ANNs

Artificial Neural Networks

Part 2

Agenda

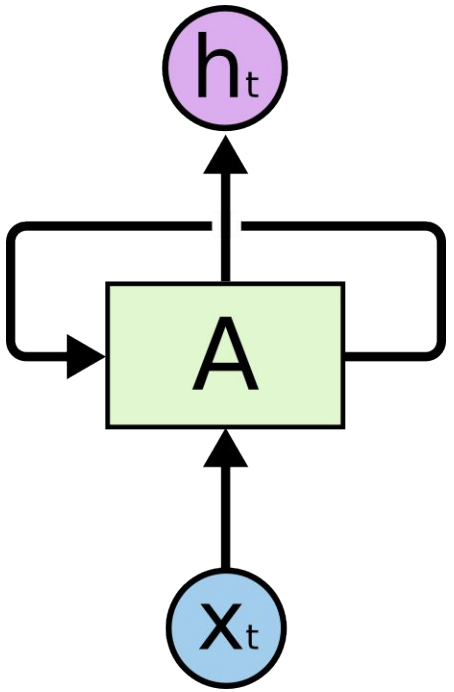
- RNNs
  - LSTM
- CNNs
  - Image Classification
- GANs
  - Motivation

RNNs

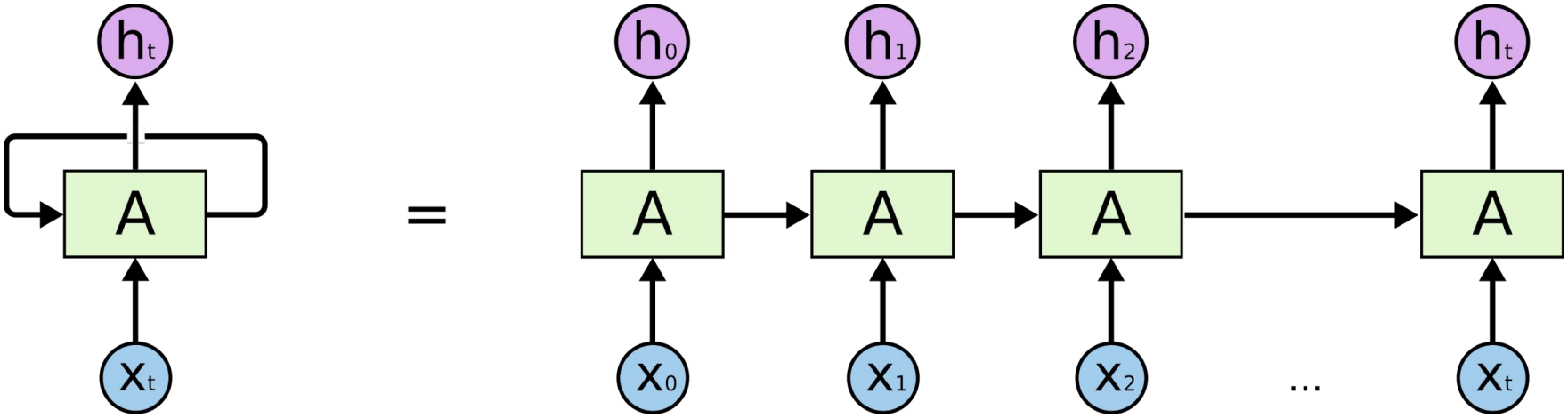
- Humans don’t start their thinking from scratch every second. Your thoughts have persistence.
- Traditional neural networks can’t do this, and it seems like a major shortcoming.
- **Recurrent neural networks** address this issue. They are networks with loops in them, allowing information to persist.

RNNs Have Loops

- In this diagram, a chunk of neural network, A, looks at some input X_t and outputs a value h_t
- A loop allows information to be passed from one step of the network to the next.
- These loops make recurrent neural networks seem kind of mysterious.



Unrolling the RNN



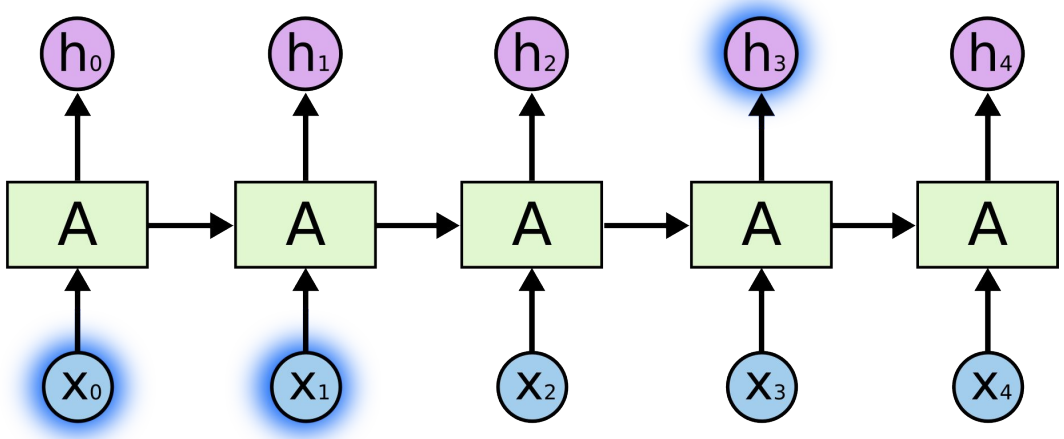
- It turns out that they aren't all that different than a normal neural network.
- A recurrent neural network can be thought of as **multiple copies** of the same network, each passing a message to a successor.
- This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists.
- They're the natural architecture of neural network to use for such data.

Long Short Term Memory Networks (LSTMs)

- In the last few years, there have been incredible successes applying RNNs to a variety of problems:
  - speech recognition, language modeling, translation, image captioning
  - The Unreasonable Effectiveness of RNNshttp://karpathy.github.io/2015/05/21/rnn-effectiveness/
- Essential to these successes is the use of “LSTMs,” a very special kind of recurrent neural network which works, for many tasks, much better than the standard version.
- Almost all exciting results based on recurrent neural networks are achieved with them.

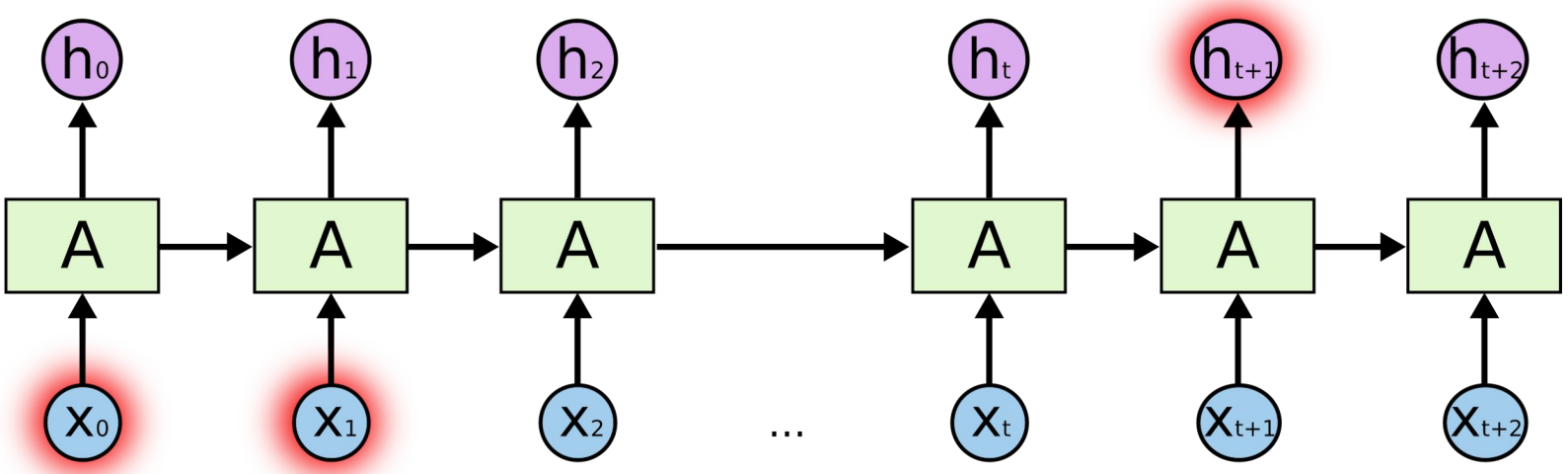
RNNs and the Problem of Long-Term Dependencies

- One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task, such as using previous video frames might inform the understanding of the present frame.
- Sometimes, we only need to look at recent information to perform the present task.
- In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.
- For example, predicting "sky" in "the clouds are in the …" is a **short-term dependency.**



RNNs and the Problem of Long-Term Dependencies

- But there are also cases where we need more context.
- Consider trying to predict the last word in the text “I grew up in France… I speak fluent ….”
- Recent information suggests that the next word is probably the name of a language (French), but if we want to narrow down which language, we need the context of France, from further back.
- It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large. This is a **long-term dependency.**



RNNs and the Problem of Long-Term Dependencies

- Unfortunately, as that dependency gap grows, RNNs become unable to learn to connect the information.
- In theory, RNNs are absolutely capable of handling such "long-term dependencies."
  - A human could carefully pick parameters for them to solve toy problems of this form.
- Sadly, in practice, RNNs don't seem to be able to learn them.
  - The problem was explored in depth by ***Hochreiter (1991) [German] and Bengio, et al. (1994)***, who found some pretty fundamental reasons why it might be difficult.
  - Learning Longer-term Dependencies in RNNs with Auxiliary Losses (2018)
    - https://arxiv.org/pdf/1803.00144.pdf
- Thankfully, LSTMs don't have this problem!

