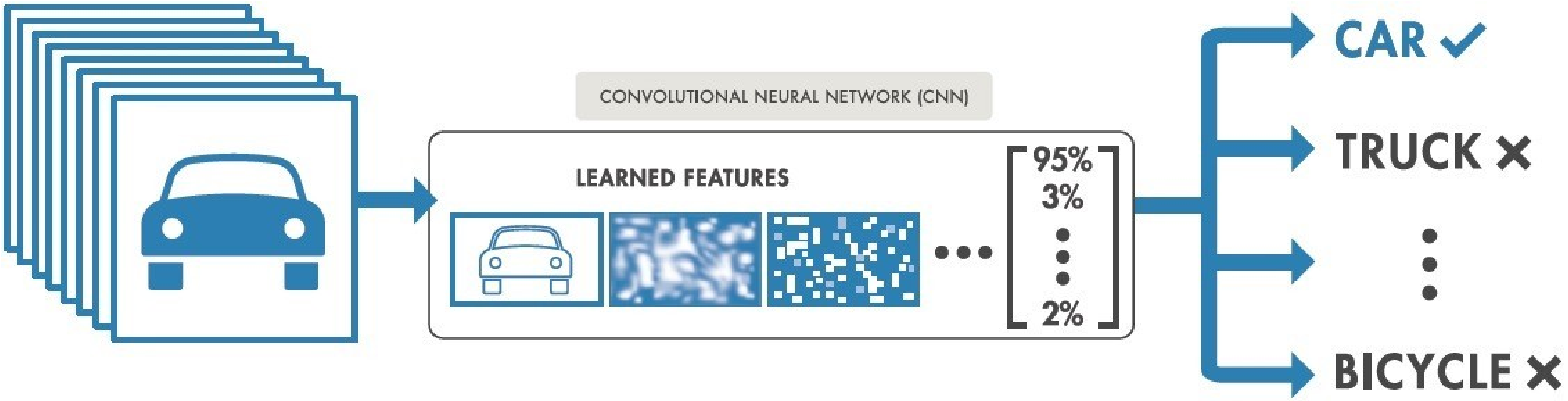
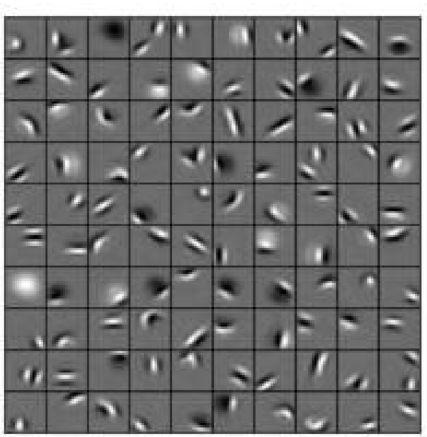
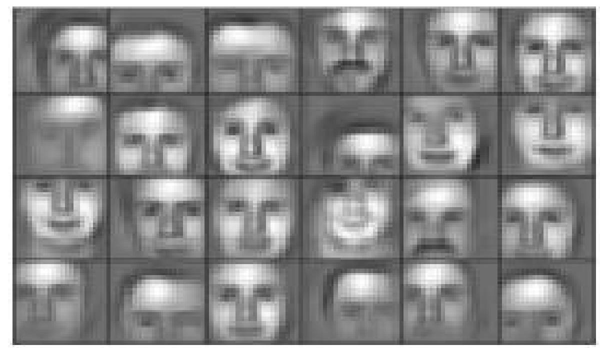
Convolutional Neural Network (CNN) and Recursive Neural Networks (RNN)

Image Classification



Hierarchical Features of Images



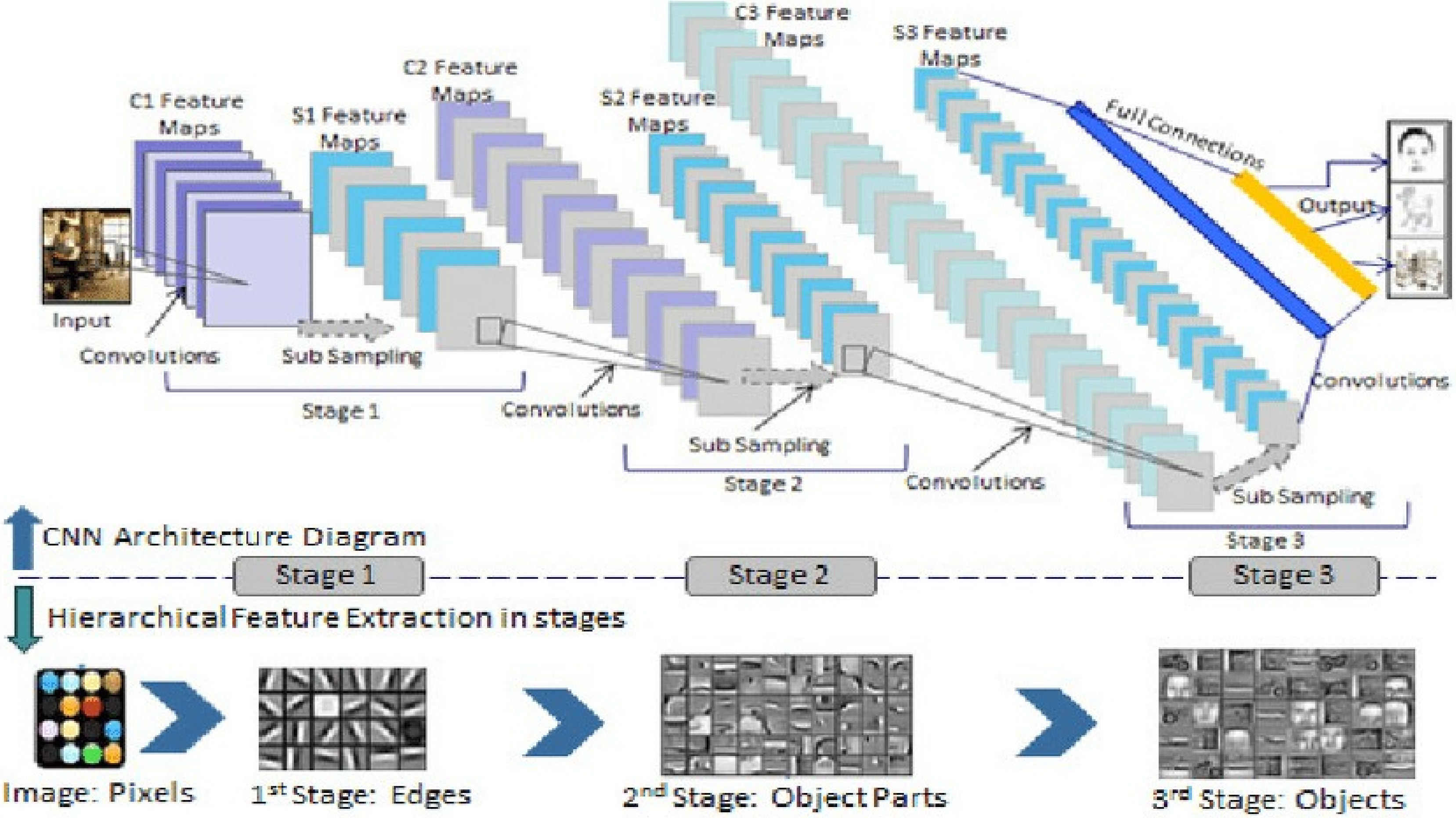


Low Level
(Edges, dark spots)

Mid Level
(Eyes, ears, etc.)

High Level
(Facial structure)

