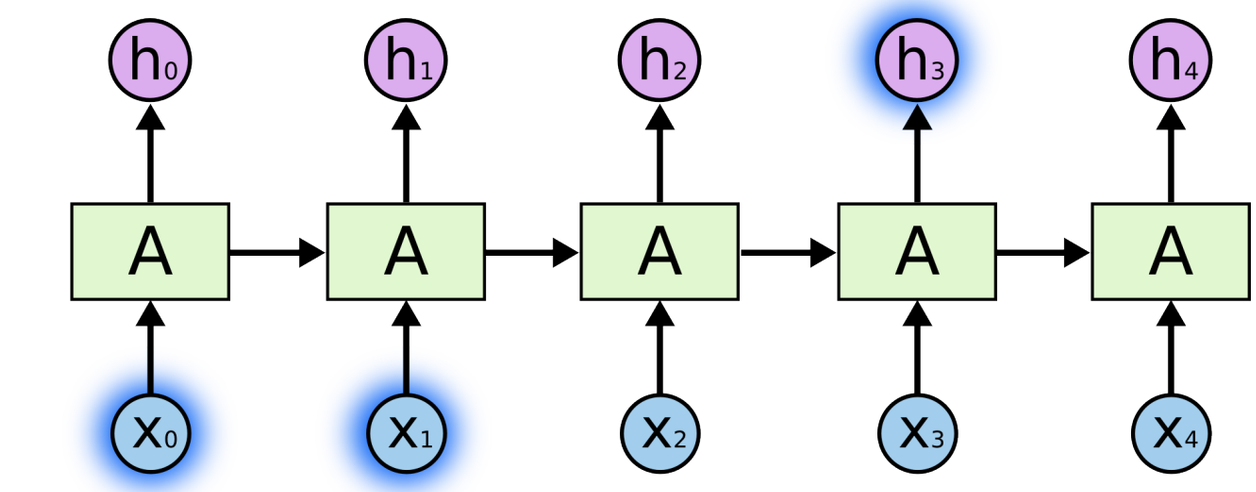


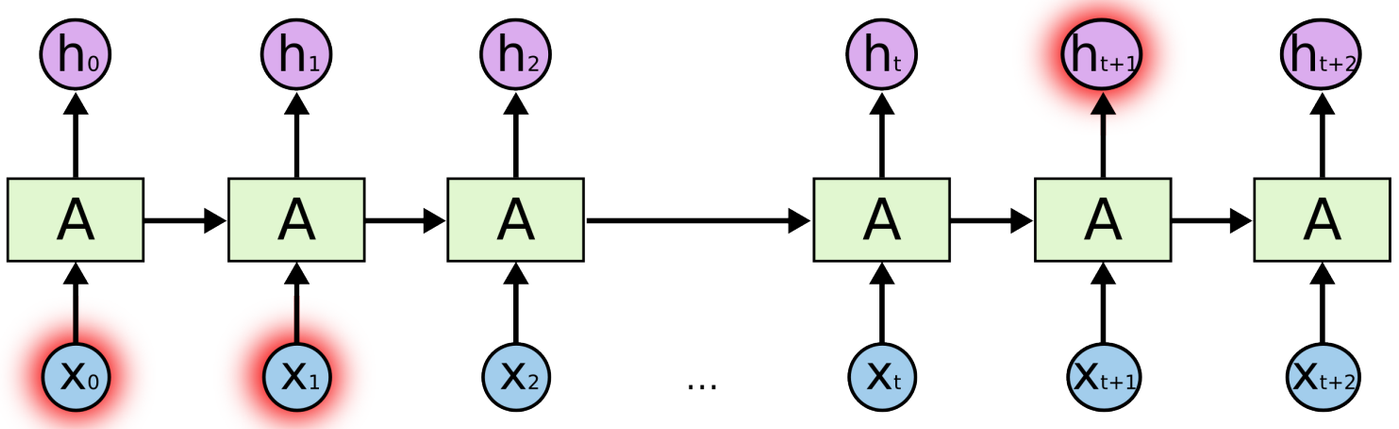
$$h_t = \tanh(Wx_t + Uh_{t-1} + b)$$

Slide credit: Christopher Olah