RNN Disadvantages

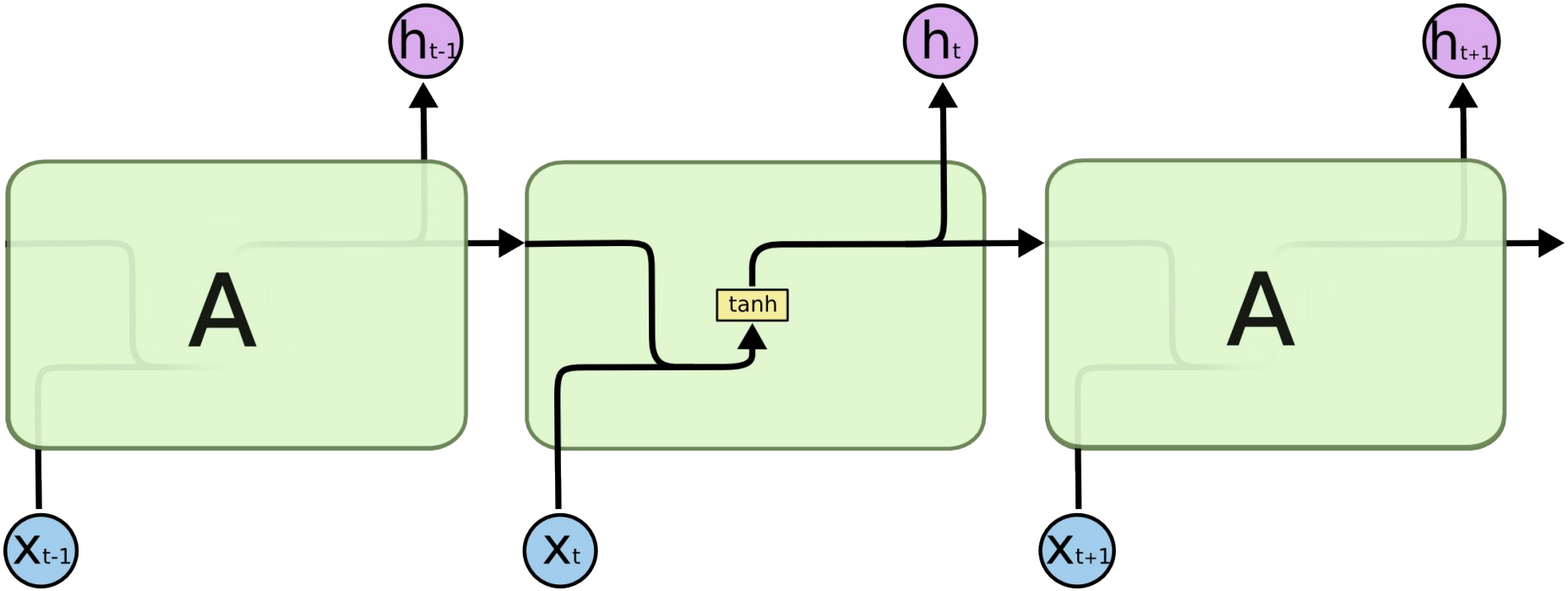
- Training RNNs
- The vanishing or exploding gradient problem
- RNNs cannot be stacked up
- Slow and Complex training procedures
- Difficult to process longer sequences

LSTMs

- **Long Short Term Memory** networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.
- They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.
- **LSTMs are explicitly designed to avoid the long-term dependency problem**. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

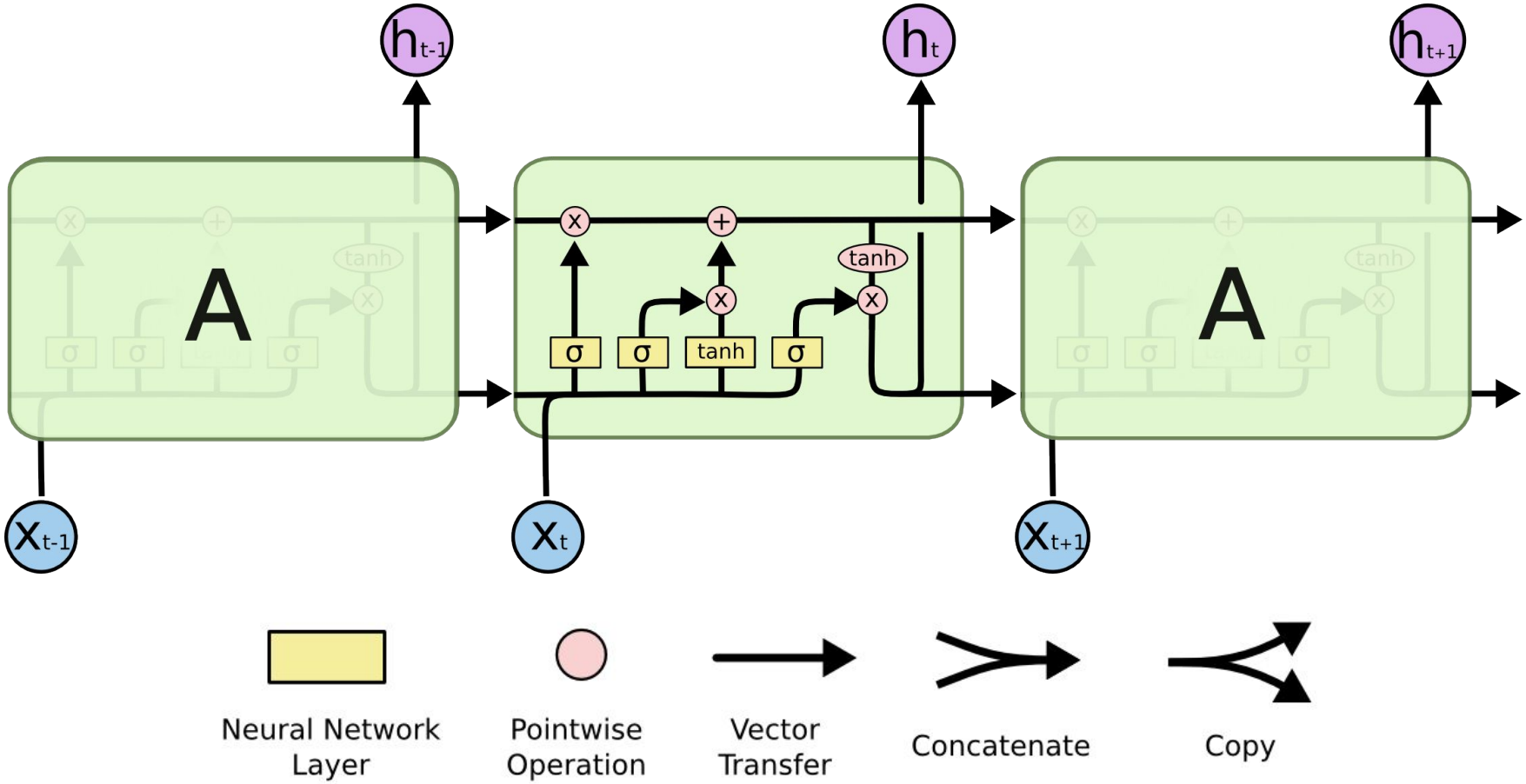
Standard RNNs Repeating Module

- All recurrent neural networks have the form of a chain of repeating modules of neural network.
- The repeating module in a standard RNN contains a single layer.
- This repeating module will have a very simple structure, such as a single tanh layer.



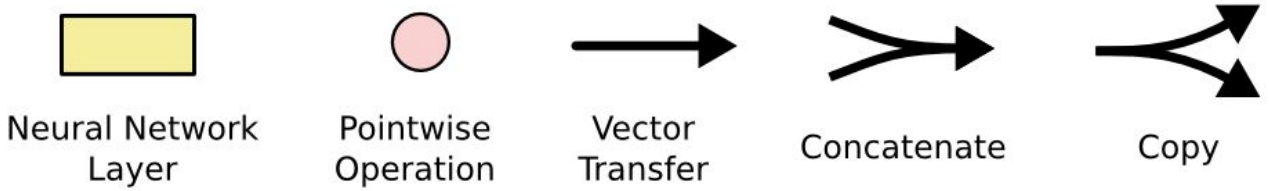
LSTM Repeating Module

- The repeating module in an LSTM contains four interacting layers, interacting in a very special way.



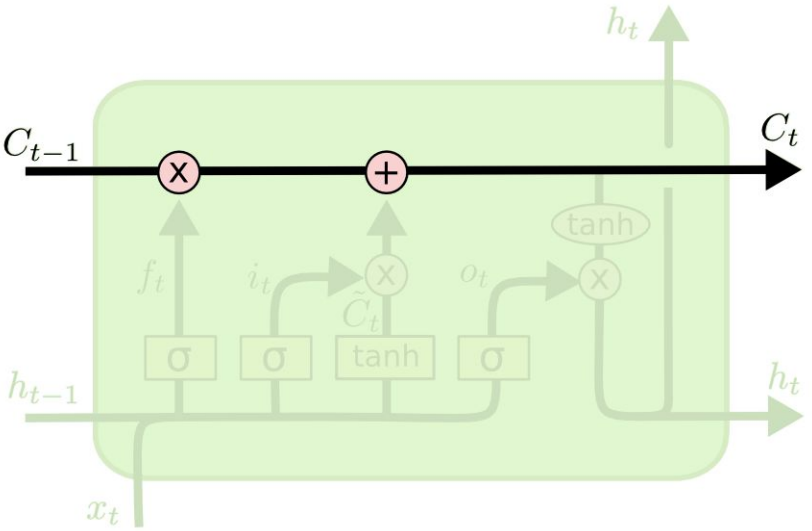
Notation

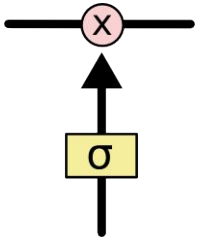
- Each line carries an entire vector, from the output of one node to the inputs of others.
- The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers.
- Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations.



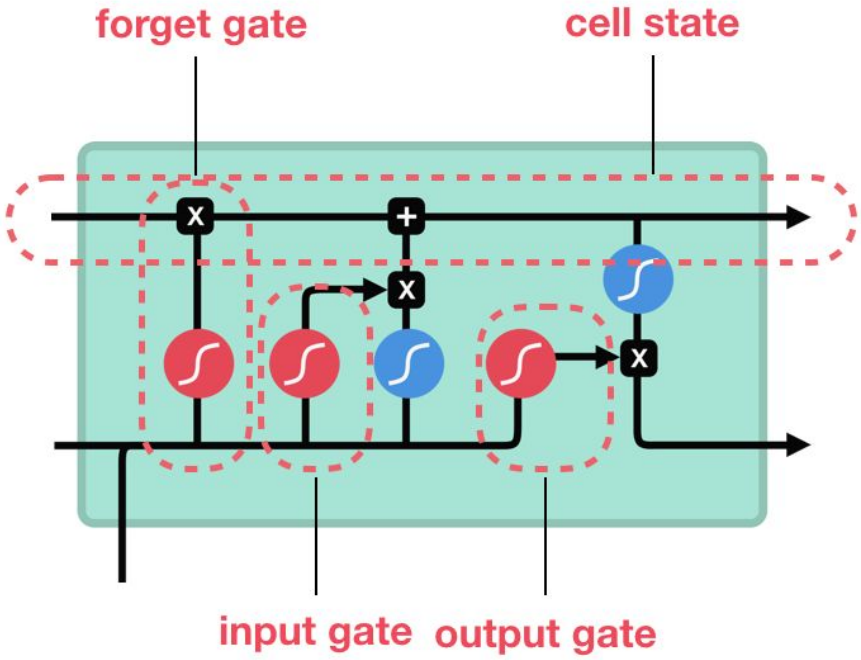
LSTM Core Idea

- The key to LSTMs is the **cell state**, the horizontal line running through the top of the diagram.
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- **An LSTM has three of these gates**, to protect and control the cell state.
- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.