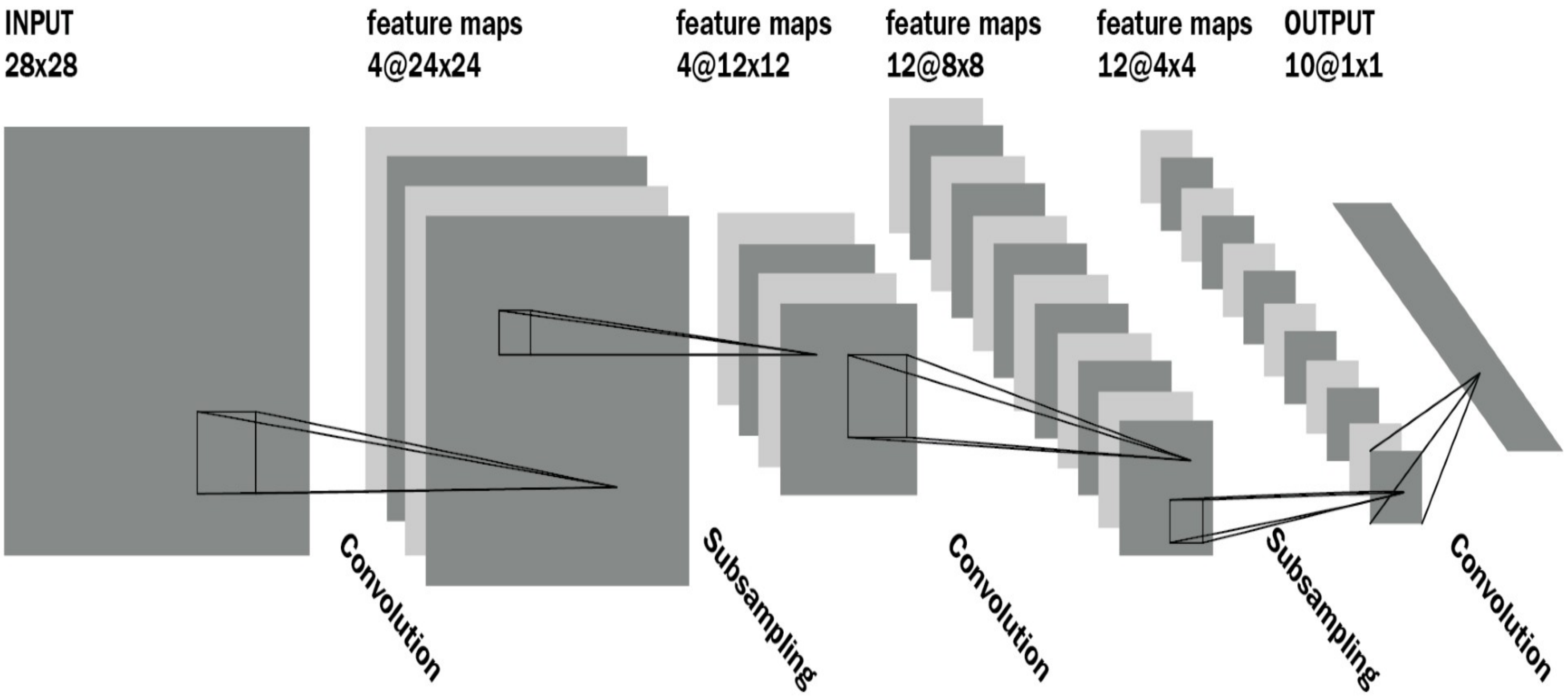
Feature Detection for Classification



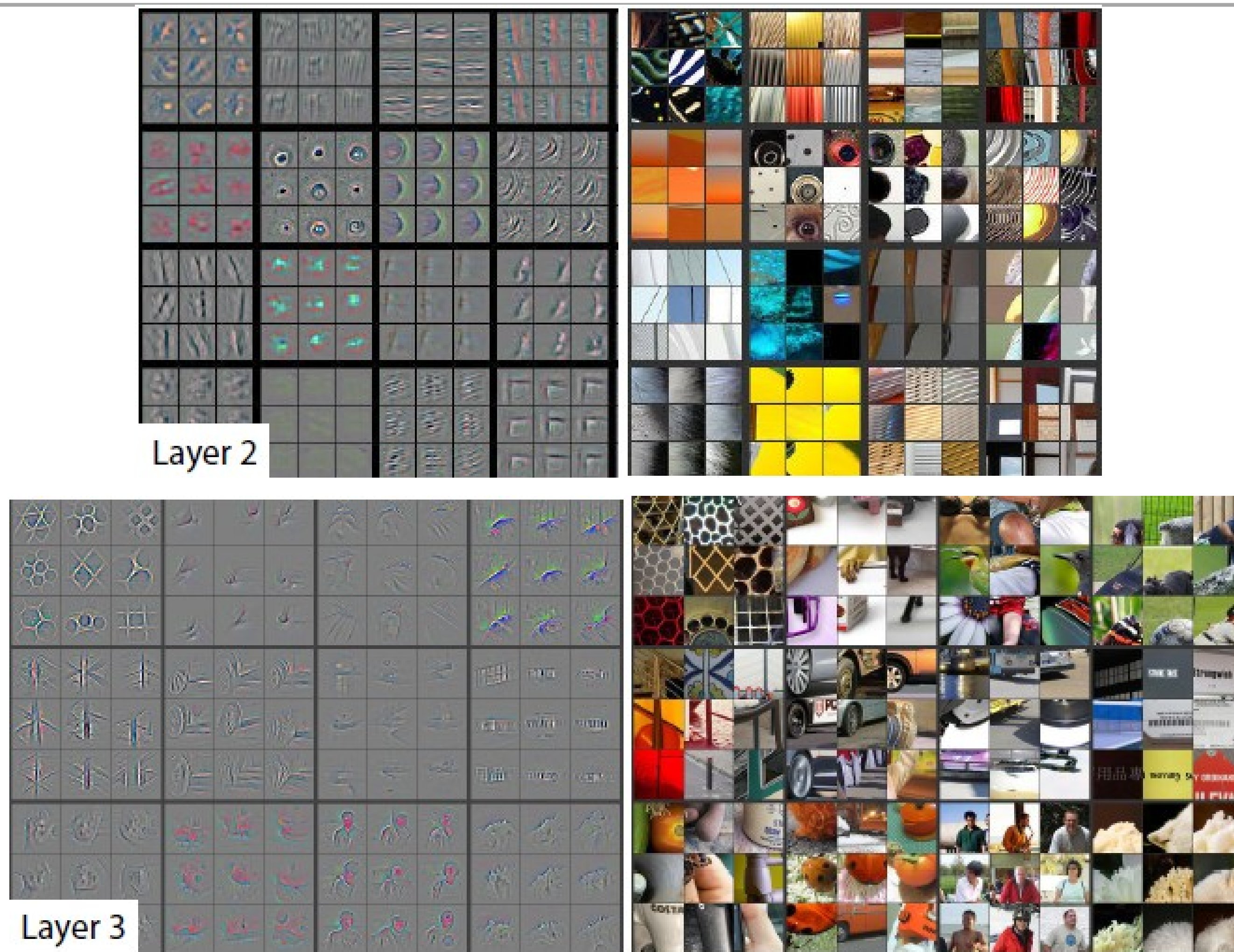
CNN: Keywords

- **Convolutional layer**
  - Feature matching
  - Convolutional block/filter
- **Pooling layer:**
  - Dimension reduction
  - Max-pooling
- **Dropout:**
  - Address the overfitting issue
  - Removing nodes randomly from a network temporarily during training to prevent one of a small number of nodes from getting largest weights and dominating the outputs
- **Flatten layer**
  - Reshape the data to the shape of the required output
  - Sigmoid function and softmax function