Recurent Neural Networks

The Problem of Long-Term Dependencies

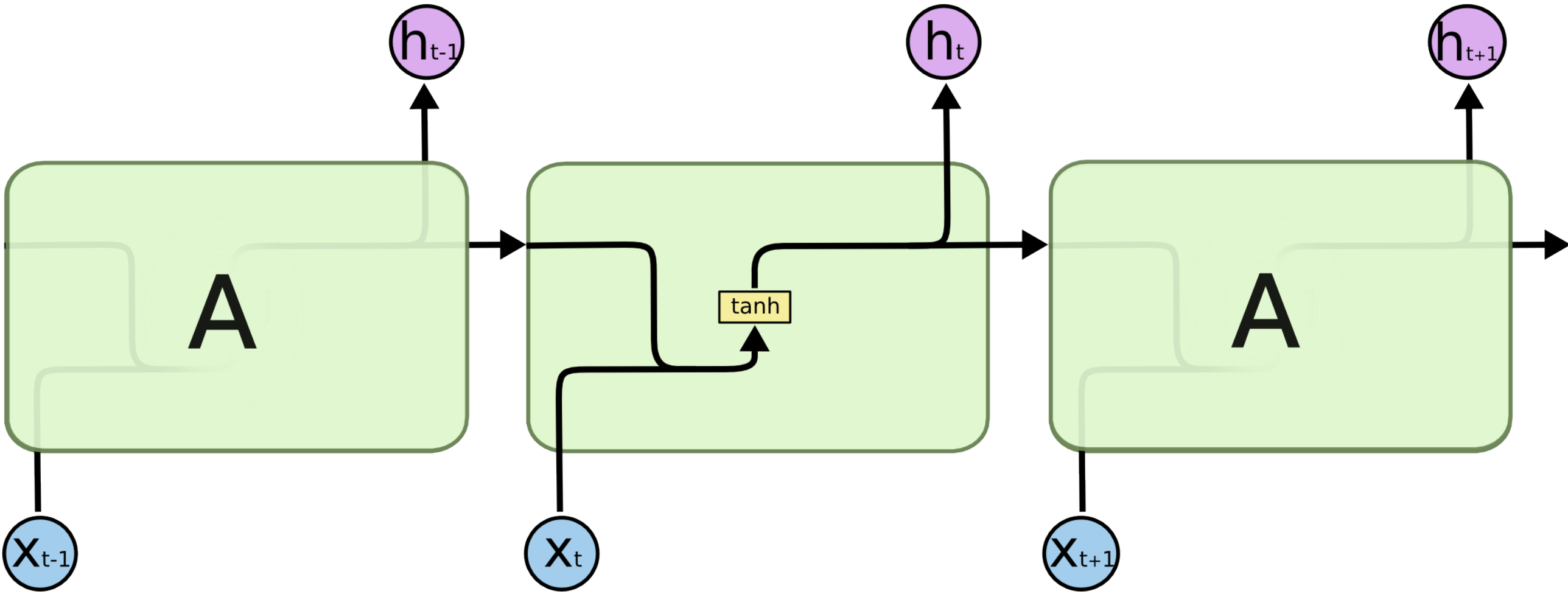




Slide credit: Christopher Olah

LSTMs!