LSTM Gates

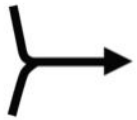


sigmoid

tanh

pointwise multiplication

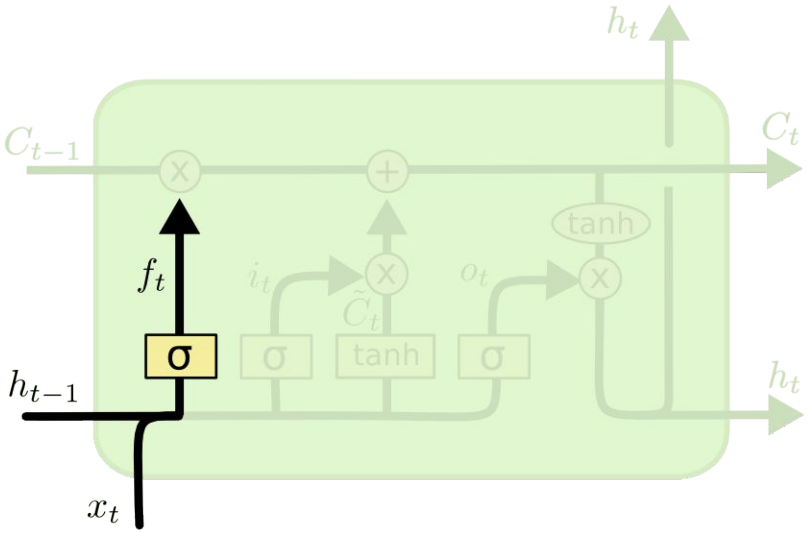
pointwise addition



vector concatenation

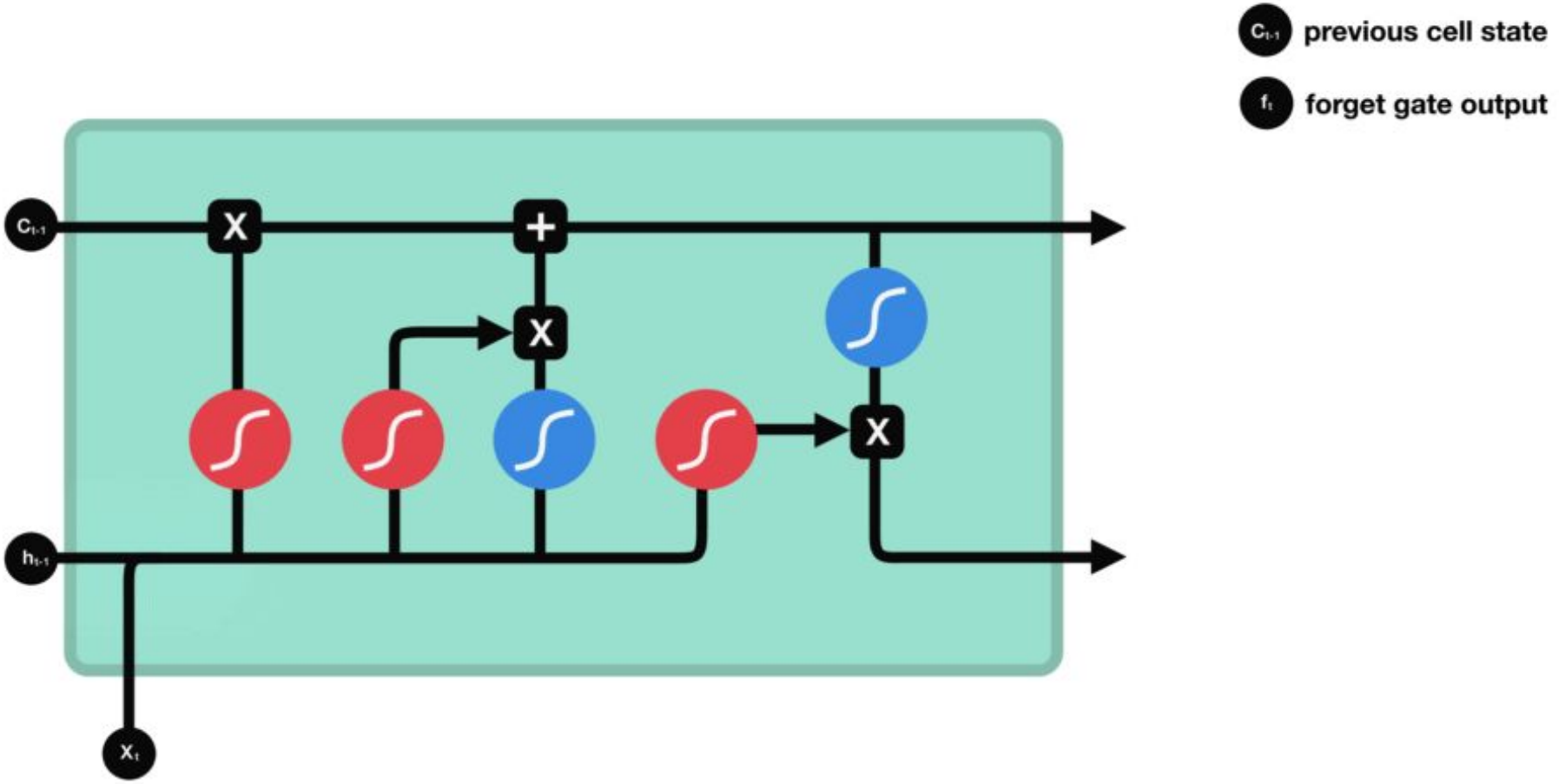
Forgetting

- The first step in our LSTM is to decide what information we're going to throw away from the cell state.
- This decision is made by a sigmoid layer called the "**forget gate layer**."
- It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.
- A 1 represents "completely keep this" while a 0 represents "completely get rid of this."



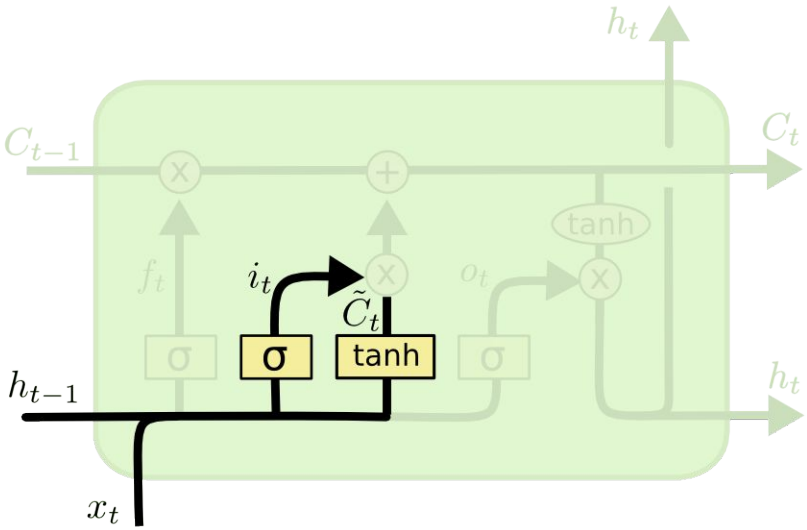
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Forget Gate



Remembering

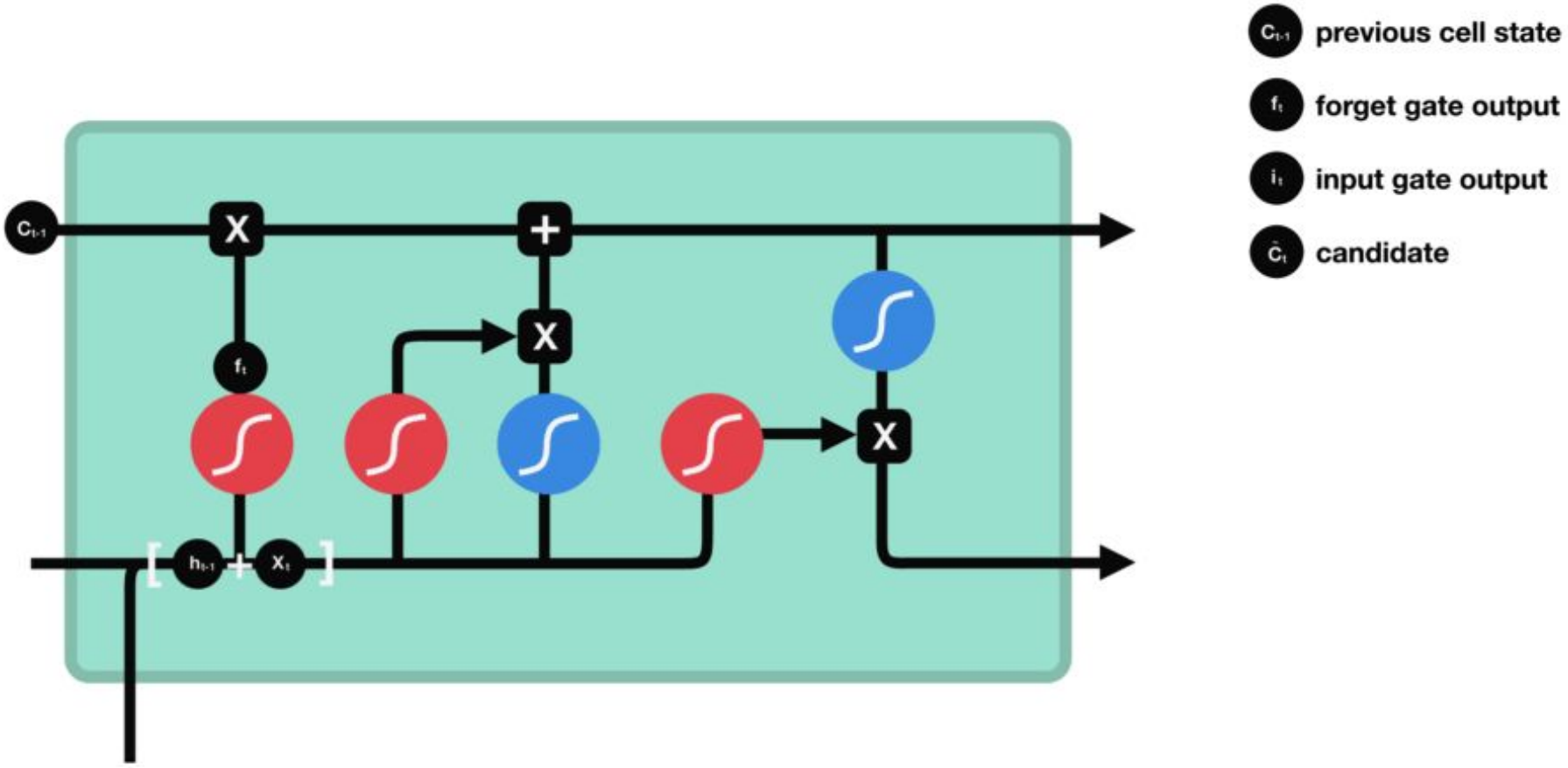
- The next step is to decide what new information we're going to store in the cell state. This has two parts.
- First, a sigmoid layer called the "**input gate layer**" decides which values we'll update.
- Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.
- In the next step, we'll combine these two to create an update to the state.



