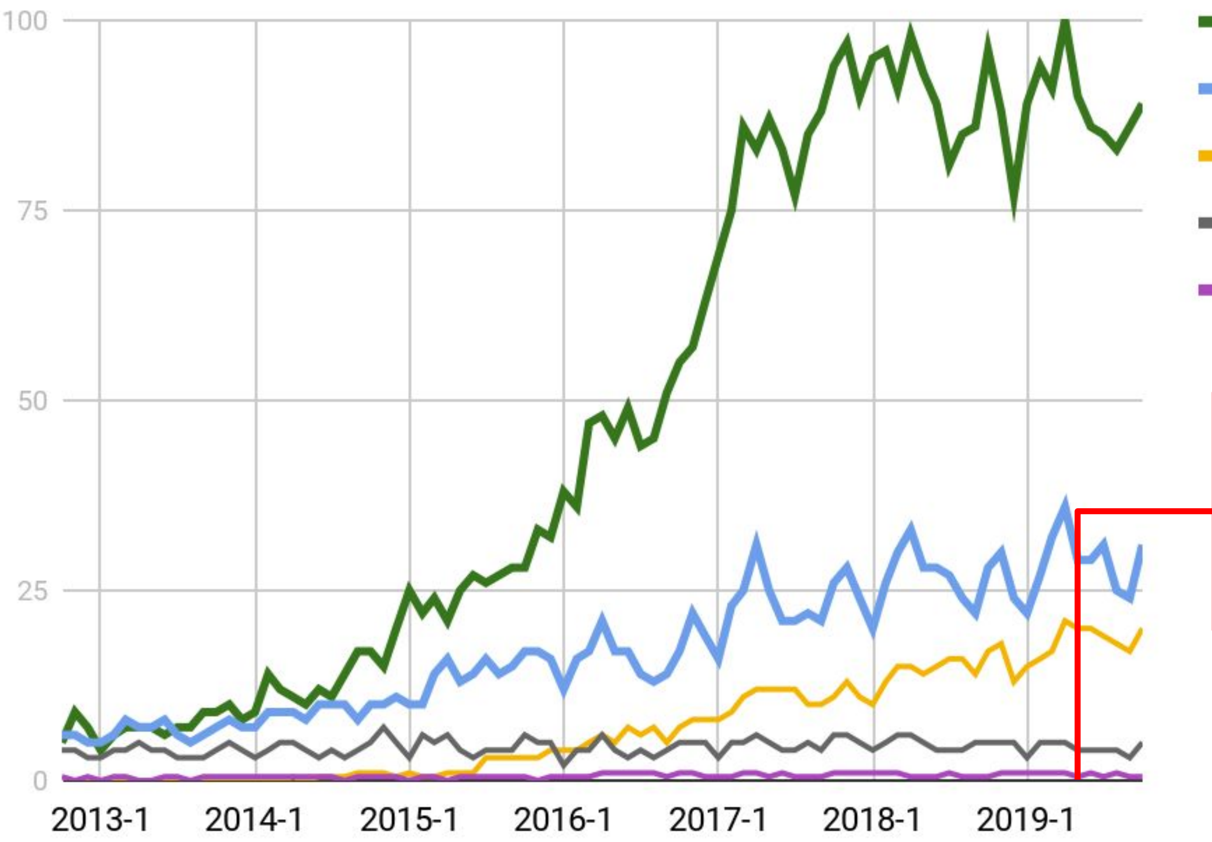
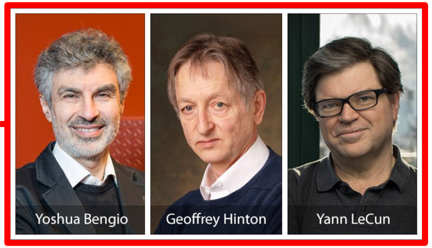
Learning Machine Learning with Kaggle Challenges