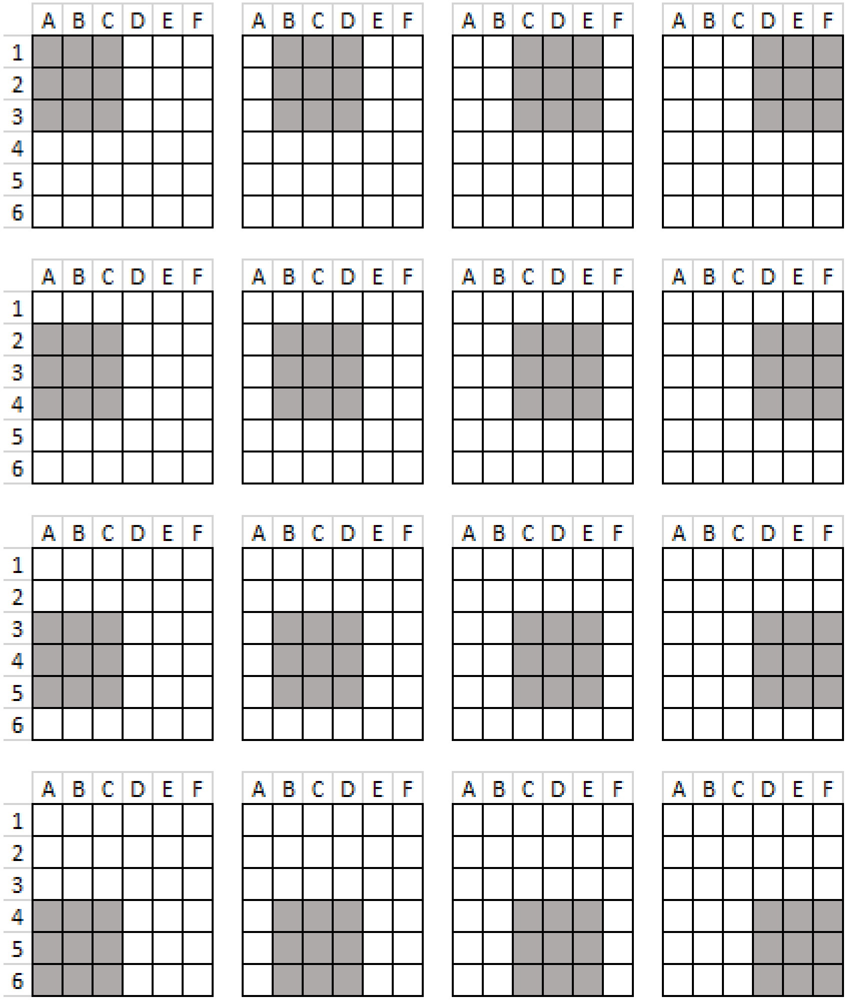
CNN: The **LeNet** architecture



CNN: Convolution



CNN: Convolution



CNN: Convolution

Input Layer

	A	B	C	D	E	F
1	0.5	0.3	0.1			
2	0.2	0.6	0.1			
3	0.1	0.1	0.7			
4				0.5	0.6	0.7
5				0.2	0.1	0.1
6				0.1	0.1	0.0

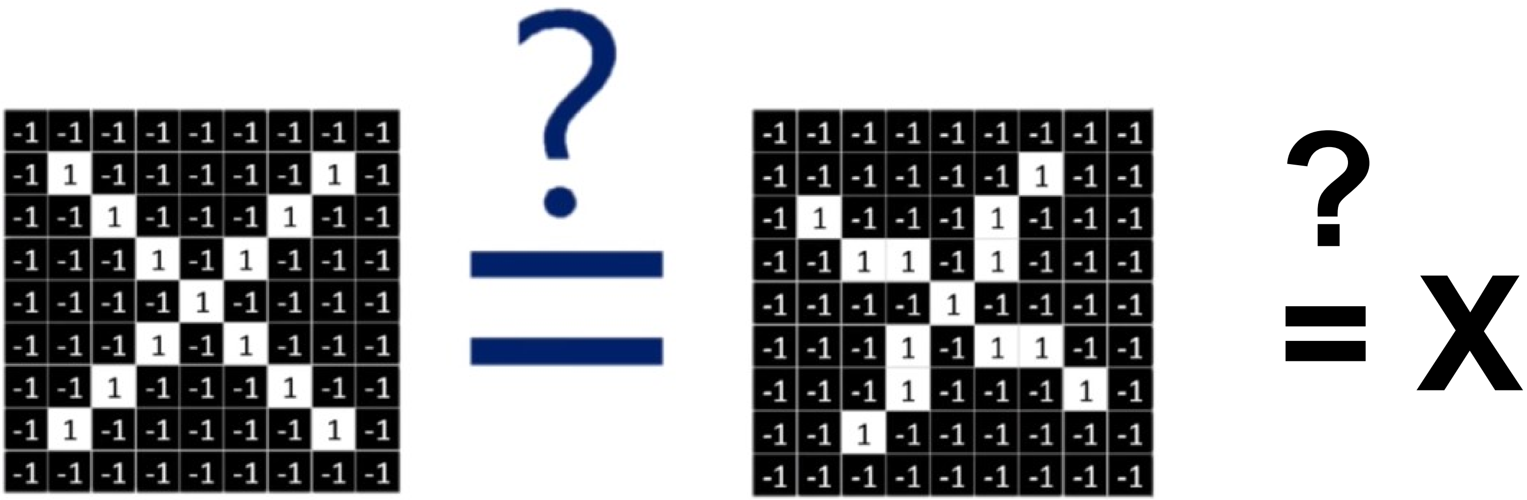
Conv. Block

3	1	1
1	3	1
1	1	3

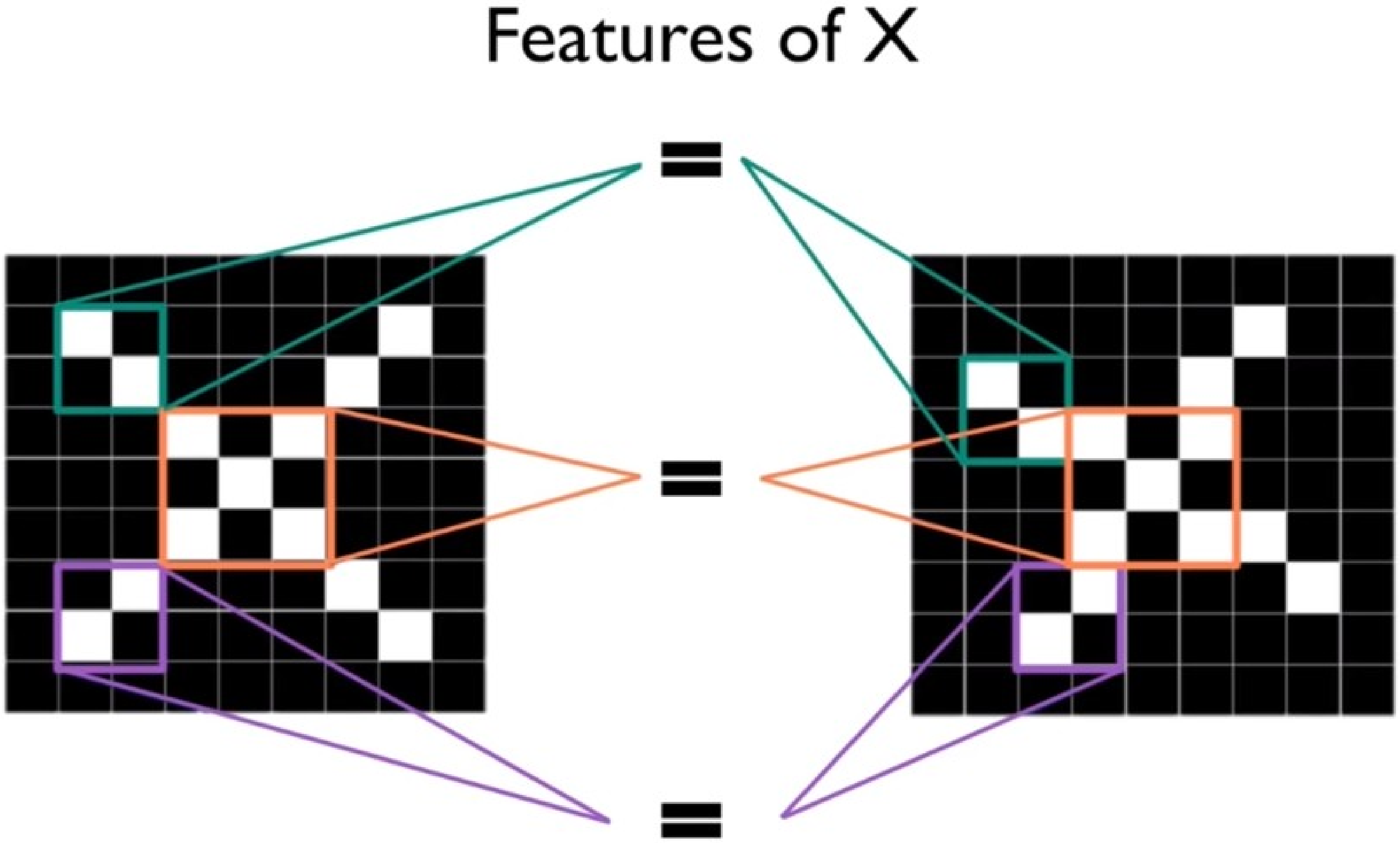
Output Layer

6.3	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	XX
XX	XX	XX	3.6

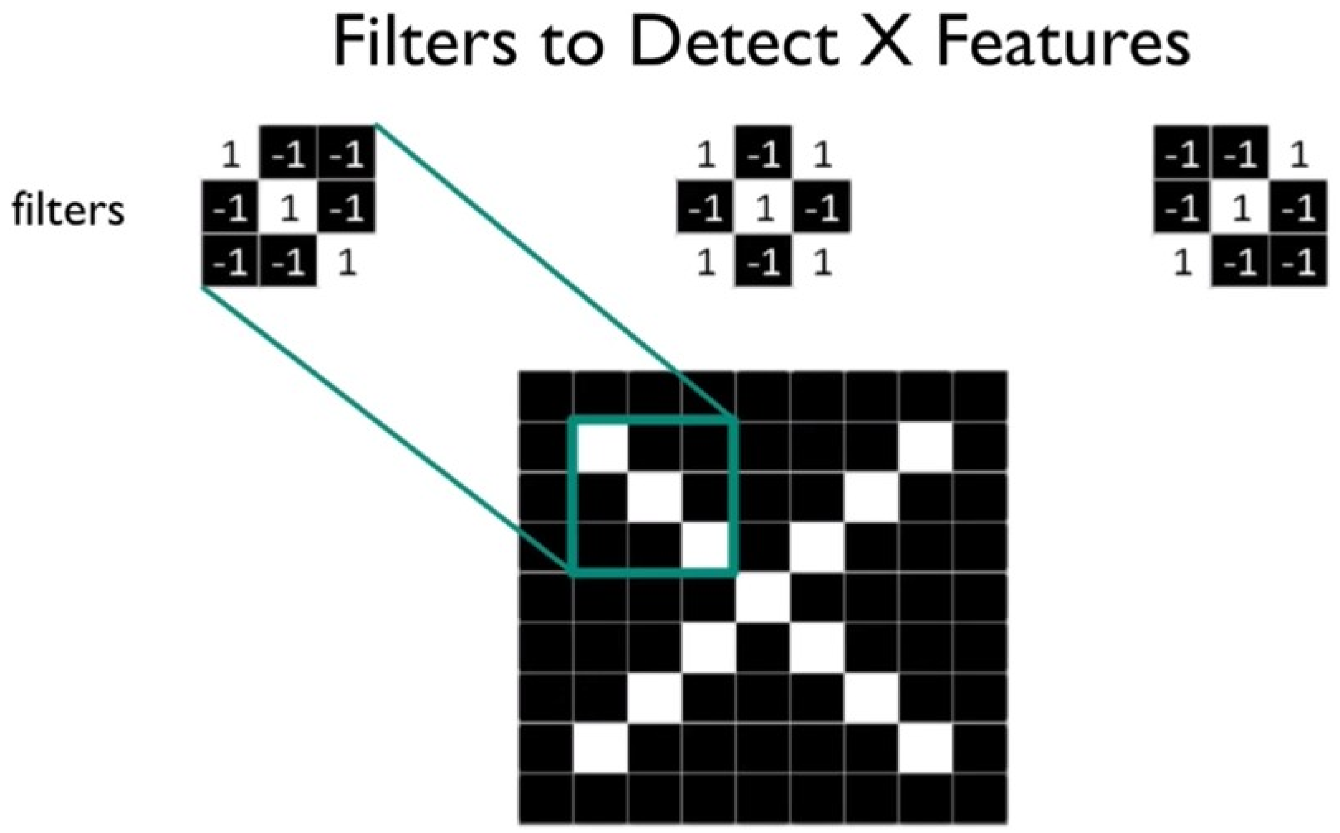
CNN: Example


-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 -1 -1 -1
-1 -1 1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 -1 -1 1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
?
=
-1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 1 -1 -1
-1 1 -1 -1 -1 1 -1 -1 -1
-1 -1 1 1 -1 1 -1 -1 -1
-1 -1 -1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 -1 1 1 -1 -1
-1 -1 -1 1 -1 -1 -1 1 -1
-1 -1 1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1
?
= X