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

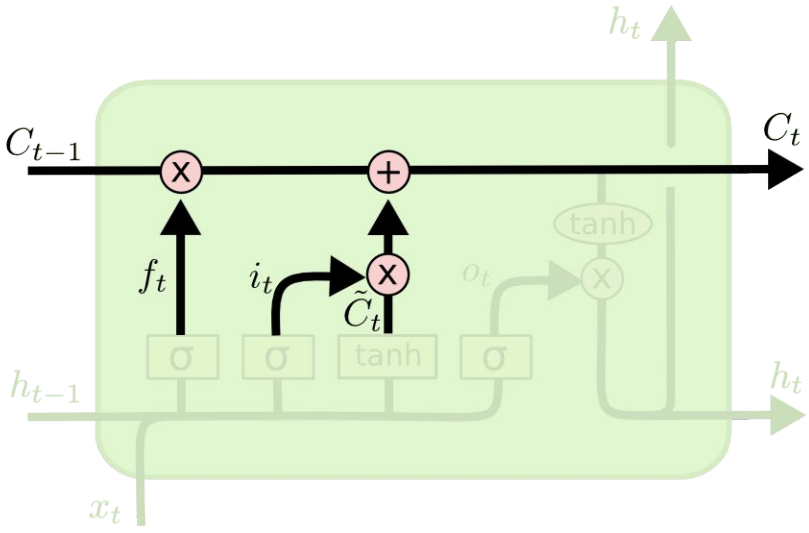
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Input Gate


C_{t-1}
X
+
f_t
X
$[h_{t-1}, x_t]$
X
C_{t-1} previous cell state
f_t forget gate output
i_t input gate output
$\tilde{C}_t$ candidate

Updating

- It's now time to update the old cell state, C_{t-1}, into the new cell state C_t.
- The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by f_t, forgetting the things we decided to forget earlier.
- Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

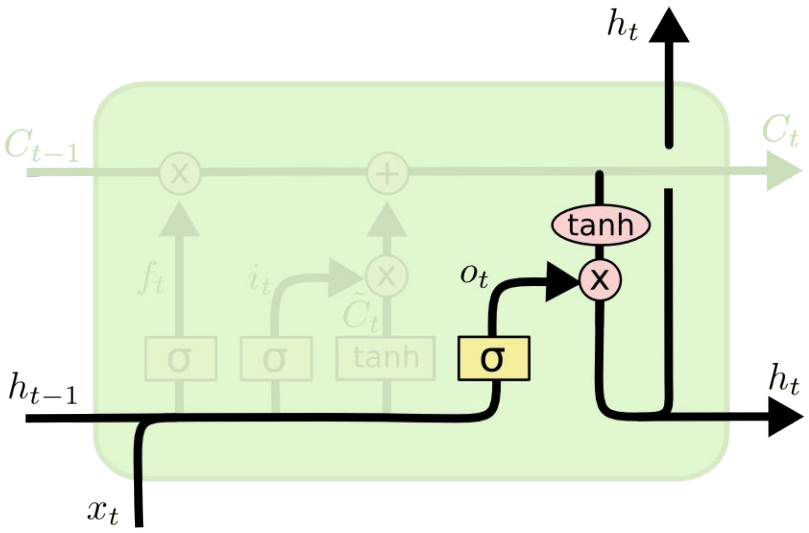


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Calculating Cell State

Outputting

- Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version.
- First, we run a sigmoid layer which decides what parts of the cell state we're going to output.
- Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



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

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

Output Gate

LSTM Variants

- One popular LSTM variant, introduced by Gers & Schmidhuber (2000), is adding “peephole connections.” This means that we let the gate layers look at the cell state.
- Another variation is to use coupled forget and input gates. Instead of separately deciding what to forget and what we should add new information to, we make those decisions together. We only forget when we’re going to input something in its place. We only input new values to the state when we forget something older.
- A slightly more dramatic variation on the LSTM is the **Gated Recurrent Unit, or GRU**, introduced by Cho, et al. (2014). It combines the forget and input gates into a single “update gate.” It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.

LSTM Variants

- Greff, et al. (2015) do a nice comparison of popular variants, finding that they're all about the same.
- Jozefowicz, et al. (2015) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.
- LSTMs and GRUs
  - https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

LSTM in Practice

- Stock Prices
  - https://www.kdnuggets.com/2018/11/keras-long-short-term-memory-lstm-model-predict-stock-prices.html
  - https://www.datacamp.com/community/tutorials/lstm-python-stock-market
  - https://github.com/jaungiers/LSTM-Neural-Network-for-Time-Series-Prediction
  - https://www.altumintelligence.com/articles/a/Time-Series-Prediction-Using-LSTM-Deep-Neural-Networks

CNNs

For Images

CNN Core Idea

- At its most basic, convolutional neural networks can be thought of as a kind of neural network that uses many identical copies of the same neuron.
- This allows the network to have lots of neurons and express computationally large models while keeping the number of actual parameters – the values describing how neurons behave – that need to be learned fairly small.
- When programming, we write a function once and use it in many places – not writing the same code a hundred times in different places makes it faster to program, and results in fewer bugs.
- Similarly, a convolutional neural network can learn a neuron once and use it in many places, making it easier to learn the model and reducing error.

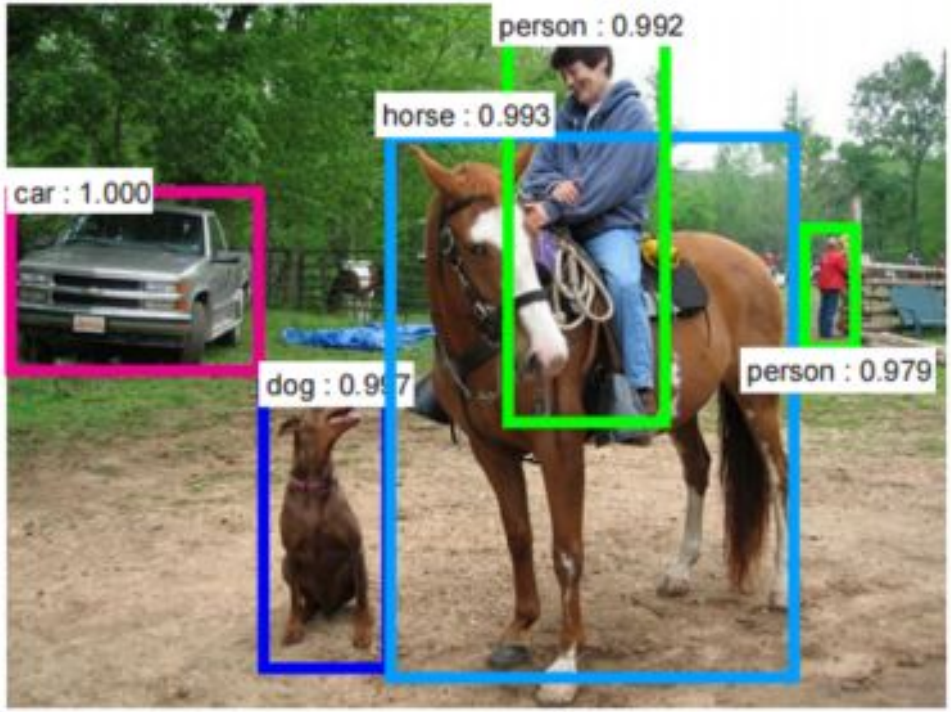
Recognize Scenes & Provide Captions

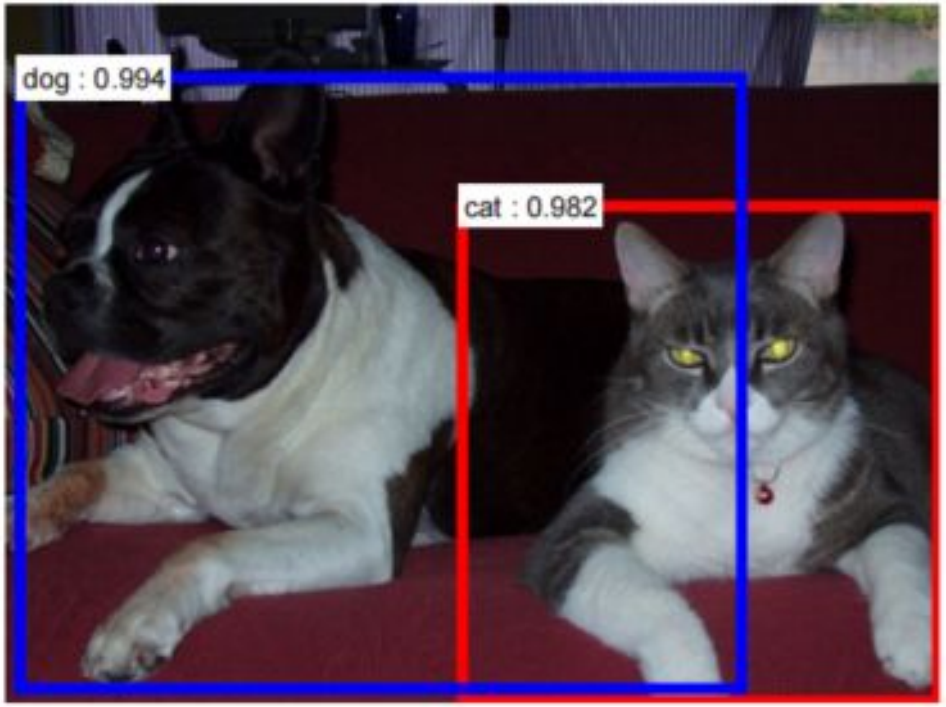
a soccer player is kicking a soccer ball

a street sign on a pole in front of a building

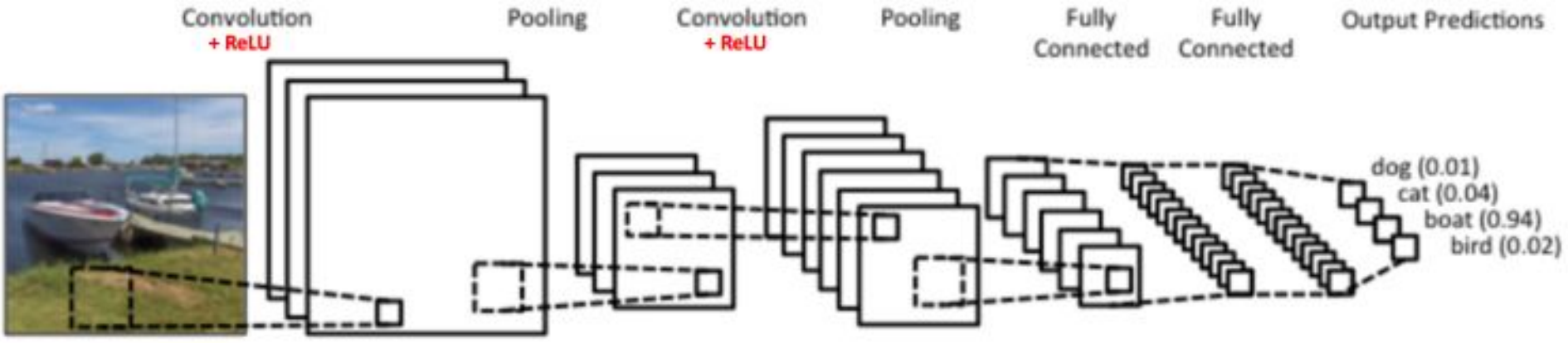
a couple of giraffe standing next to each other

Recognizing Objects, Humans, Animals, Text (e.g. NLP)





CNN for Image Classification



- LeNet (1998) was one of the very first convolutional neural networks which helped propel the field of Deep Learning.
  - http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
  - https://towardsdatascience.com/neural-network-architectures-156e5bad51ba
- At that time the LeNet architecture was used mainly for character recognition tasks such as reading zip codes, digits, etc.
- The architecture above is based on LeNet and used for image classification for four categories. Let's review it step by step.