(3) Deep Learning

Qiyang Hu
IDRE

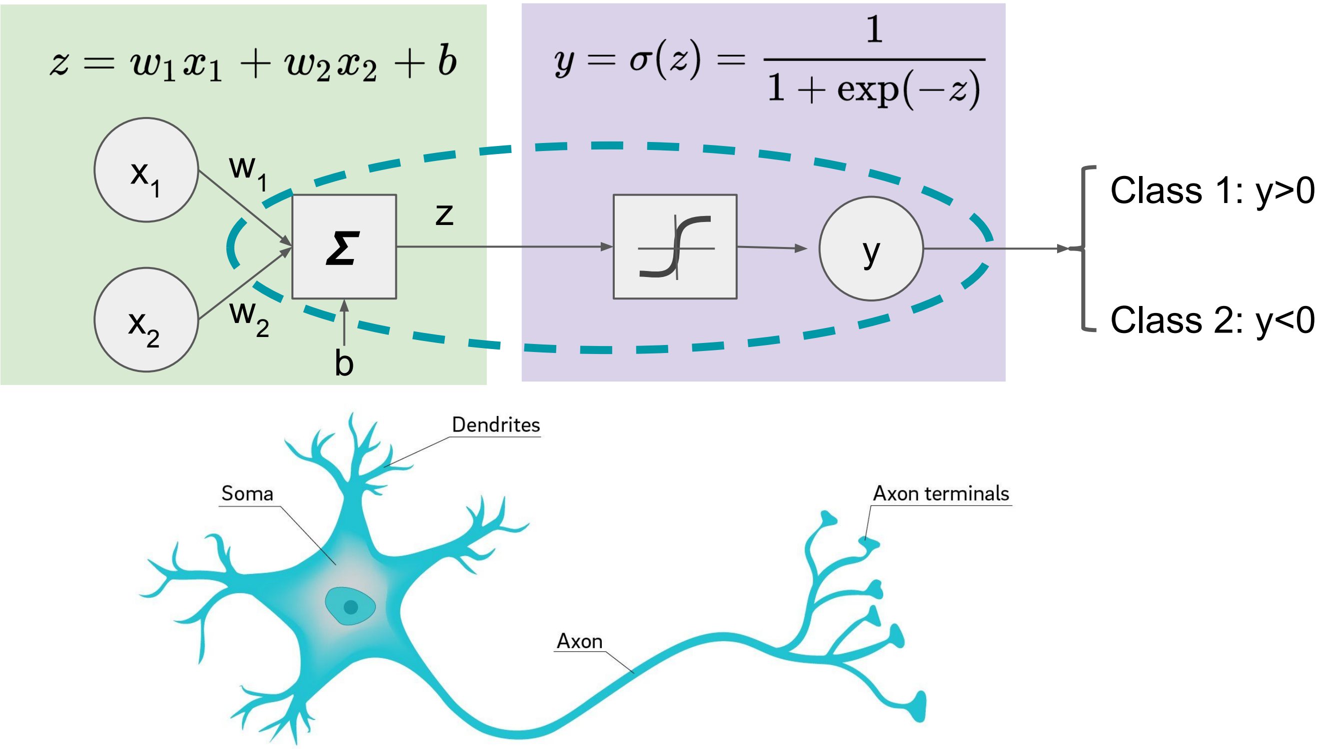
Interest over time from Google Trends



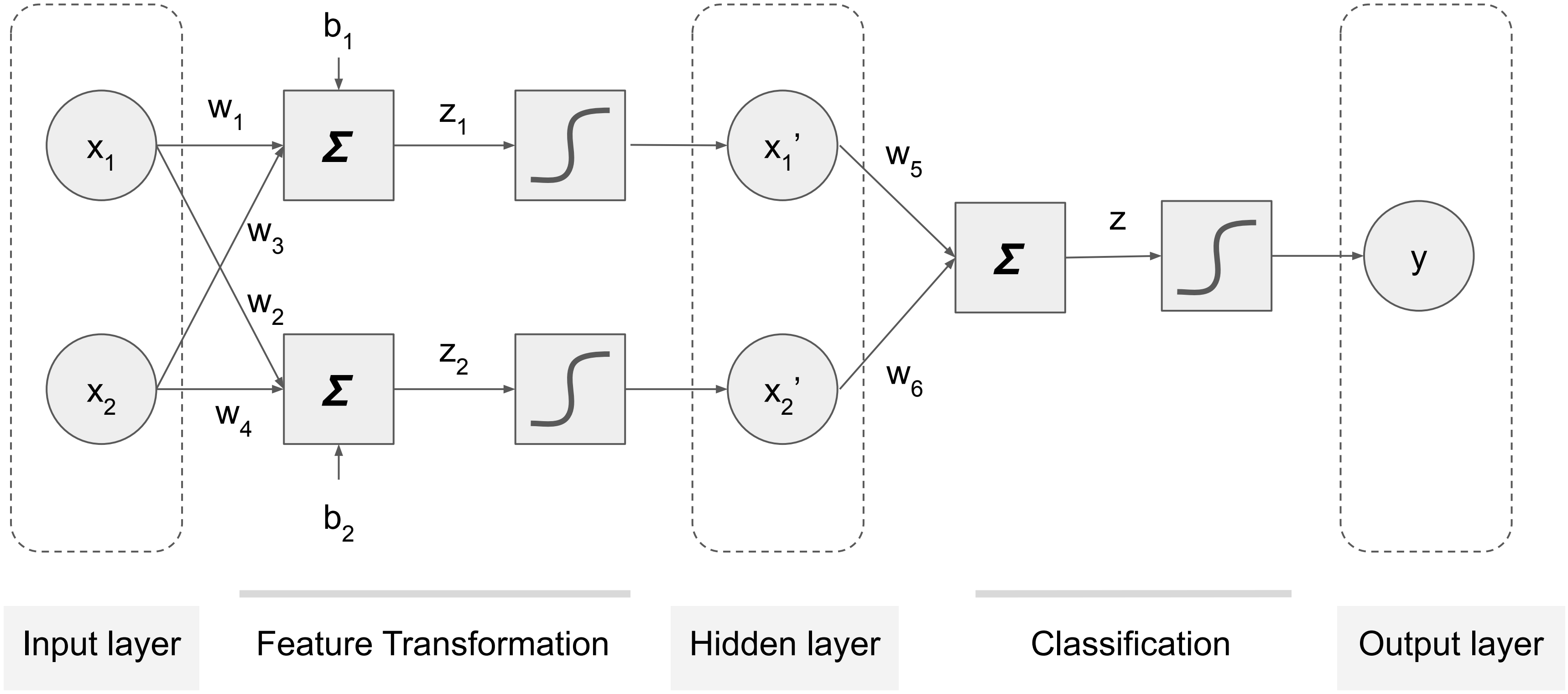


[2018 ACM Turing Award Recipients]

A logistic-regression classifier ~ one artificial neuron



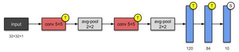
(Deep) Neural Networks ~ piling/stacking logistic-regression classifiers



Qiyang Hu

How deep a deep learning network can be?

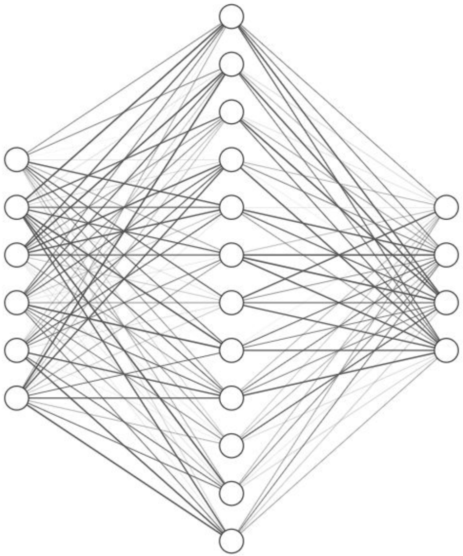
- LeNet-5 (1998)