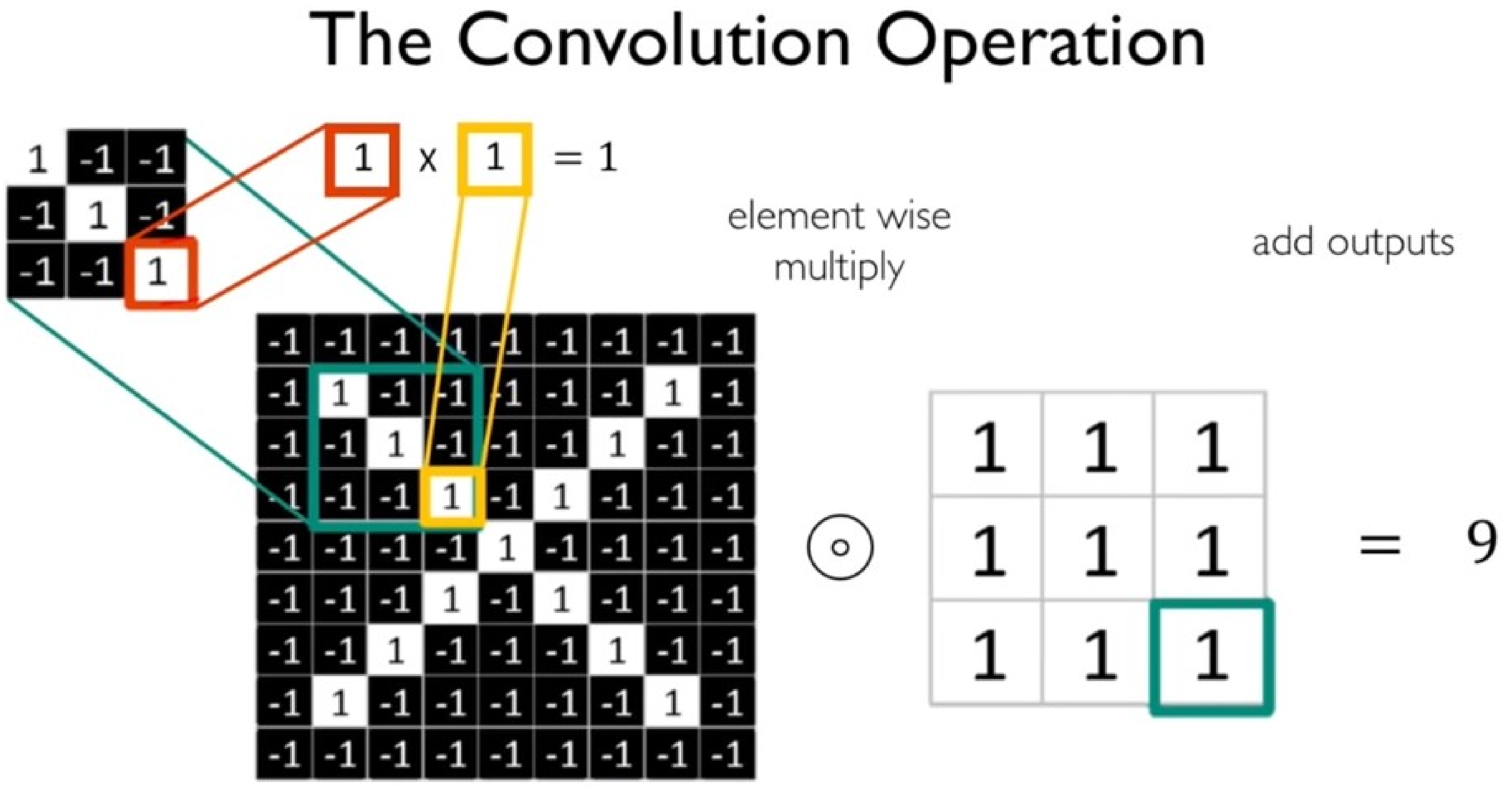
CNN: Example



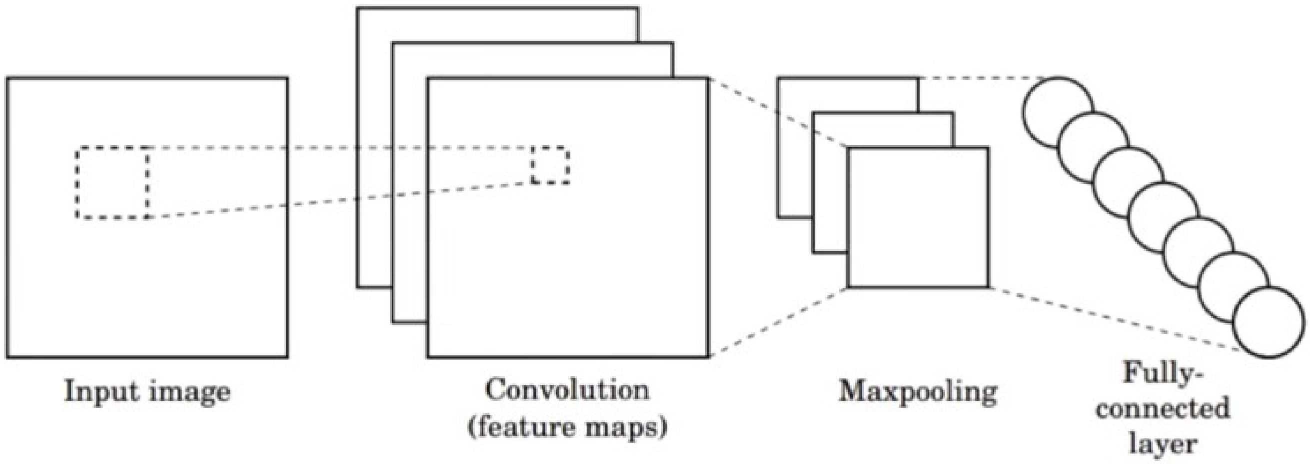
CNN: Example


Filters to Detect X Features
filters
1 -1 -1
-1 1 -1
-1 -1 1
1 -1 1
-1 1 -1
1 -1 1
-1 -1 1
-1 1 -1
1 -1 -1