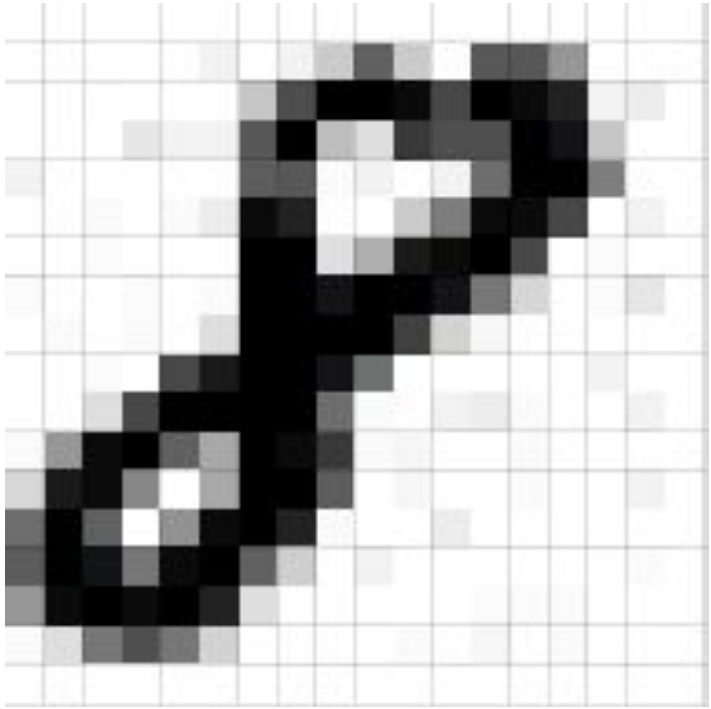
CNN Steps

There are four main operations in the CNN shown previously:

1. Convolution
2. Non Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification (Fully Connected Layer)

What is an image (data)?

- Essentially, every image can be represented as a matrix of pixel values.
- An image from a standard digital camera will have three channels (components) – red, green and blue – you can imagine those as three 2d-matrices stacked over each other (one for each color), each having pixel values in the range 0 to 255.
- A grayscale image, on the other hand, has just one channel.
- For the purposes here, we will only consider grayscale images, so we will have a single 2d matrix representing an image.
- The value of each pixel in the matrix will range from 0 to 255 – zero indicating black and 255 indicating white.

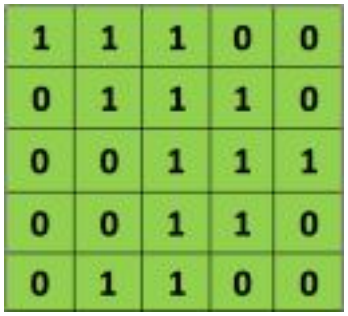


Convolution Step

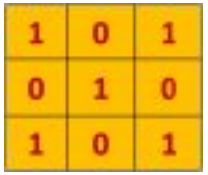
- CNNs derive their name from the “convolution” operator.
  - Element wise matrix multiplication and addition
  - The math behind it
    - https://cs231n.github.io/convolutional-networks/
    - https://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/
- The primary purpose of Convolution in case of a CNNs is to **extract features from the input image**.
- Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

Convolution Step

Consider a 5x5 pixel **image** with only values 0 or 1 (instead of 0-255) represented as green matrix A

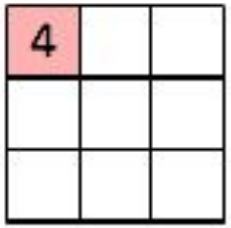


Consider another 3x3 yellow matrix B, which is like a **filter, or 'kernel' or 'feature detector'**



The convolution of A and B (A*B) results in a "**Convolved Feature" Or "Feature Map**" also of dimension 3x3

Image



Convolved Feature

Filters

*

- Different values of the filter matrix will produce different Feature Maps for the same input image
- This means that different filters can detect different features from an image, for example edges, curves etc.

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Filters

- In CNNs, **filters are not defined**. The value of each filter is learned during the training process.
  - Random initialization ensures that each filter learns to identify different features.
- We still need to specify parameters such as number of filters, filter size, architecture of the network etc. before the training process.
- The more filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images.

Another Filter Example

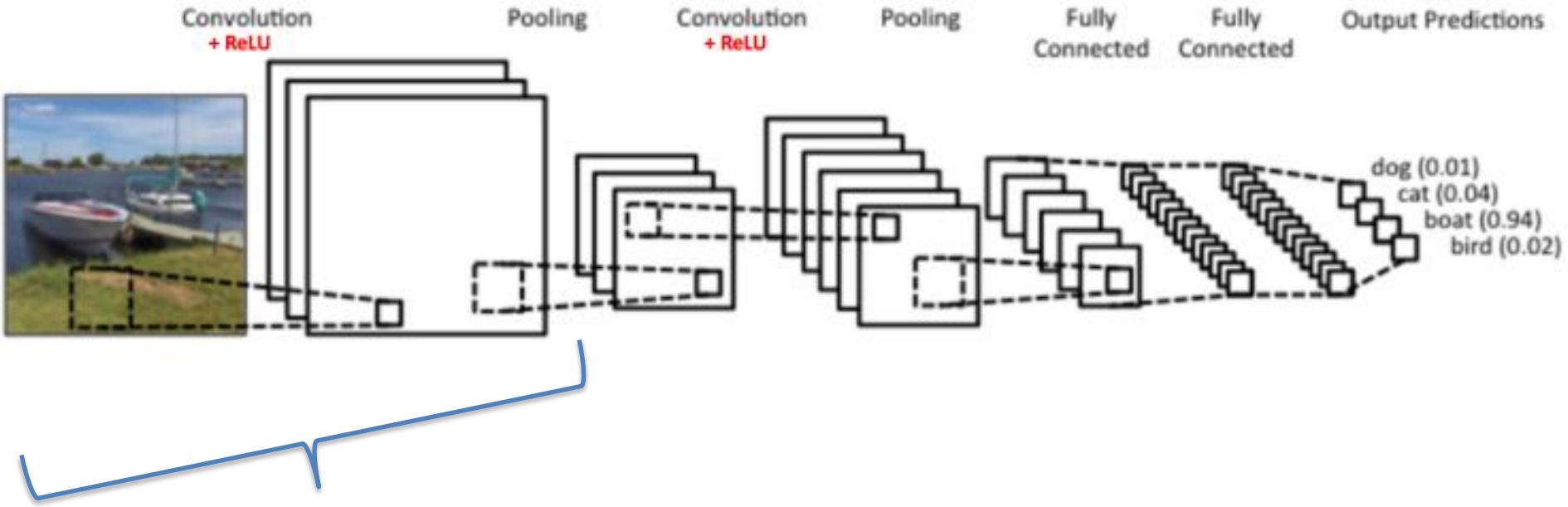
- A filter (with red outline) slides over the input image (convolution operation) to produce a feature map.
- The convolution of another filter (with the green outline), over the same image gives a different feature map as shown.
- It is important to note that the Convolution operation captures the local dependencies in the original image.
- Also notice how these two different filters generate different feature maps from the same original image. Remember that the image and the two filters above are just numeric matrices as we have discussed above.

Feature Map (Convolved Feature)

The size of the Feature Map is controlled by three parameters that we need to decide before the convolution step is performed:

- **Depth**: Depth corresponds to the number of filters we use for the convolution operation.
- **Stride**: Stride is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1 then we move the filters one pixel at a time. Having a larger stride will produce smaller feature maps.
- **Zero-padding**: Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix. A nice feature of zero padding is that it allows us to control the size of the feature maps. Adding zero-padding is also called *wide convolution*, and not using zero-padding would be a *narrow convolution*.

Review Where We Are



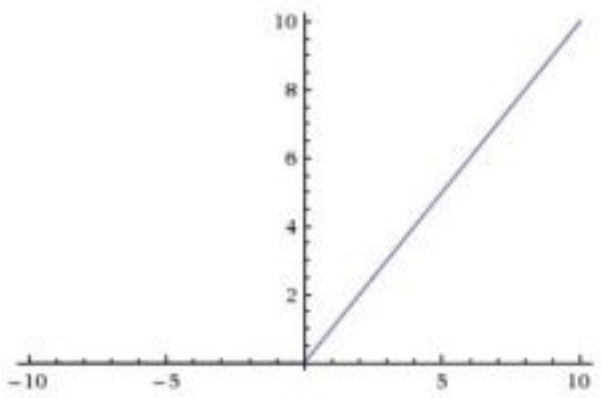
Feature Map having depth of 3 (since 3 filters have been used)

Convolution Operation

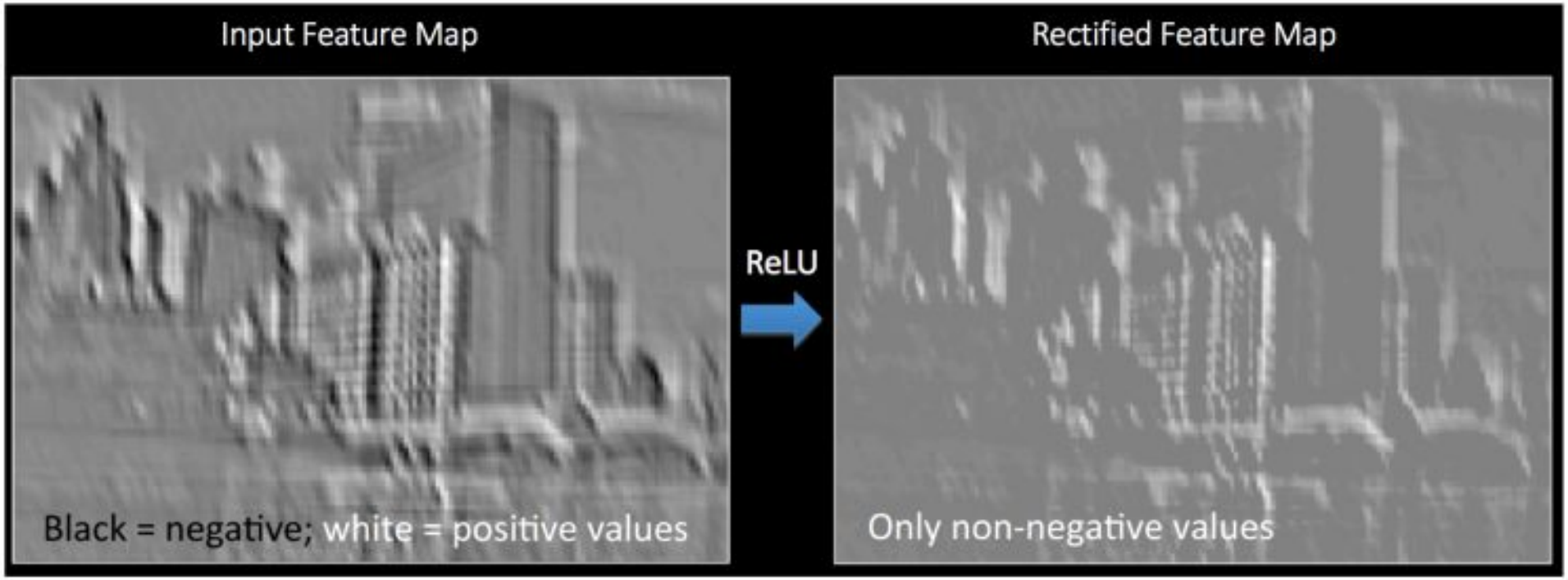
Introducing Non Linearity (ReLU)

- ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero.
- The purpose of ReLU is to introduce non-linearity in our CNN, since most of the real-world data we would want our CNN to learn would be non-linear
  - Convolution is a linear operation – element wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU.