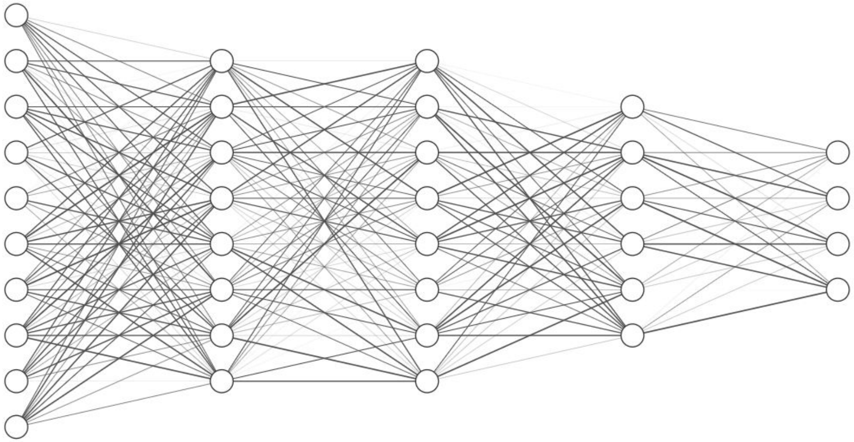
Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

Why deep?

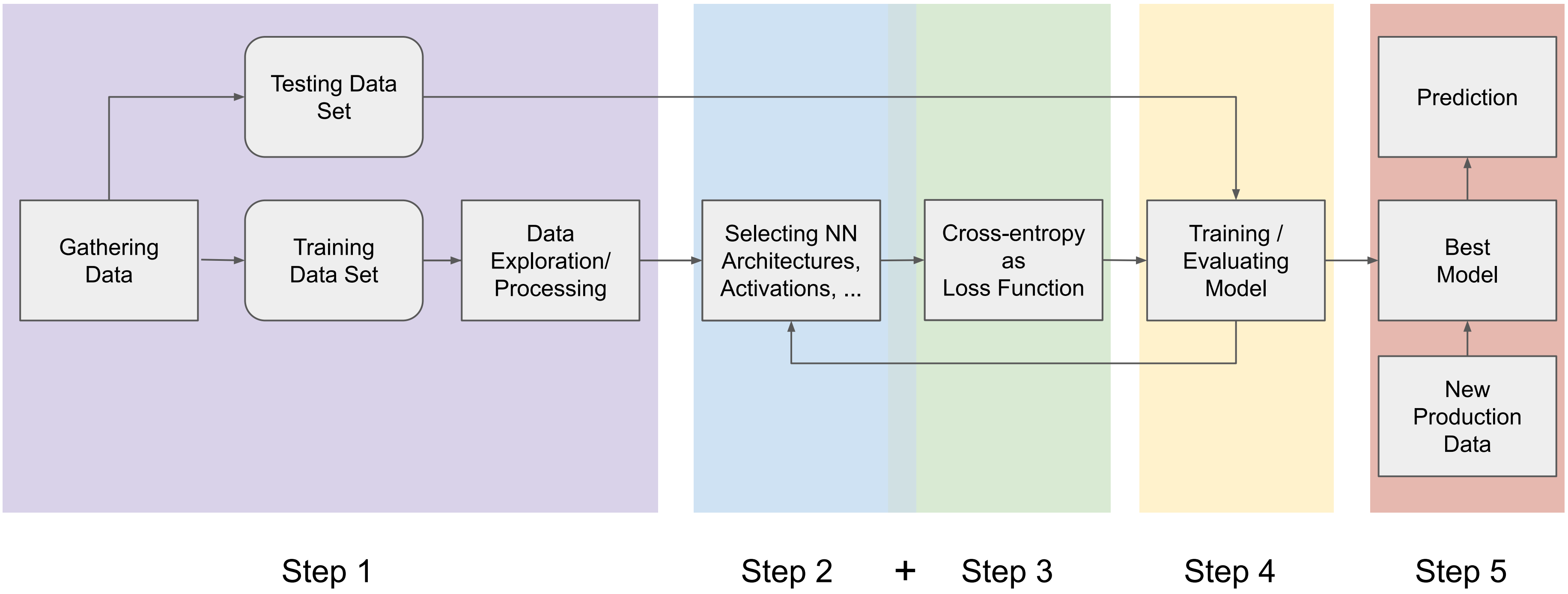
- Shallow network can fit any function
  - Has less number of hidden layers
  - Has to be really "fat"