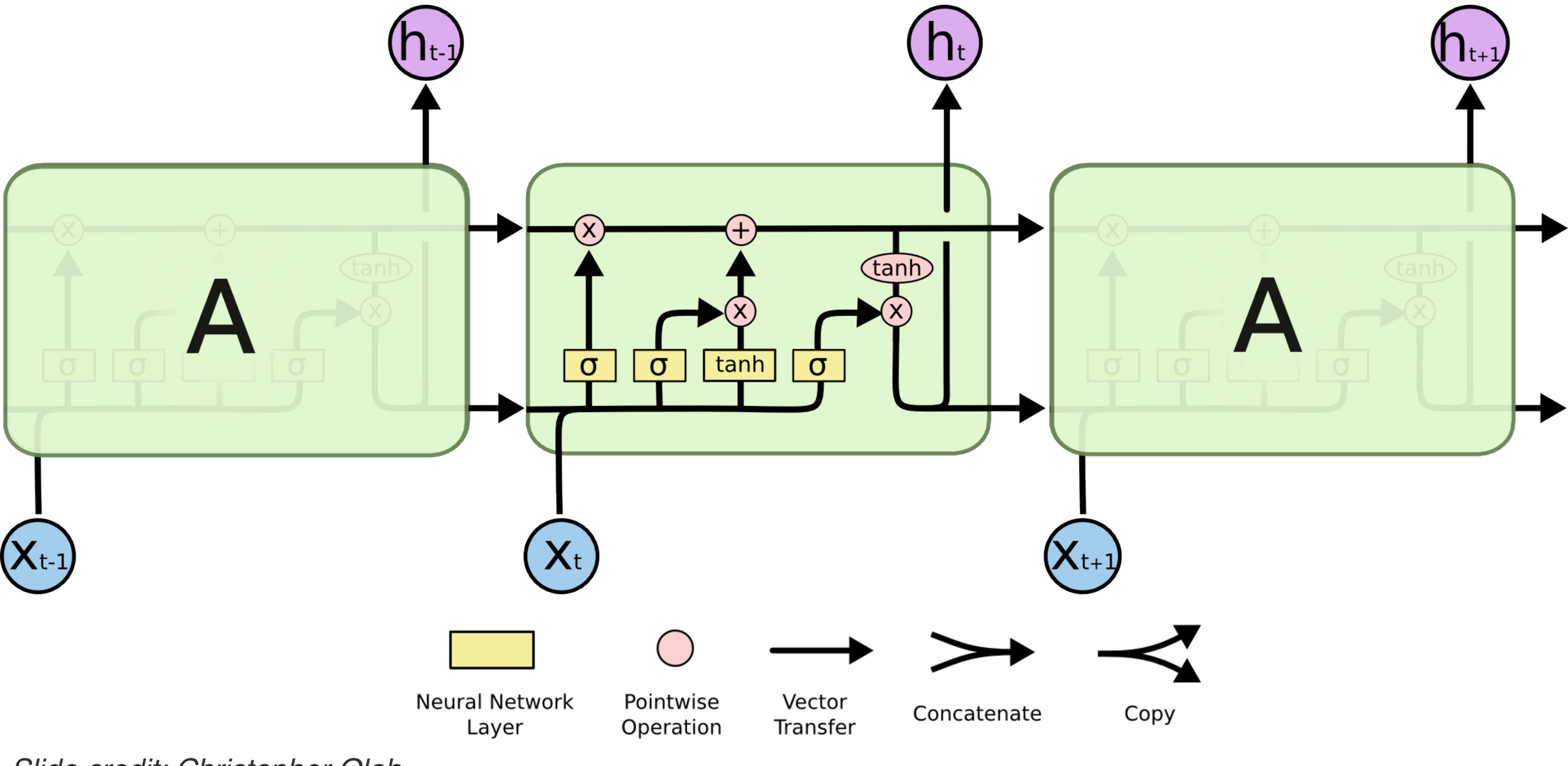
Long Short Term Memory networks



Slide credit: Christopher Olah

LSTMs!