Output = Max(zero, Input)



ReLU Step

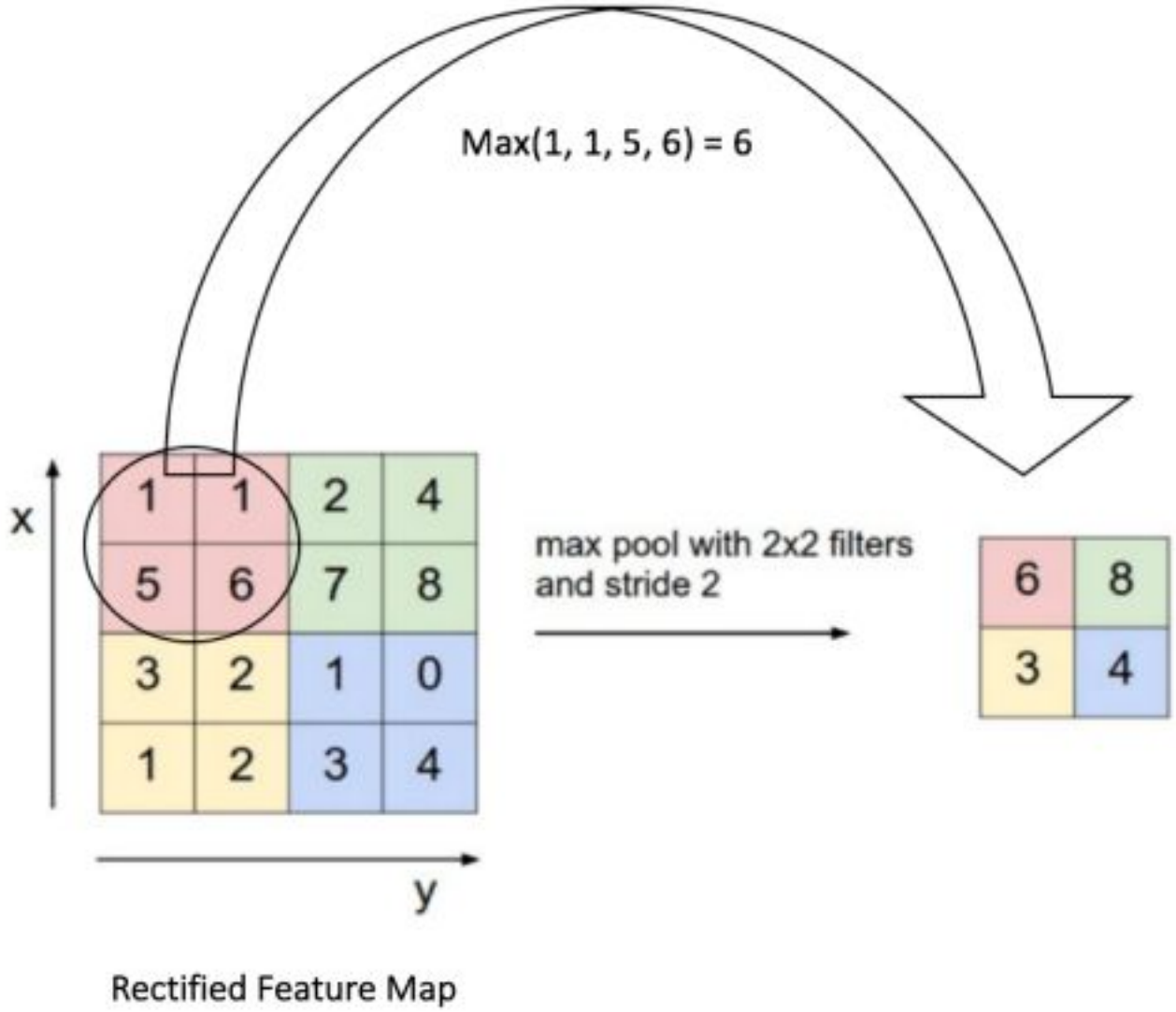


- The above shows the ReLU operation applied to one of the feature maps obtained previously.
- The output feature map is also referred to as the 'Rectified' feature map.
- Other non linear functions such as tanh or sigmoid can also be used instead of ReLU, but ReLU has been found to perform better in most situations.

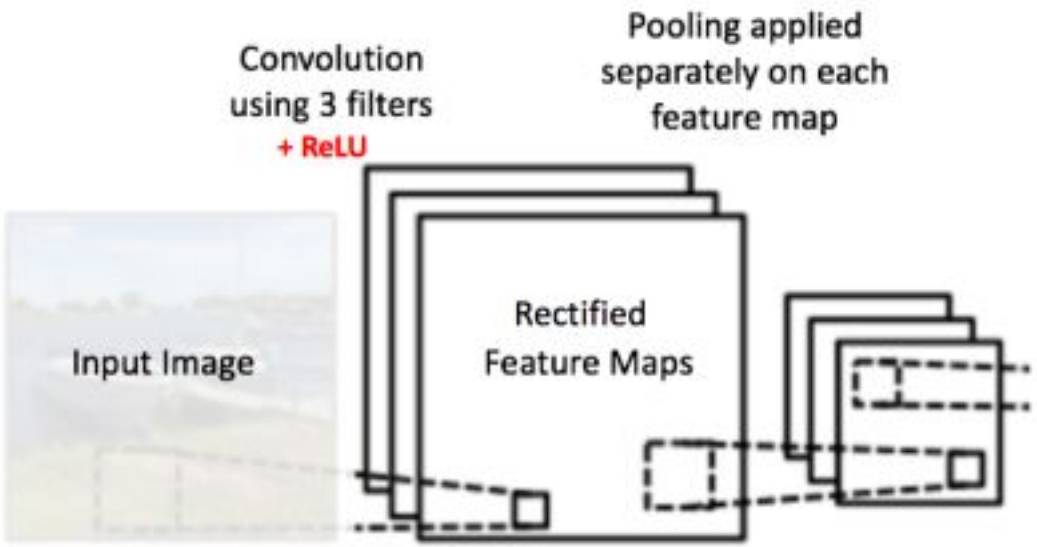
Pooling Step

- Spatial Pooling (also called subsampling or downsampling) **reduces the dimensionality of each feature** map but retains the most important information.
- Spatial Pooling can be of different types: Max, Average, Sum etc.
- In case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window.
- Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window.
- In practice, Max Pooling has been shown to work better.

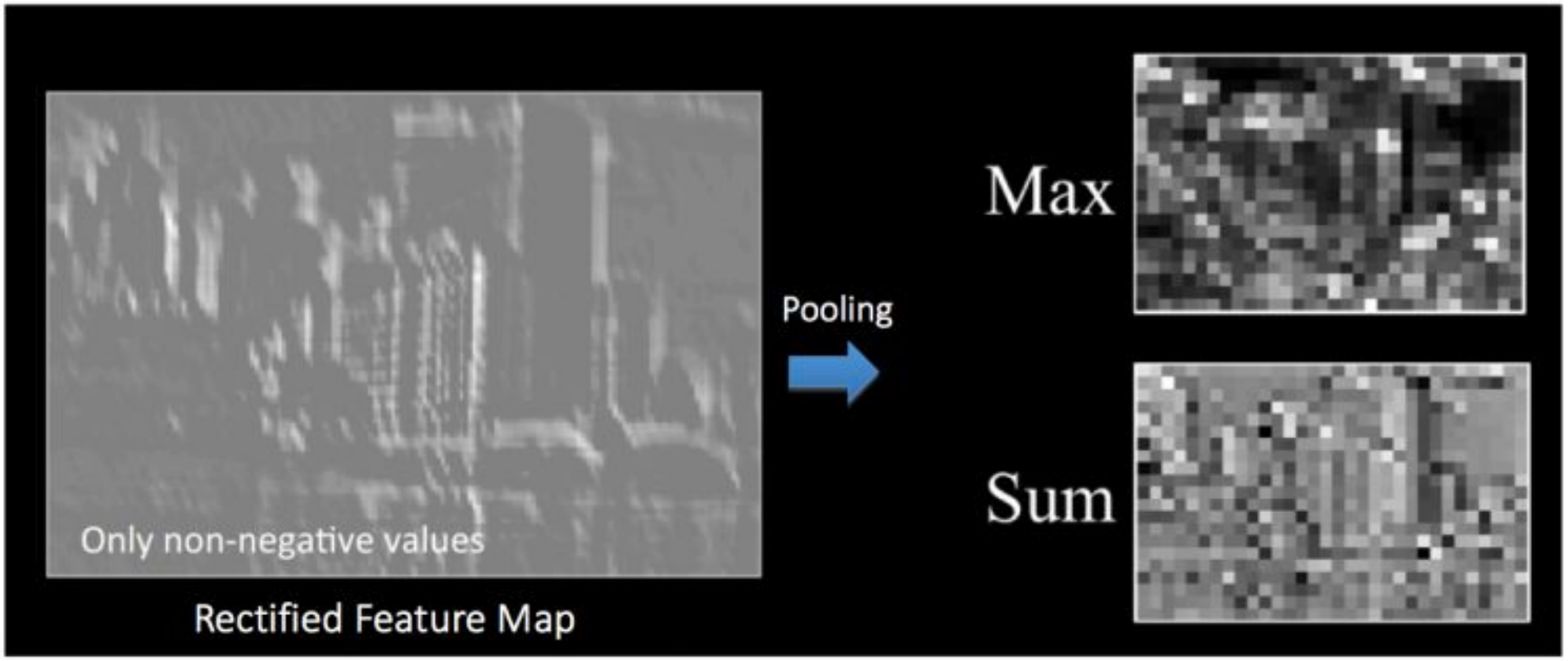
Max Pooling Operation



Pooling applied to Rectified Feature Maps



Pooling on the Rectified Feature Map

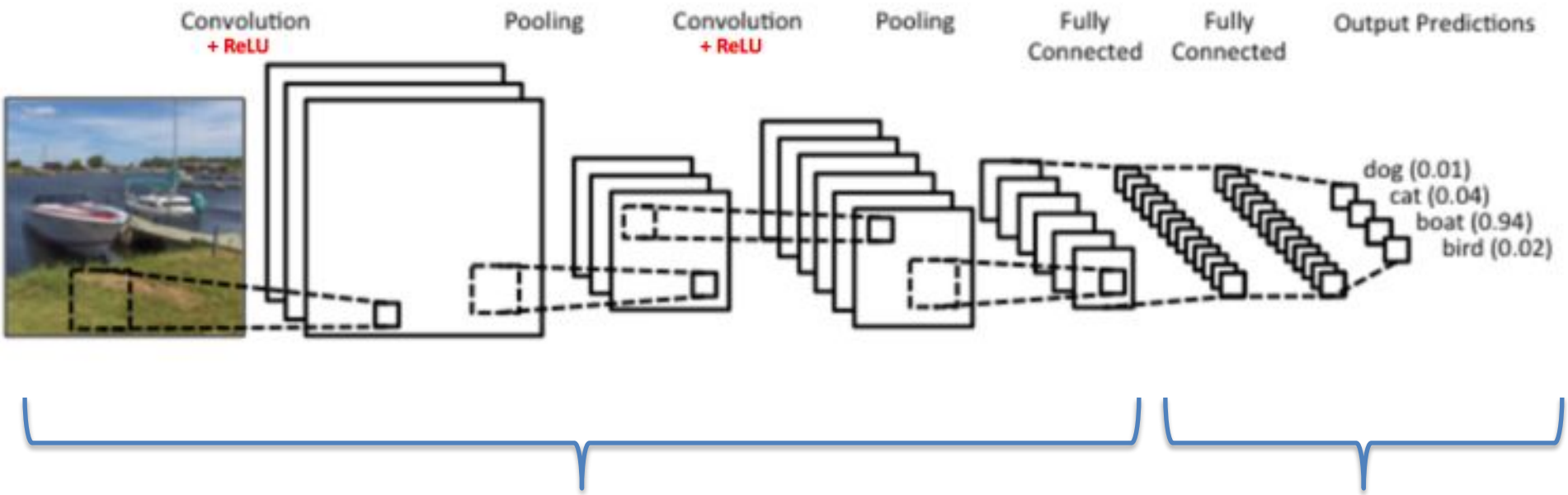


Why Pool?