CNN: Example


The Convolution Operation
1 x 1 = 1
element wise
multiply
add outputs
= 9

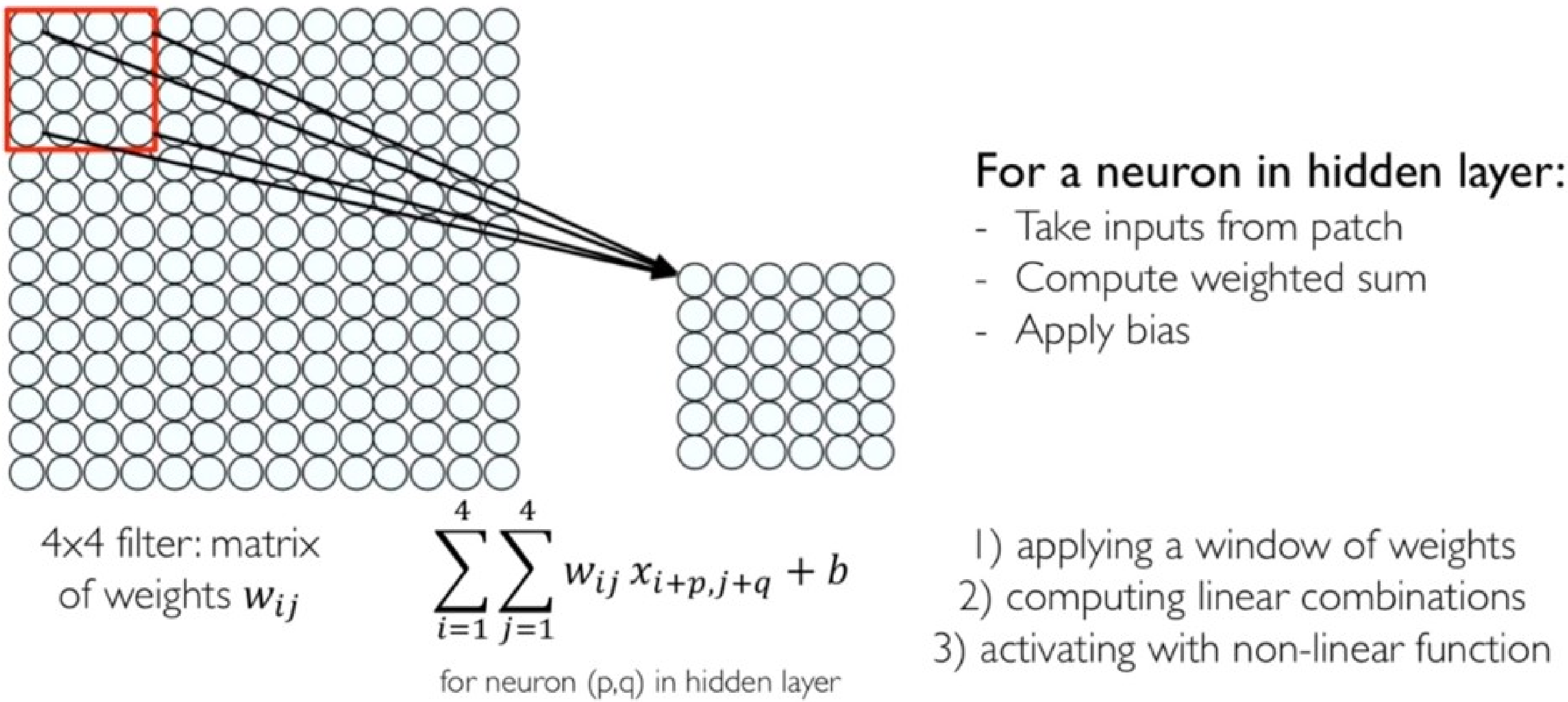
CNN: Architecture of Operation



1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

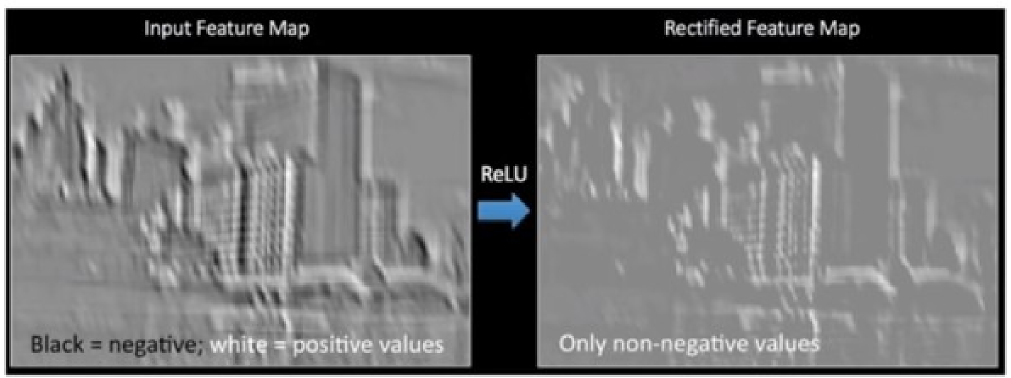
Train model with image data.
Learn weights of filters in convolutional layers.

CNN: Architecture of Operation

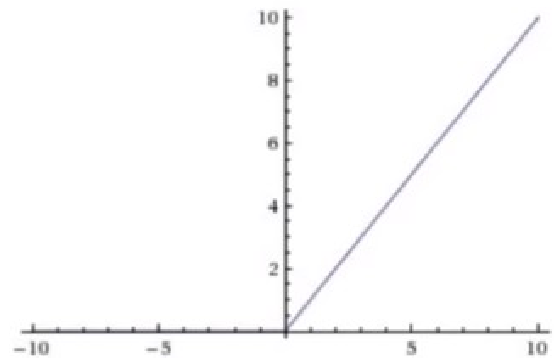


CNN: Non-Linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**



Rectified Linear Unit (ReLU)



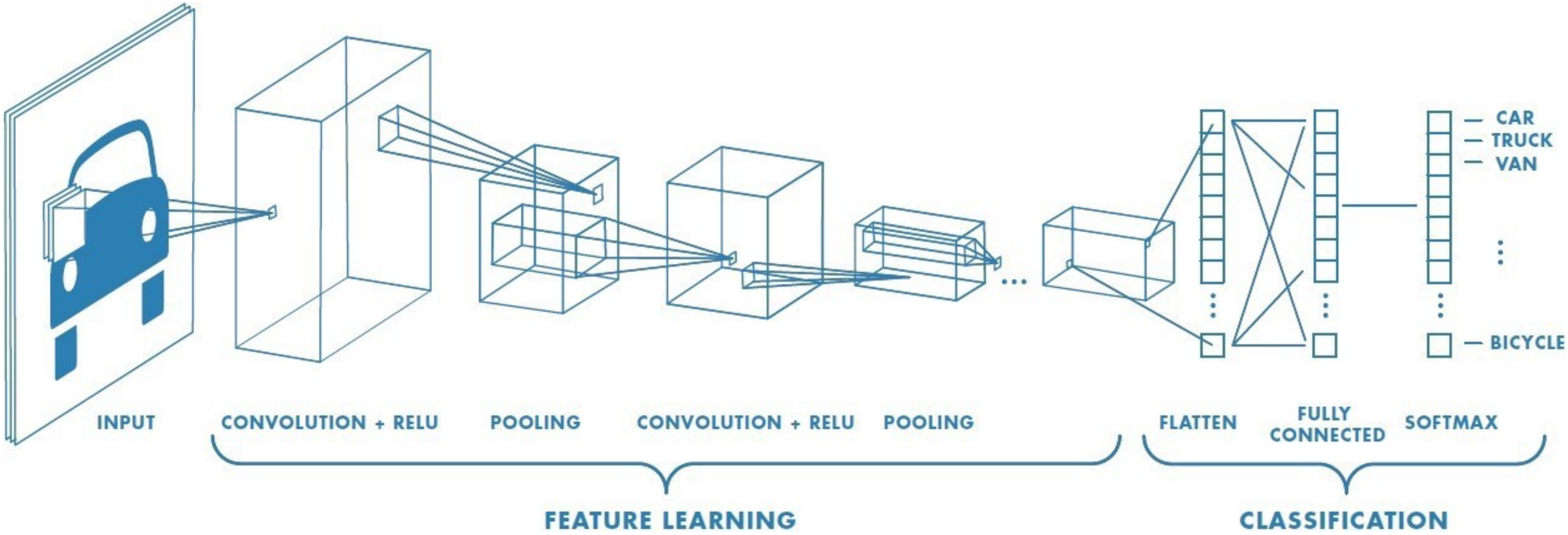
$$g(z) = \max(0, z)$$

CNN: Max-Pooling

	A	B	C	D	E	F
1	7	0	8	6	2	1
2	6	6	0	8	4	2
3	2	2	1	2	3	6
4	5	0	2	2	2	5
5	4	2	2	1	9	7
6	7	2	0	5	2	4

7	8	4
5	2	6
7	5	9

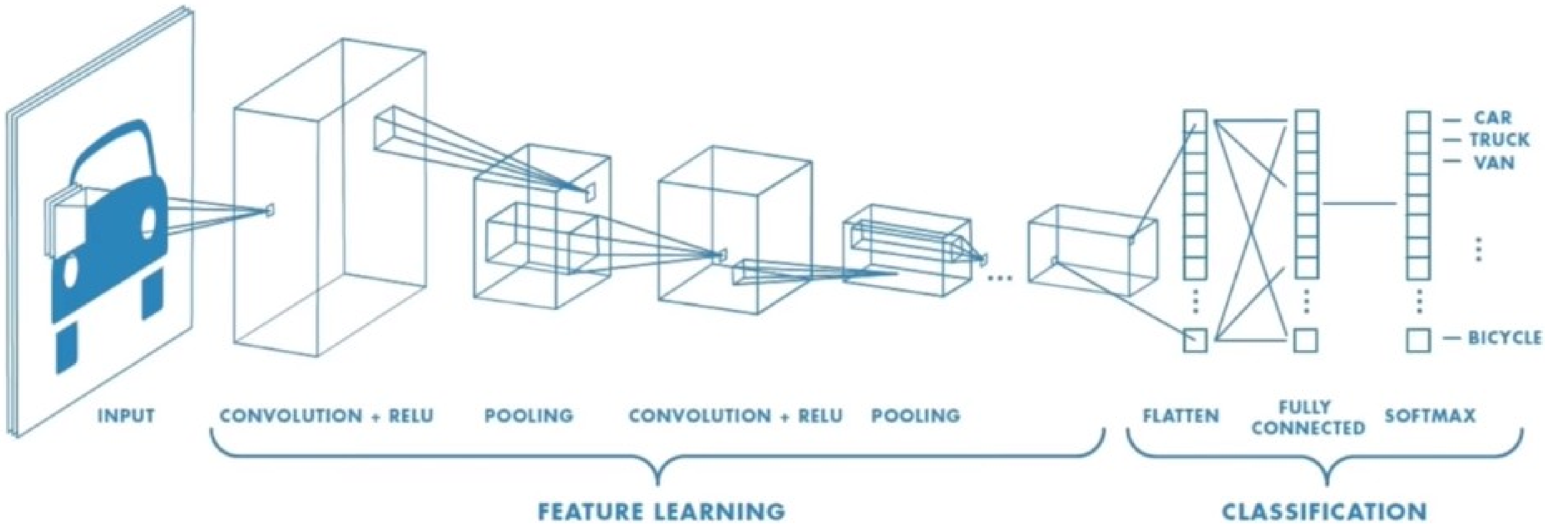
CNN: Architecture of Operation



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\mathrm{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