Long Short Term Memory networks

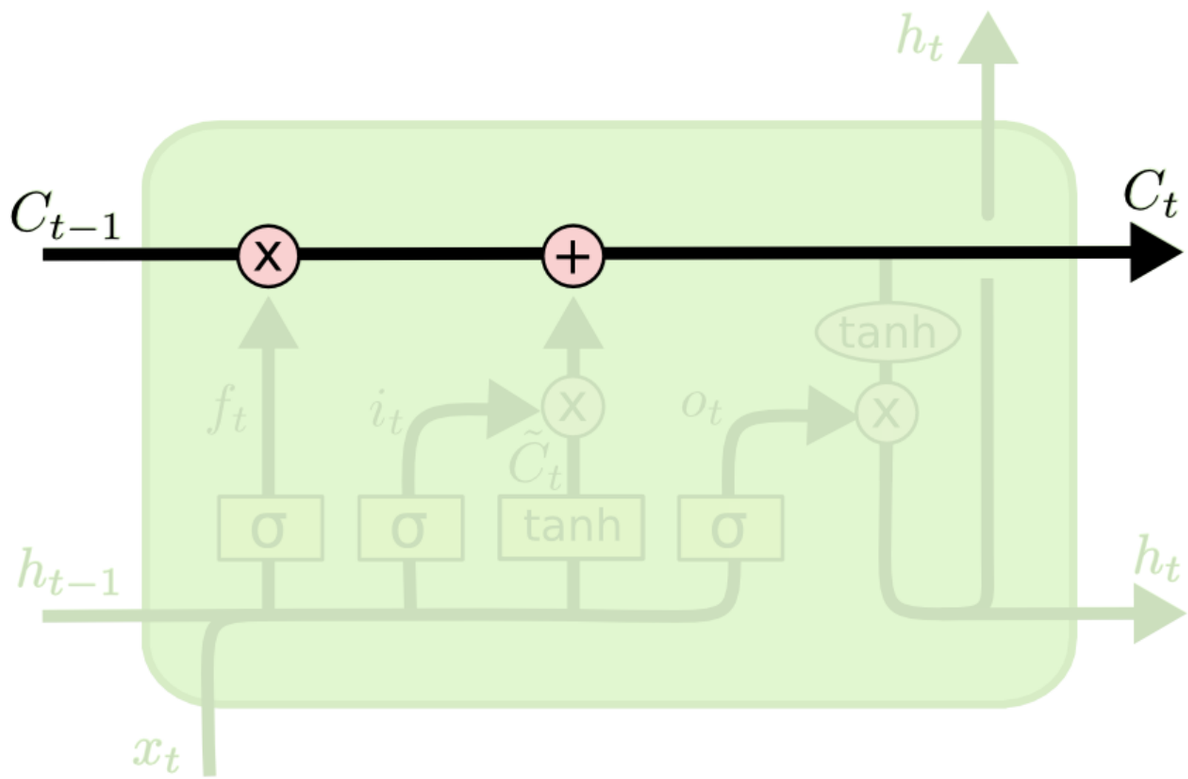


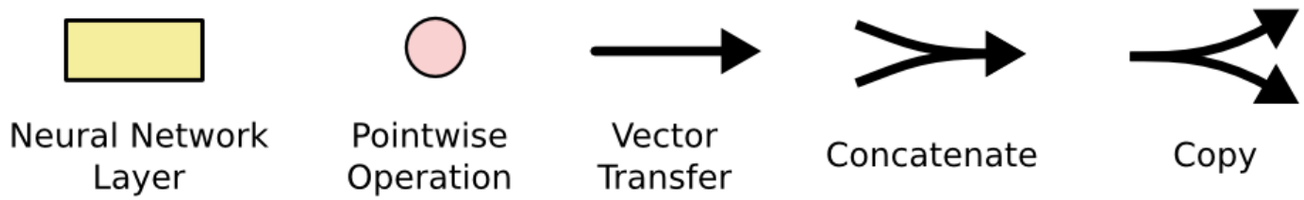
Slide credit: Christopher Olah

LSTMs!