The function of Pooling is to progressively reduce the spatial size of the input representation. In particular, pooling:

- makes the input representations (feature dimension) **smaller** and more manageable
- reduces the number of parameters and computations in the network, therefore, **controlling overfitting**
- makes the network **invariant** to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling – since we take the maximum / average value in a local neighborhood).
- helps us arrive at an almost **scale invariant representation** of our image (the exact term is "**equivariant**"). This is very powerful since we can detect objects in an image no matter where they are located.

Review Where We Are



Feature Extraction – 2 sets of Convo + ReLU + Pooling layers

Fully Connected Layer

Fully Connected Layer

- The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer (other classifiers like SVM can also be used, but will stick to softmax in this post). The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.
- The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.
- Apart from classification, adding a fully-connected layer is also a (usually) cheap way of learning non-linear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better.
- The sum of output probabilities from the Fully Connected Layer is 1. This is ensured by using the Softmax as the activation function in the output layer of the Fully Connected Layer.
- The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one.

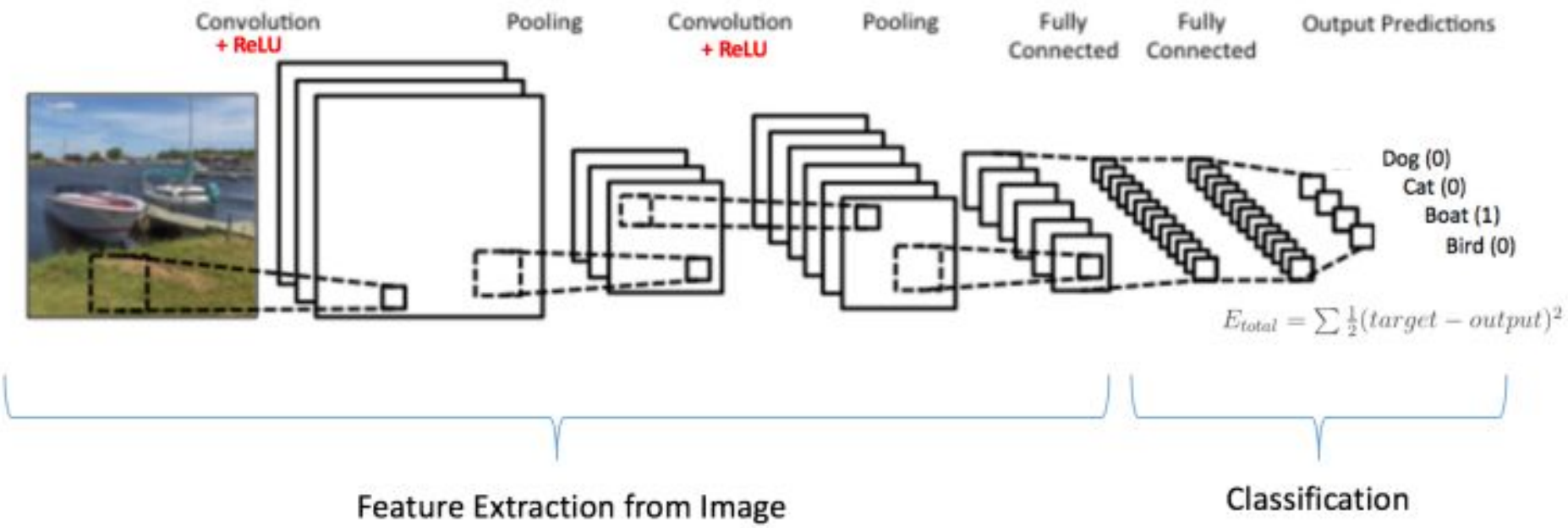
Training the CNN

- Input Image = Boat
- Target Vector = [0, 0, 1, 0] from [Dog, Cat, Boat, Bird]
- Step1:
  - We initialize all filters and parameters / weights with random values
- Step2:
  - The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.
  - Lets say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]
  - Since weights are randomly assigned for the first training example, output probabilities are also random.
- Step3:
  - Calculate the total error at the output layer (summation over all 4 classes)
  - Total Error = $\sum$ ½ (target probability – output probability) ^{2}

Training the CNN

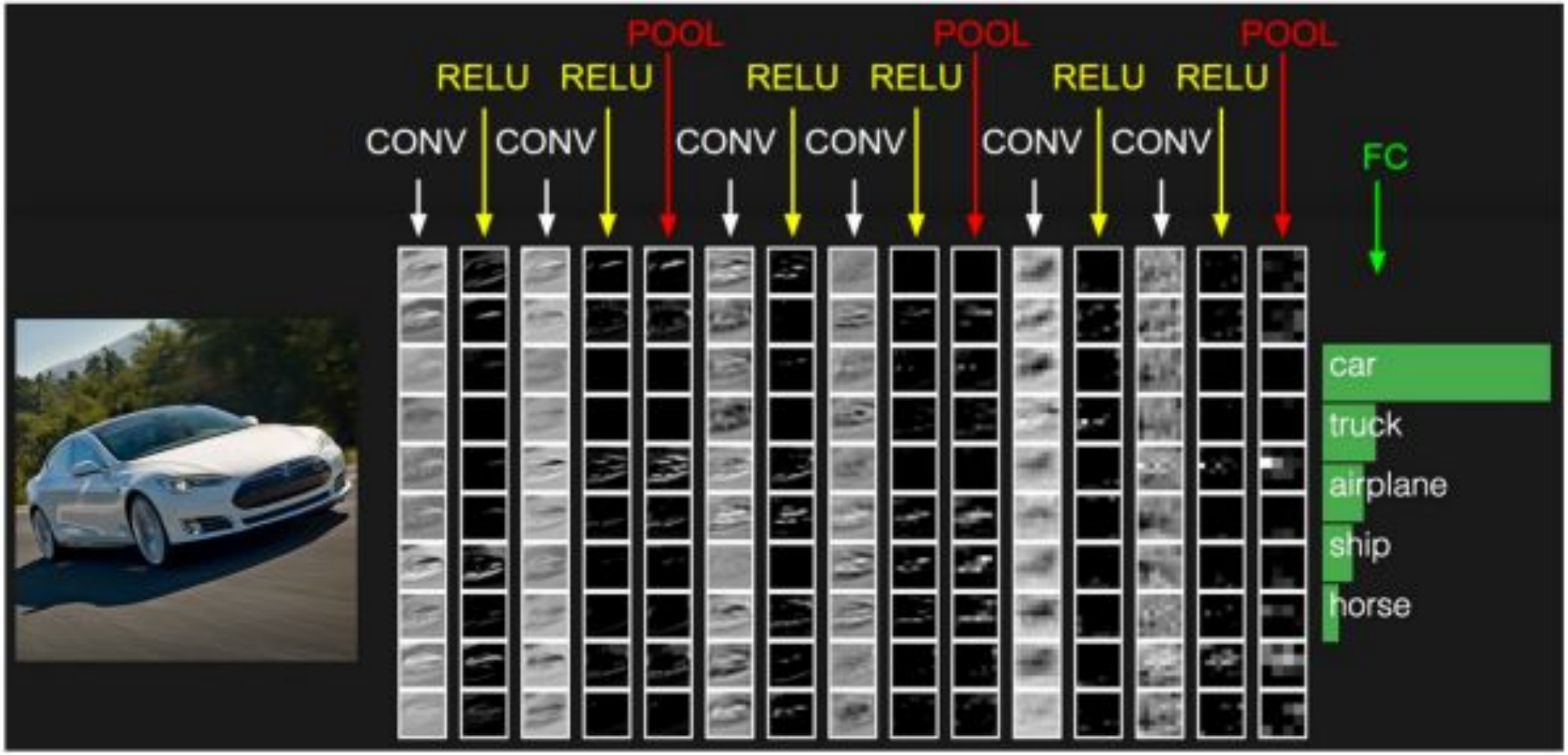
- Step4:
  - Use Backpropagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values to minimize the output error.
  - The weights are adjusted in proportion to their contribution to the total error.
  - When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0].
  - This means that the network has learnt to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced.
  - Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.
- Step5:
  - Repeat steps 2-4 with all images in the training set.
- The above steps train the CNN– this essentially means that all the weights and parameters of the CNN have now been optimized to correctly classify images from the training set.

Putting it all together



- When a new (unseen) image is input into the CNN, the network would go through the forward propagation step and output a probability for each class (for a new image, the output probabilities are calculated using the weights which have been optimized to correctly classify all the previous training examples).
- If our training set is large enough, the network will (hopefully) generalize well to new images and classify them into correct categories.

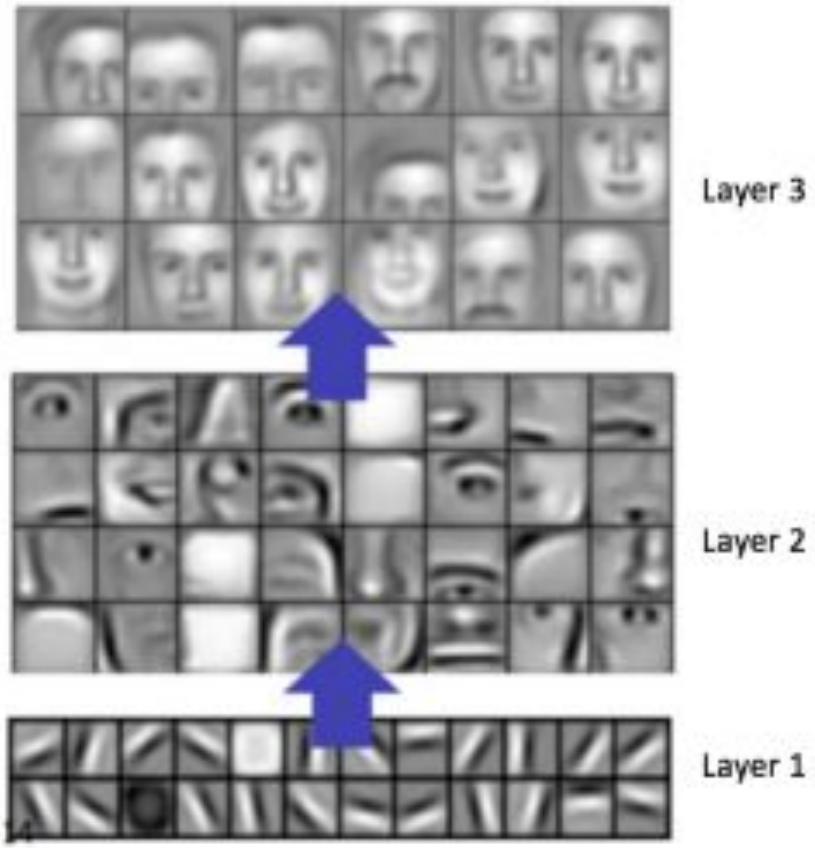
Another CNN Example



- Some of the best performing CNNs have tens of Convolution and Pooling layers.
- It is not necessary to have a Pooling layer after every Convolutional Layer.