CNN: Backward Propagation for Training

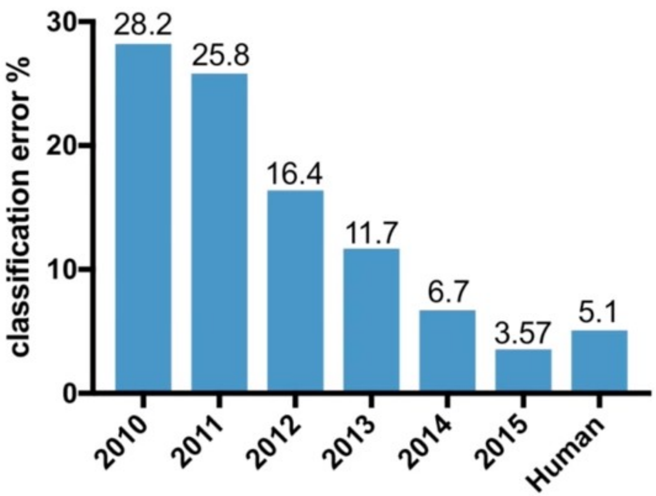


Learn weights for convolutional filters and fully connected layers

Backpropagation: cross-entropy loss

$$J(\boldsymbol{\theta}) = \sum_i y^{(i)} \log(\hat{\boldsymbol{y}}^{(i)})$$

ImageNet Challenge: Classification



2012: AlexNet. First CNN to win.
- 8 layers, 61 million parameters

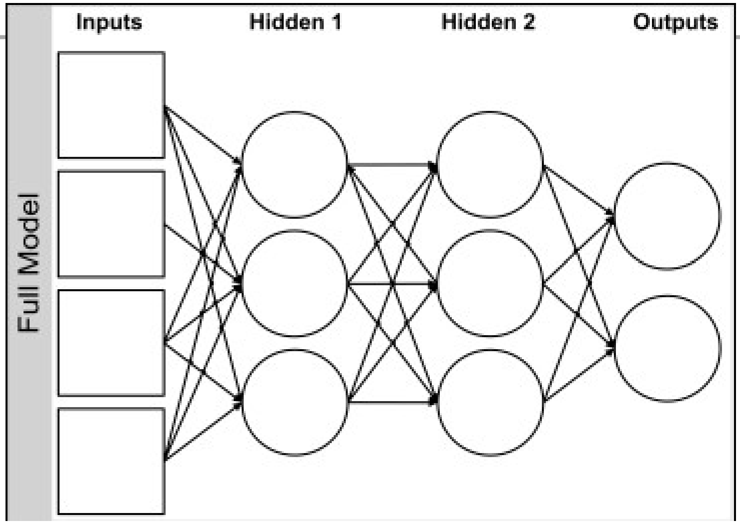
2013: ZFNet
- 8 layers, more filters

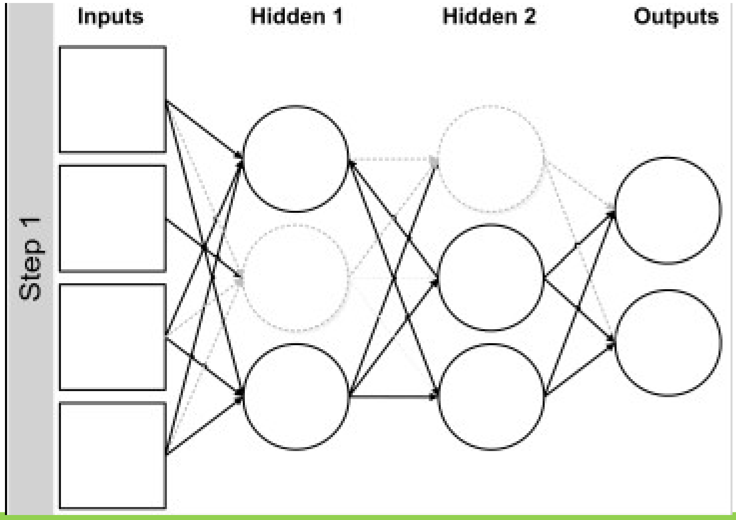
2014: VGG
- 19 layers

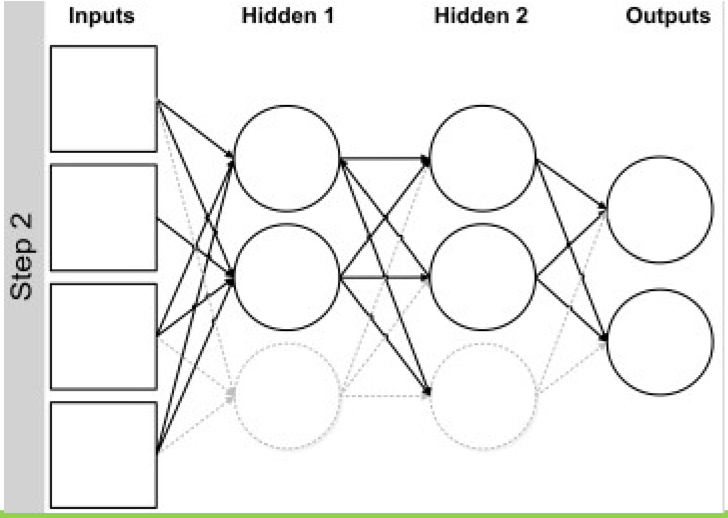
2014: GoogLeNet
- "Inception" modules
- 22 layers, 5million parameters

2015: ResNet
- 152 layers

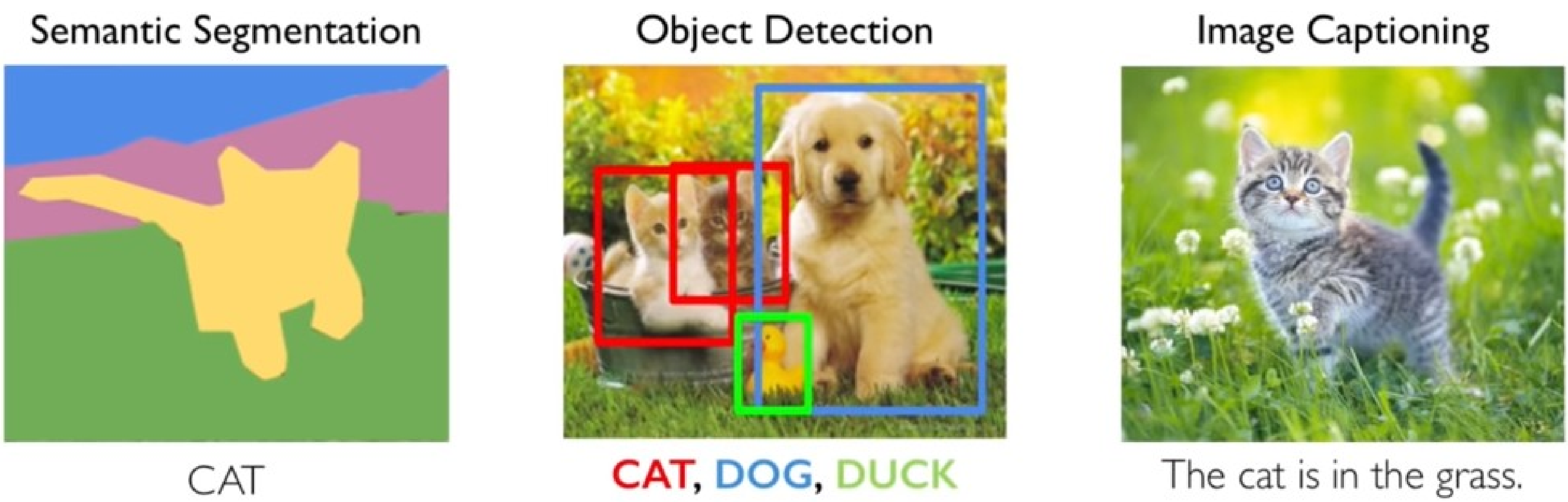
CNN: Dropout


Full Model
Inputs
Hidden 1
Hidden 2
Outputs


Step 1
Inputs
Hidden 1
Hidden 2
Outputs


Step 2
Inputs
Hidden 1
Hidden 2
Outputs

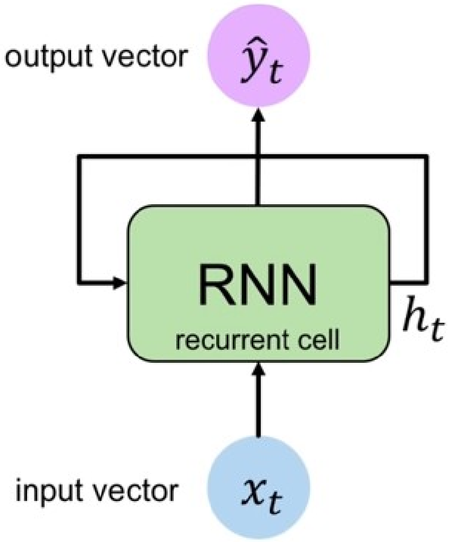
CNN: Beyond Classification



Recurrent Neural Network

- **Memory**
  - Your understanding of something is based on your understanding of previous things
- **RNN:** can be thought of as multiple copies of the same network, each passing a message to a successor