Long Short Term Memory networks

The CELL GATE! The Core Idea Behind LSTMs



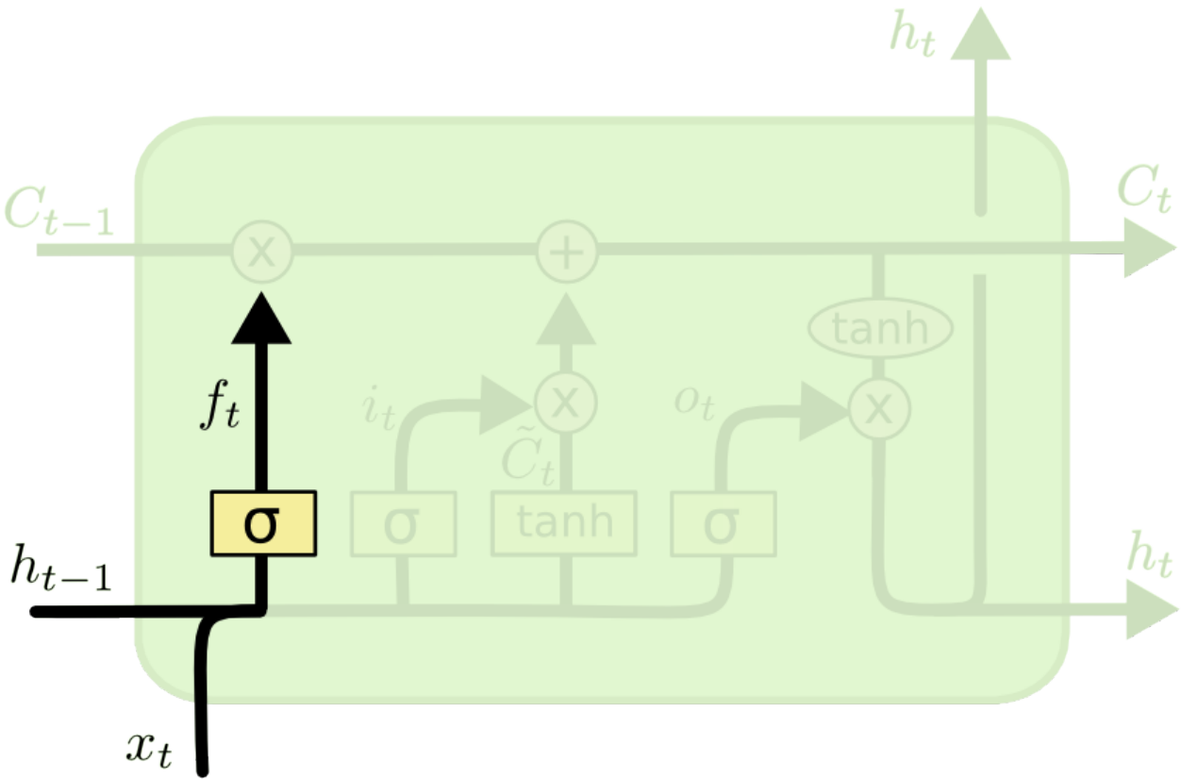


Slide credit: Christopher Olah

LSTMs!

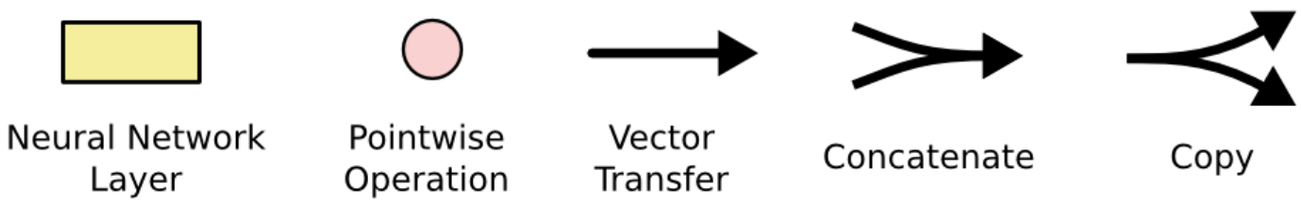
Long Short Term Memory networks

First step: decide what information we're going to throw away from the cell state.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Sigmoid!