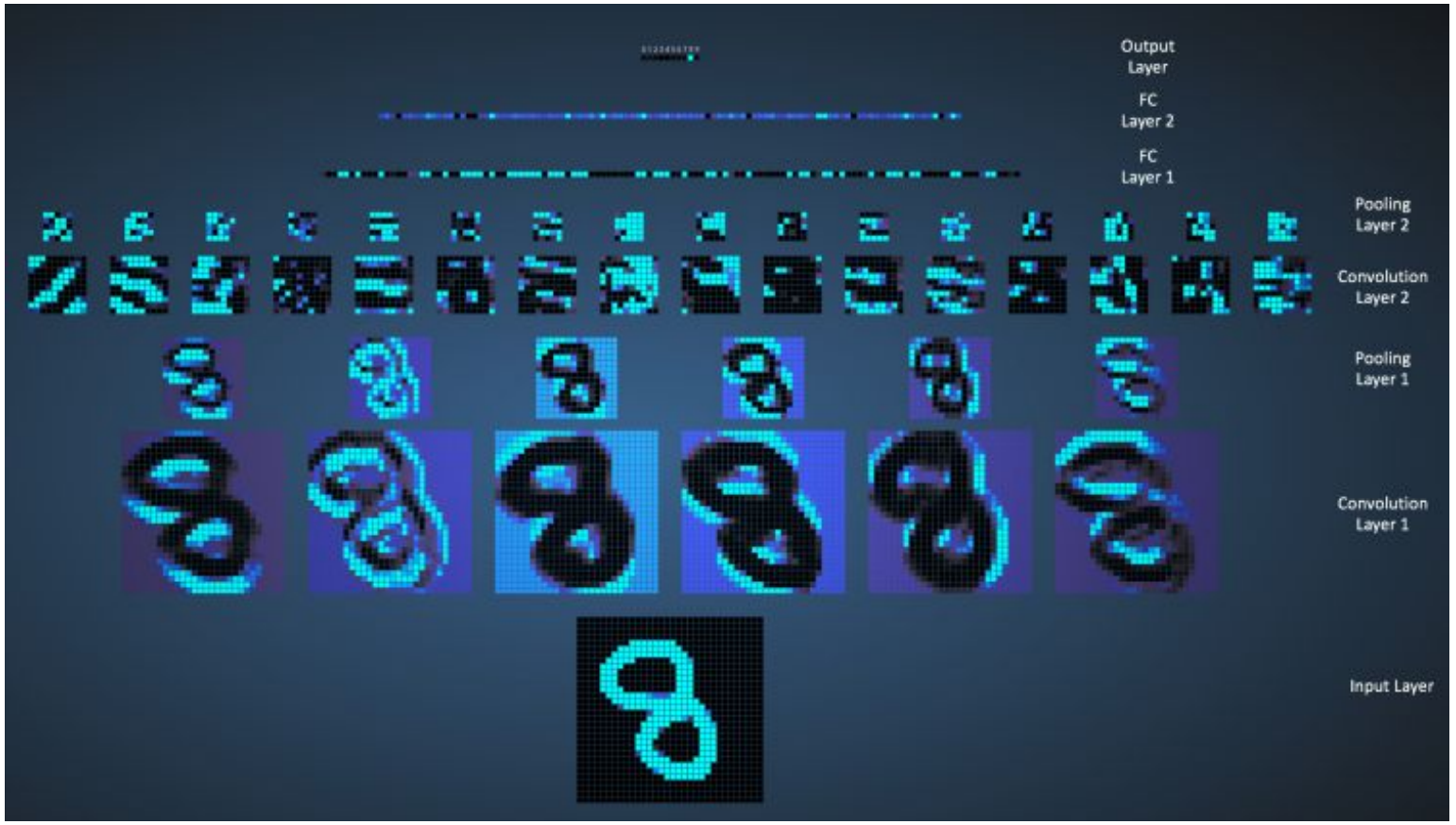
Visualizing CNNs

- In general, the more convolution steps we have, the more complicated features our network will be able to learn to recognize.
- For example, in Image Classification a CNN may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to detect higher-level features, such as facial shapes in higher layers.
- This is demonstrated here – these features were learned using a Convolutional Deep Belief Network.
- The figure is included here just for demonstrating the idea (this is only an example: real life convolution filters may detect objects that have no meaning to humans).

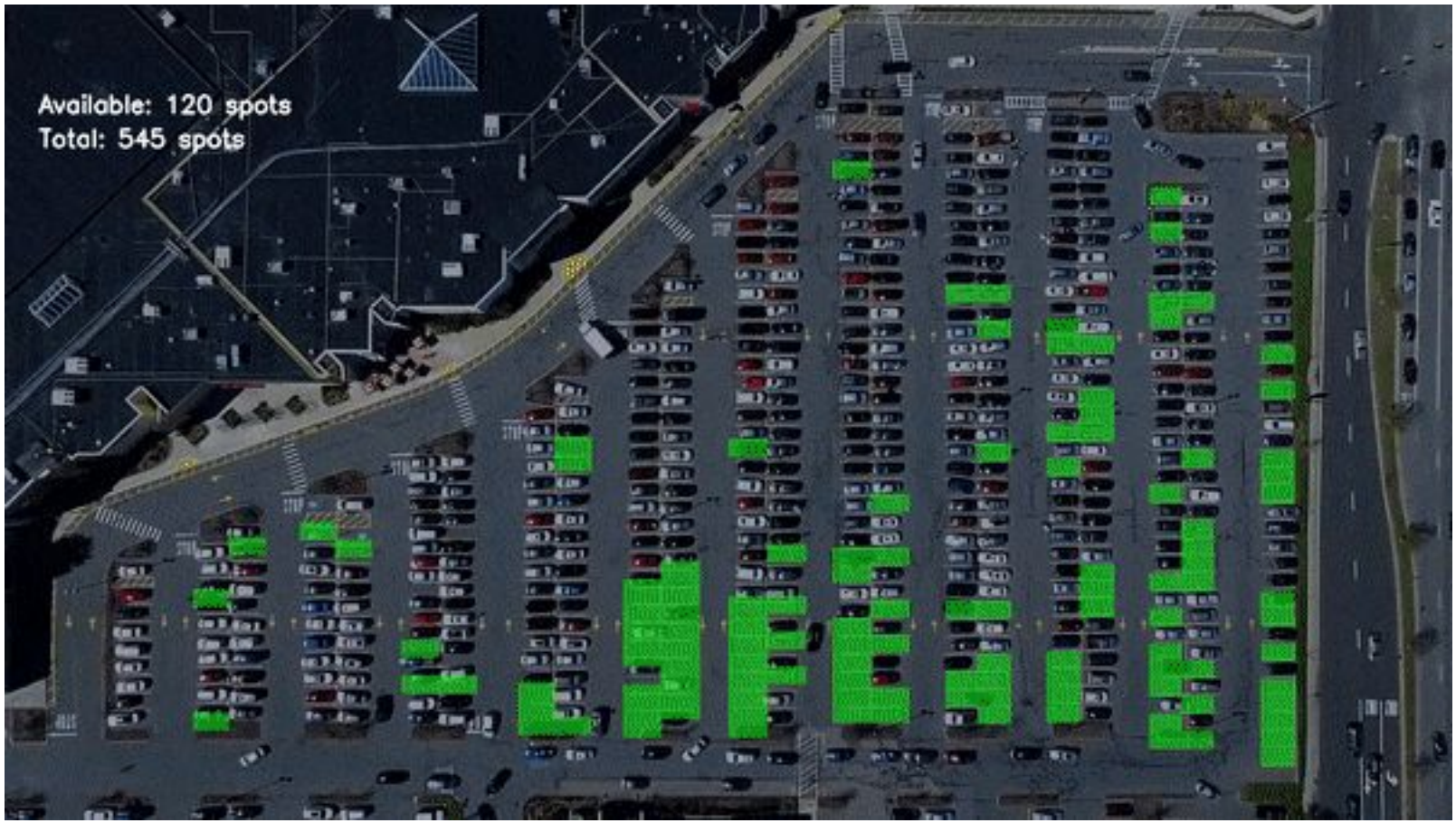


Visualizing CNNs

- https://www.cs.ryerson.ca/~aharley/vis/



Parking Lot Full = $$$?



https://towardsdatascience.com/find-where-to-park-in-real-time-using-opencv-and-tensorflow-4307a4c3da03

Google Scholar

convolutional neural network in finance

Articles

About 35,100 results (**0.10** sec)

Any time
Since 2021
Since 2020
Since 2017
Custom range...

Sort by relevance
Sort by date

Any type
include patents
include citations

Review articles

Create alert

Financial time-series data analysis using deep **convolutional neural networks**
JF Chen, WL Chen, CP Huang... - ... conference on cloud ..., 2016 - ieeexplore.ieee.org
A novel **financial** time-series analysis method based on deep learning technique is proposed in this paper. In recent years, the explosive growth of deep learning researches ...
☆ Cite Cited by 85 Related articles All 4 versions

[HTML] **Financial** quantitative investment using **convolutional neural network** and deep learning technology
C Chen, P Zhang, Y Liu, J Liu - Neurocomputing, 2020 - Elsevier
In order to make **financial** investment more stable and more profitable, **convolutional neural network** (CNN) and deep learning technology are used to quantify **financial** investment, so ...
☆ Cite Cited by 9 Related articles All 2 versions

[HTML] Bankruptcy prediction using imaged **financial** ratios and **convolutional neural networks**
T Hosaka - Expert systems with applications, 2019 - Elsevier
Convolutional neural networks are being applied to identification problems in a variety of fields, and in some areas are showing higher discrimination accuracies than conventional ...
☆ Cite Cited by 130 Related articles All 5 versions

Financial trading model with stock bar chart image time series with deep **convolutional neural networks**
OB Sezer, AM Ozbayoglu - arXiv preprint arXiv:1903.04610, 2019 - arxiv.org
Even though computational intelligence techniques have been extensively utilized in **financial** trading systems, almost all developed models use the time series data for price ...
☆ Cite Cited by 25 Related articles All 8 versions »

Forecasting stock prices from the limit order book using **convolutional neural networks**
A Tsantekidis, N Passalis, A Tefas... - 2017 IEEE 19th ..., 2017 - ieeexplore.ieee.org
... **financial** markets. In this work we proposed a deep learning methodology, based on **Convolutional Neural Networks** ... derived from the order book of **financial** exchanges. The dataset ...
☆ Cite Cited by 193 Related articles All 8 versions

GANs

Generative Adversarial Networks

Which Person is Real?