Recurrent Neural Network



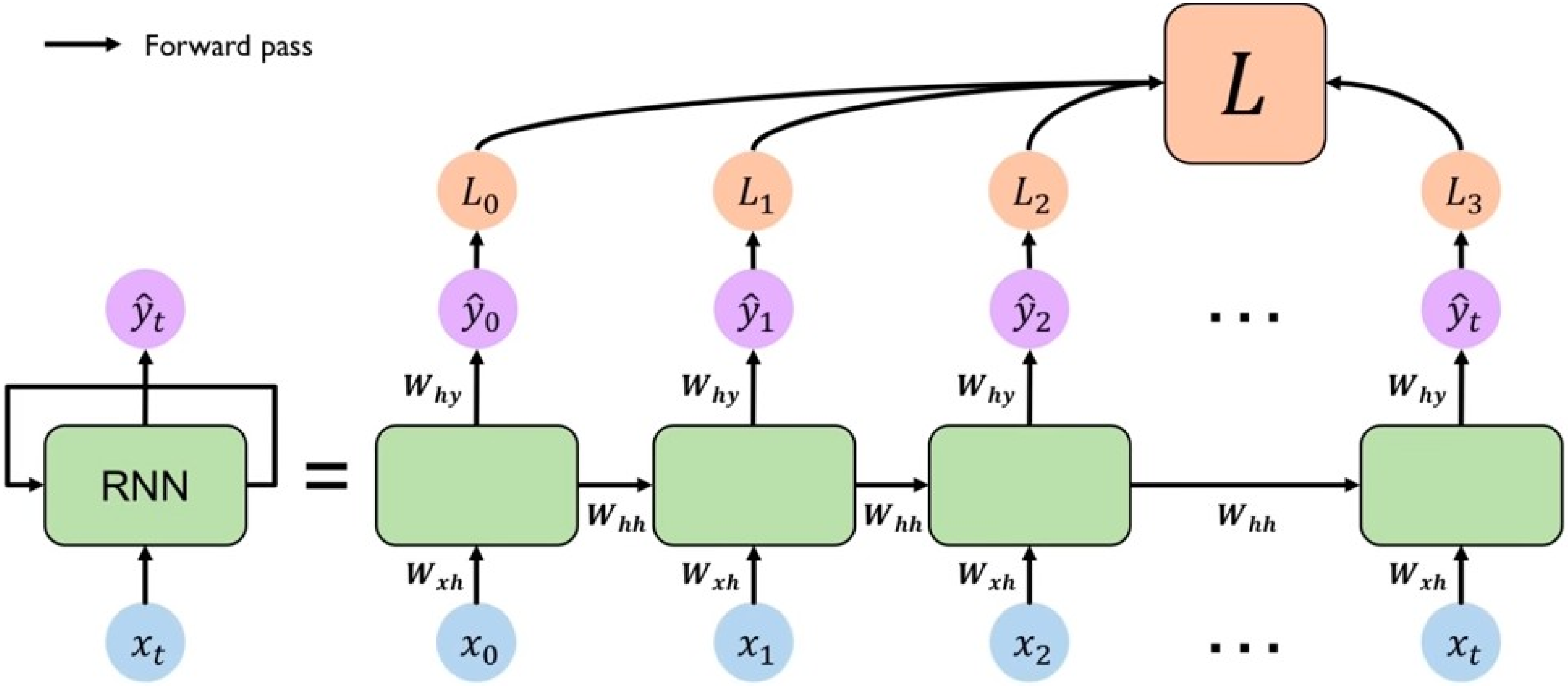
Apply a **recurrence relation** at every time step to process a sequence:

$$\underbrace{\boxed{h_t}}_{\text{cell state}} = \underbrace{\boxed{f_W}}_{\text{function parameterized by W}}(\underbrace{\boxed{h_{t-1}}}_{\text{old state}}, x_t)$$

Update Hidden State

$$h_t = \tanh(\boldsymbol{W_{hh}} h_{t-1} + \boldsymbol{W_{xh}} x_t)$$

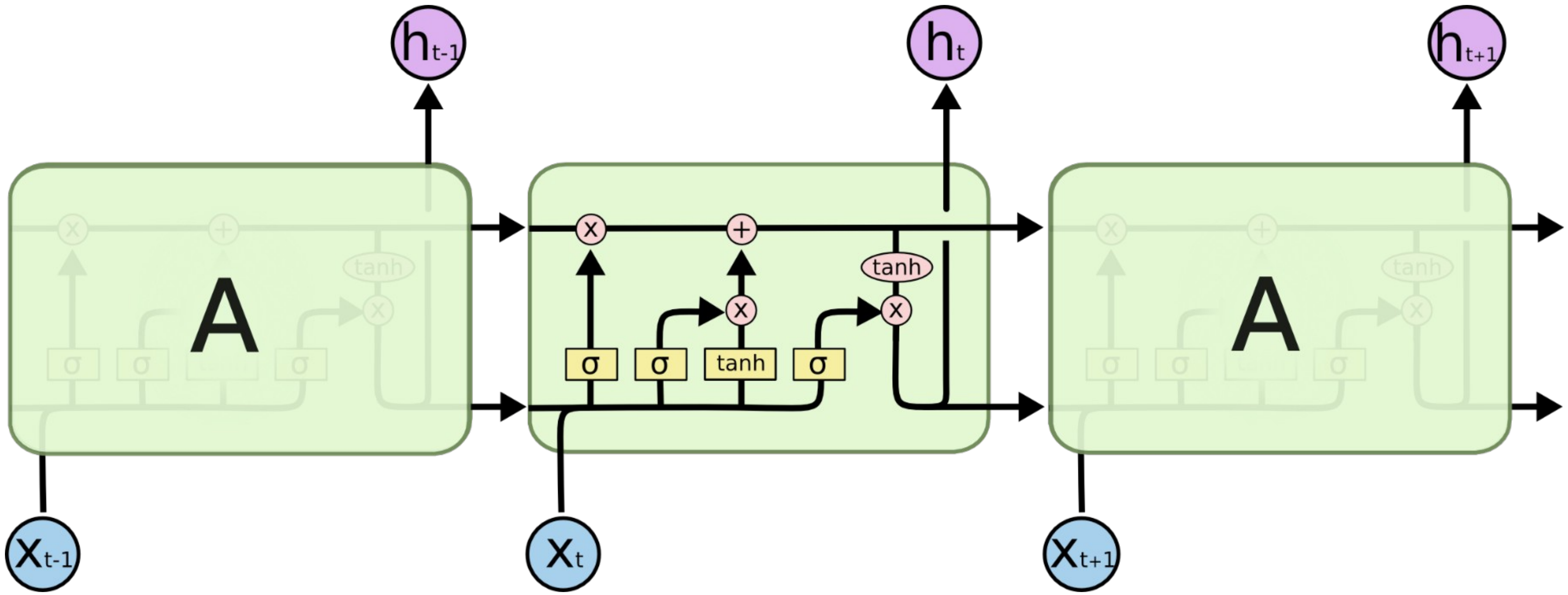
Recurrent Neural Network



LSTM: Long Short Term Memory

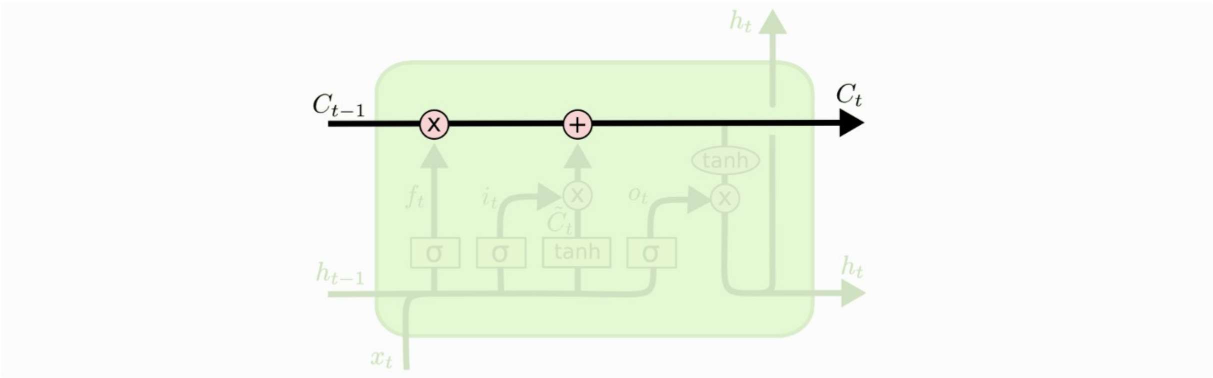
- A specific case of RNN that remember information for long periods of time.

“I grew up in France, … and I speak fluent _____.”

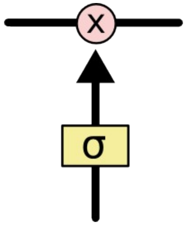


LSTM: Keywords

- **Cell State**
  - The information passing through

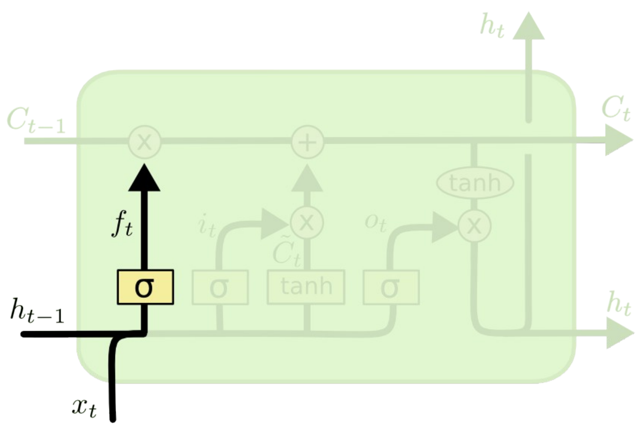


- **Gate**
  - To protect or filter information, control the information to pass through (sigmoid layer and pointwise multiplication)



LSTM: Keywords

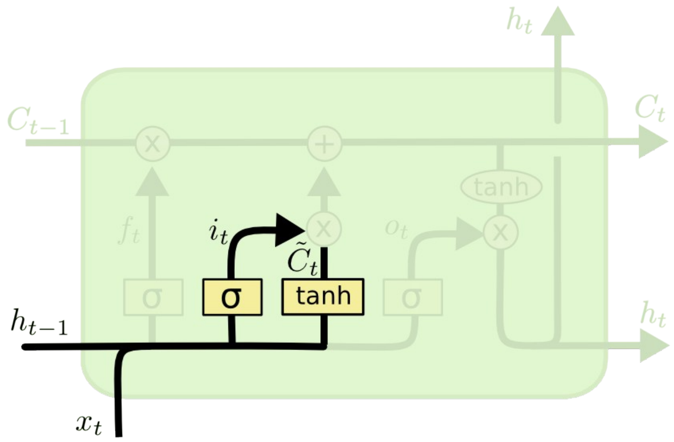
- **Step 1: Forget**
  - A **forget gate layer** decides what information we're going to throw away from the cell state, usually a *sigmoid* layer that outputs a value between 0 and 1
  - Sigmoid: outputs between 0 – completely forget vs. 1- completely keep



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

LSTM: Keywords

- **Step 2: Update**
  - Decide what information we're going to store in the cell state
  - 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 that could be added to the state.