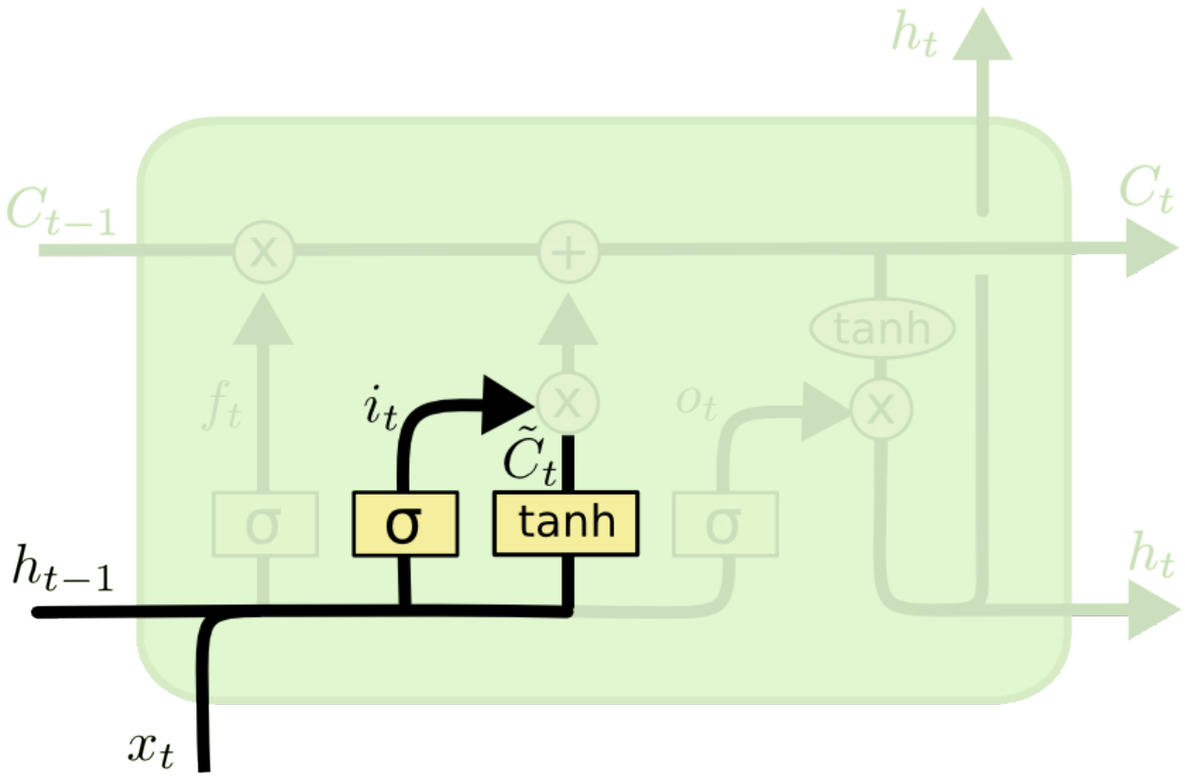


Slide credit: Christopher Olah

LSTMs!

Long Short Term Memory networks

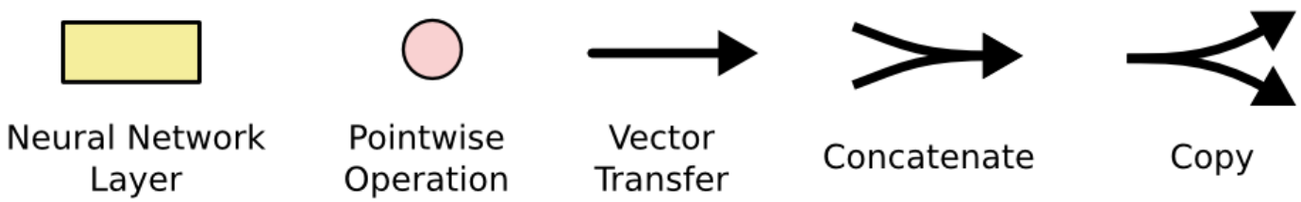
Second step: decide what information we're going to store in the cell state.



Sigmoid!