- Deep network is more efficient.
  - It can extract/build better features
  - Exponentially fewer parameters (2017)



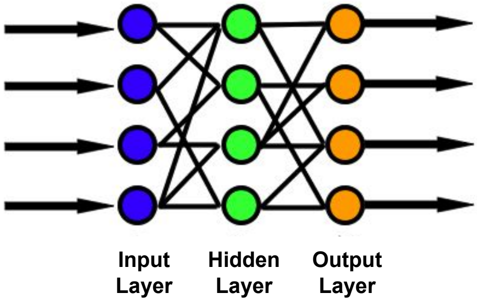
Workflow for a **deep** learning project



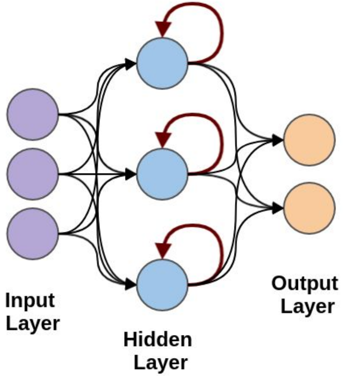
Qiyang Hu

Types of Neural Network Architectures

- Feed forward neural networks (No cycle in node connections)
  - Perceptron
  - Fully connected network
  - Convolutional networks

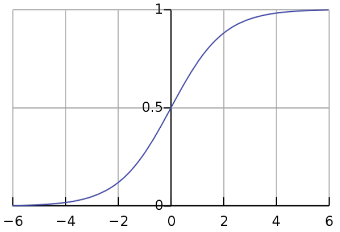


- Recurrent networks (w/ directed cycle in node connections)
  - Fully recurrent NN
  - Recursive NN
  - Long short-term memory (LSTM)
  - Symmetrically connected networks
    - Hopfield network (w/o hidden nodes)
    - Deep Bolzmann Machine (w/ hidden nodes)



Activation Function

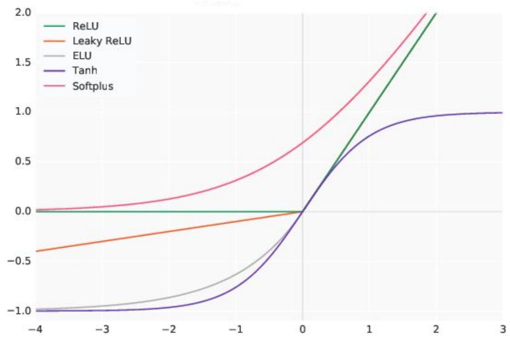
- Sigmoid function: $$\sigma(z) = \frac{1}{1+\exp(-z)}$$



- Rectified linear unit (ReLU)