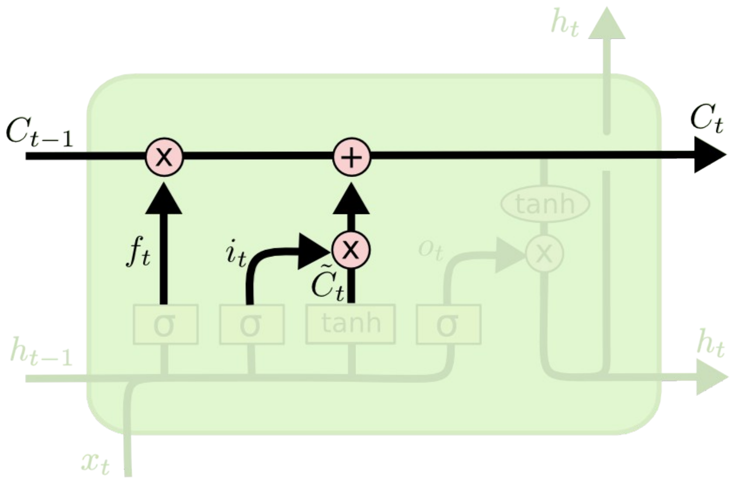


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

LSTM: Keywords

- **Step 2: Update**

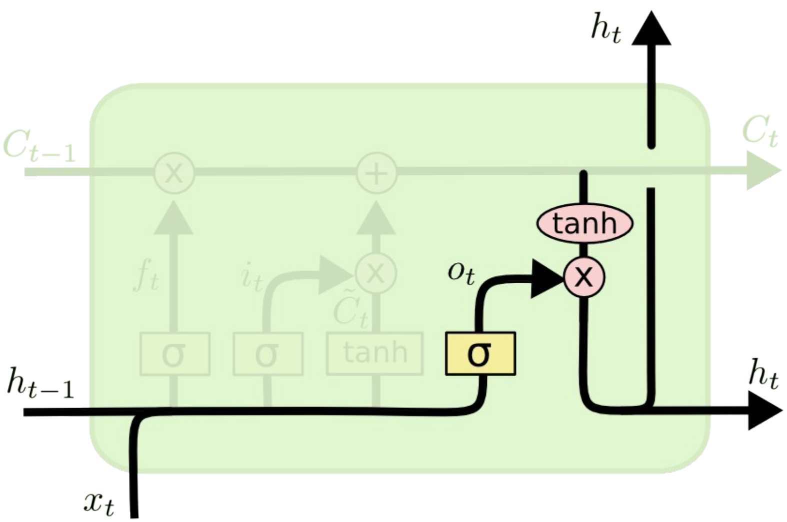


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Apply forget operation to previous internal cell state: $f_t * C_{t-1}$
- Add new candidate values, scaled by how much we decided to update: $i_t * \tilde{C}_t$

LSTM: Keywords

- **Step 3: Output**
  - Decide what we're going to output
  - Tanh: to push the values to be between −1 and 1



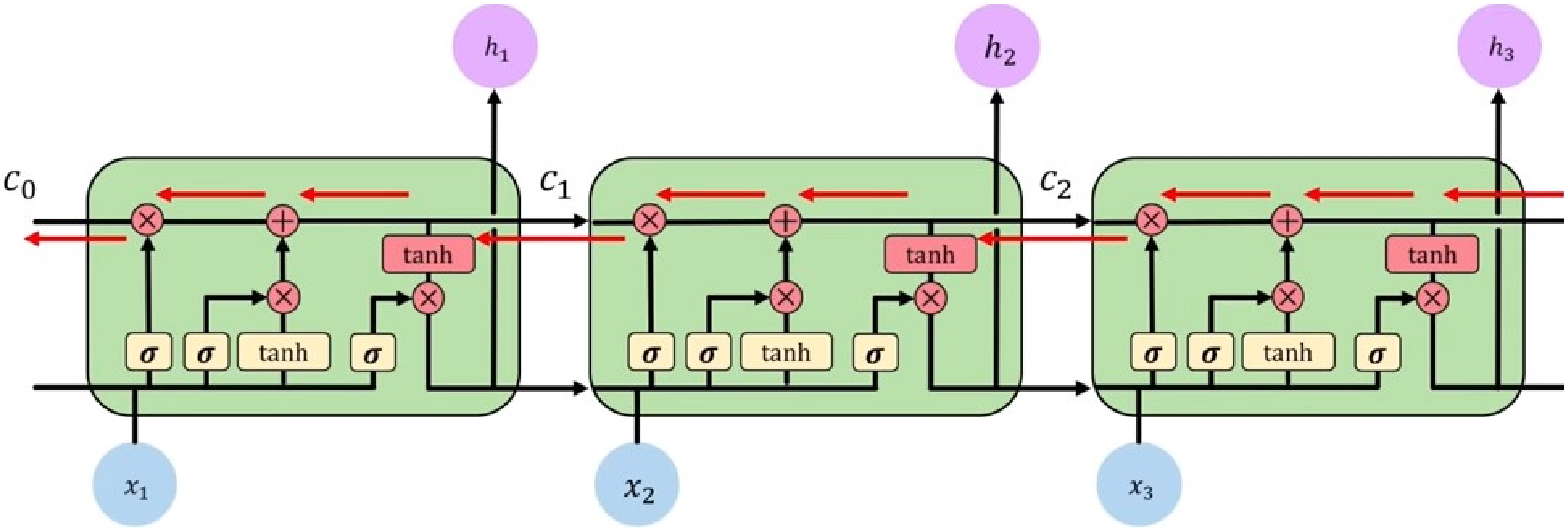
- Sigmoid layer: decide what parts of state to output

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

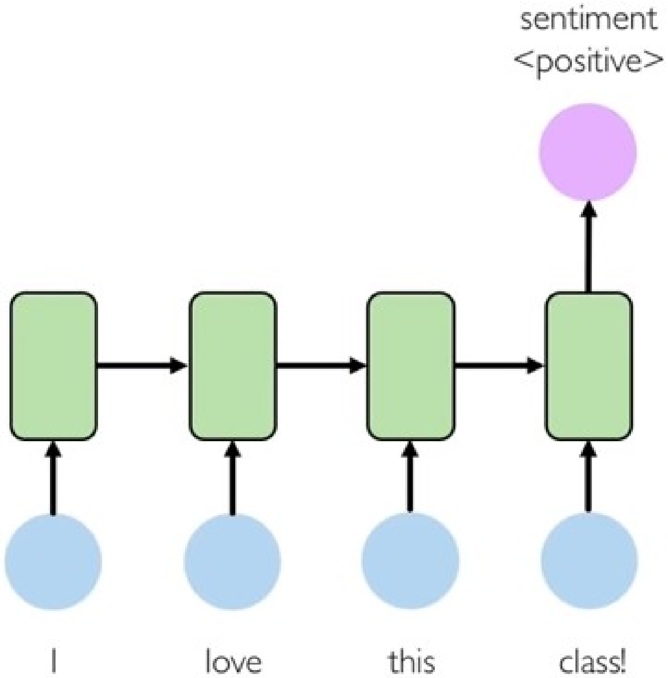
$$h_t = o_t * \tanh\left(C_t\right)$$

- Tanh layer: squash values between -1 and 1
- $o_t * \tanh(C_t)$: output filtered version of cell state

LSTM: Training



RNN: Sentiment Analysis

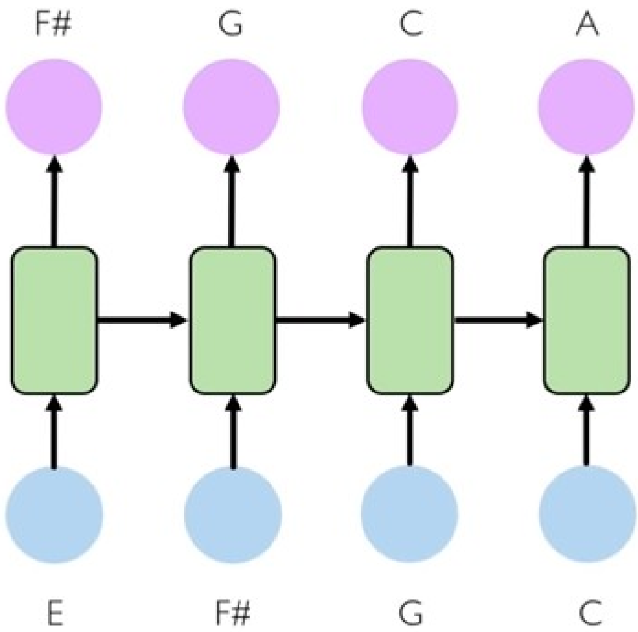


Input: sequence of words

Output: probability of having positive sentiment

```
loss = tf.nn.softmax_cross_entropy_with_logits(
    labels=model.y, logits=model.pred
)
```

RNN: Music Generation



Input: sheet music

Output: next character in sheet music

RNN: Machine Translation