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C$$

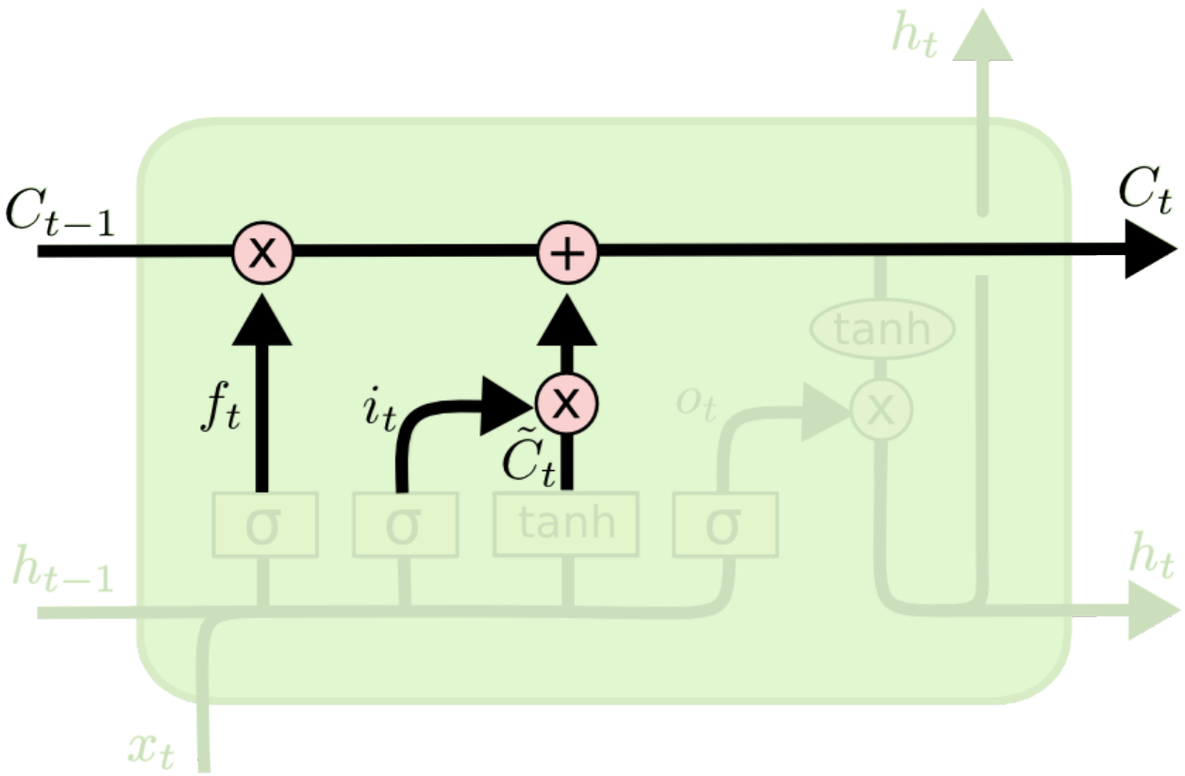


Slide credit: Christopher Olah

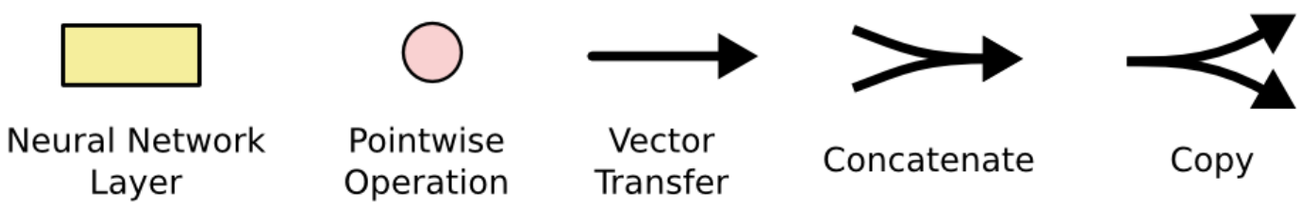
LSTMs!

Long Short Term Memory networks

Merge the last two steps in the cell gate



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

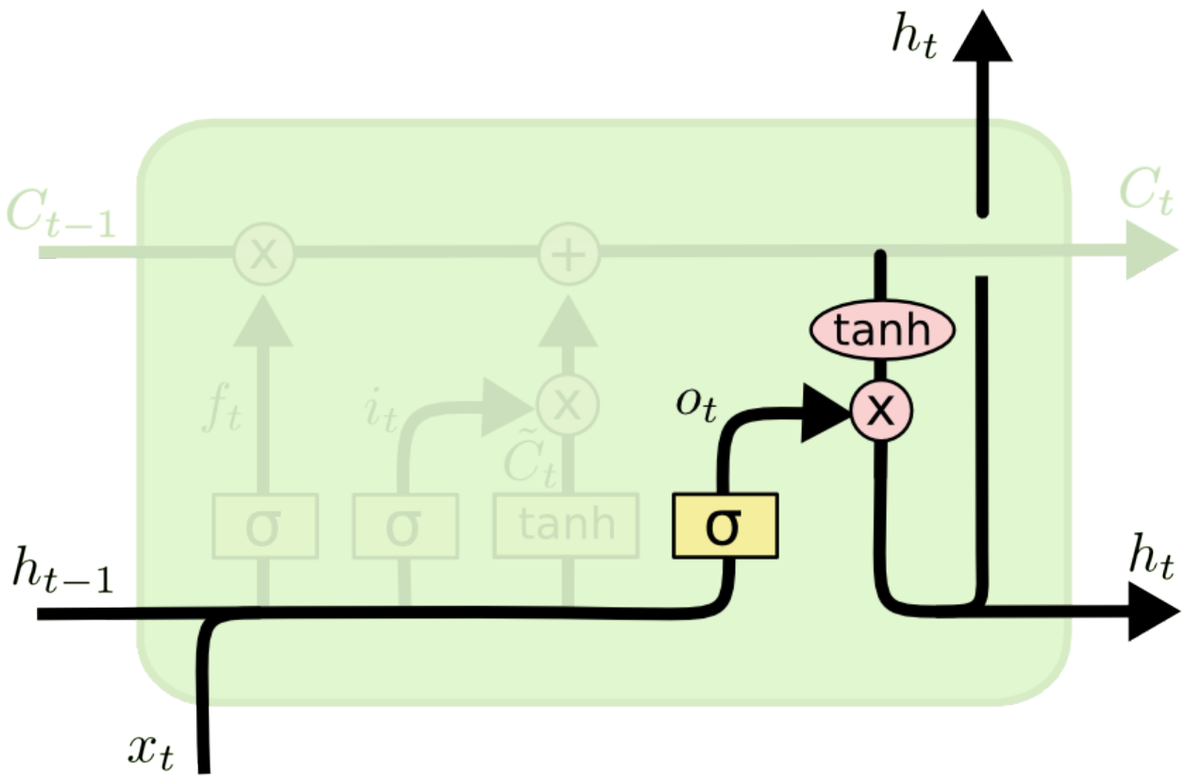


Slide credit: Christopher Olah

LSTMs!

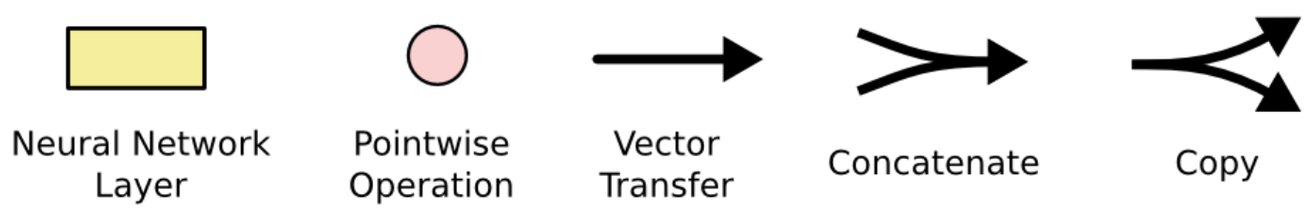
Long Short Term Memory networks

Finally, a more standard output (weighted by the cell gate!)



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$



Slide credit: Christopher Olah

LSTMs!

Long Short Term Memory networks

RNN Bible
@RNN_Bible

Suivre

3:11 The LORD is my strength and glory, strifes of corrupt things: and thou wilt be thy supplications.

LSTMs!

Long Short Term Memory networks

SEED: Jobs

Good afternoon. God bless you.

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done. The promise of the men and women who were still going to take out the fact that the American people have fought to make sure that they have to be able to protect our part. It was a chance to stand together to completely look for the commitment to borrow from the American people. And the fact is the men and women in uniform and the millions of our country with the law system that we should be a strong stretcks of the forces that we can afford to increase our spirit of the American people and the leadership of our country who are on the Internet of American lives.

Thank you very much. God bless you, and God bless the United States of America.

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

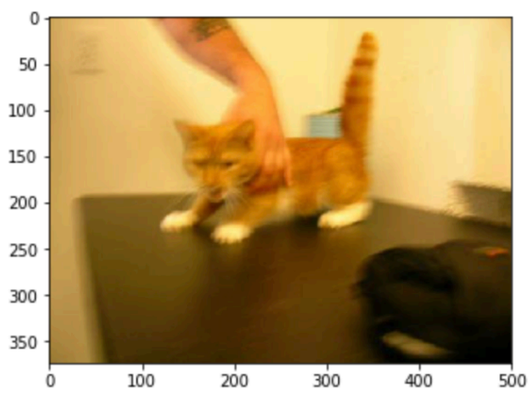
We are going to use the data from the Kaggle contest. The dataset is quite big: there are 25000 training samples, and 12500 test samples, all of different sizes. We are thus going to work with only a few of them: we are taking 3000 samples from the training set, and using 2000 of them for training and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

l]:
```
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 *Netflix 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()
```