$$f(x) = x^+ = \max(0, x)$$

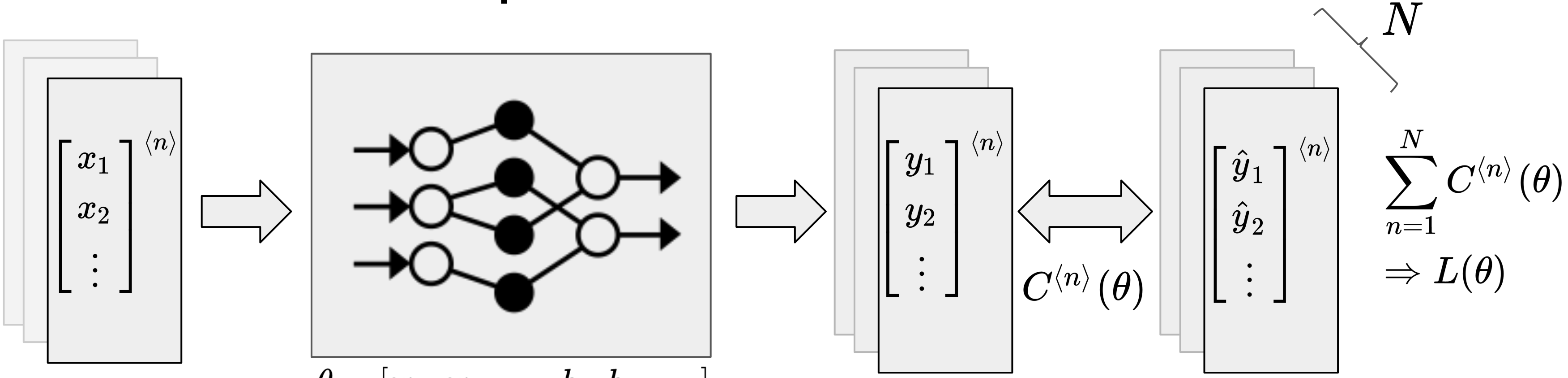
  - Softplus
  - Leaky ReLU
  - Exponential LU (ELUs)



- Softmax function: $$y_i = \frac{e^{z^{(i)}}}{\sum_{j=0}^{K} e^{z^{(j)}}}$$

- Maxout Network:
  - *Learnable* activation function

How to train a deep neural network?



$$\theta = [w_1, w_2, \ldots, b_1, b_2, \ldots]$$

$$\nabla L(\theta) = \sum_{n=1}^{N} \begin{bmatrix} \frac{\partial C^{\langle n \rangle}(\theta)}{\partial w_1} \\ \frac{\partial C^{\langle n \rangle}(\theta)}{\partial w_2} \\ \vdots \\ \frac{\partial C^{\langle n \rangle}(\theta)}{\partial b_1} \\ \vdots \end{bmatrix}$$

$$\theta_0 \to \nabla L(\theta_0) \to \theta_1 \to \nabla L(\theta_1) \to \theta_2 \to \cdots$$

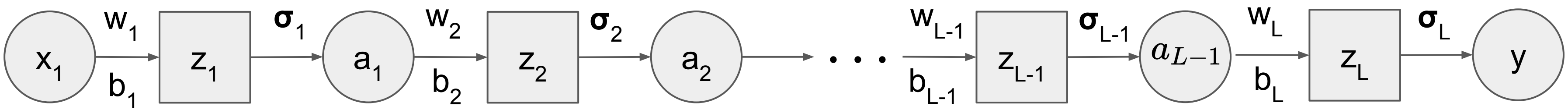
$$\theta_1 = \theta_0 - \eta \nabla L(\theta_0)$$

$$\theta_2 = \theta_1 - \eta \nabla L(\theta_1)$$

$$\vdots$$

Qiyang Hu

Backpropagation: a game of chain rule



$$y = \sigma_L\Big(w_L \cdot \sigma_{L-1}\big(\cdots w_2 \cdot \underbrace{\sigma_1(\underbrace{w_1 \cdot x + b_1}_{z_1})}_{a_1} + b_2\big) + b_L\Big)$$

$$\frac{\partial C\big(y(w)-\hat{y}\big)}{\partial w} = \frac{\partial z}{\partial w}\frac{\partial C}{\partial z} = \frac{\partial z}{\partial w}\left[\frac{\partial a}{\partial z}\frac{\partial C}{\partial a}\right] = \frac{\partial z}{\partial w}\left[\sigma' \cdot \left(\frac{\partial z_{(+1)}}{\partial a}\frac{\partial C}{\partial z_{(+1)}}\right)\right]$$

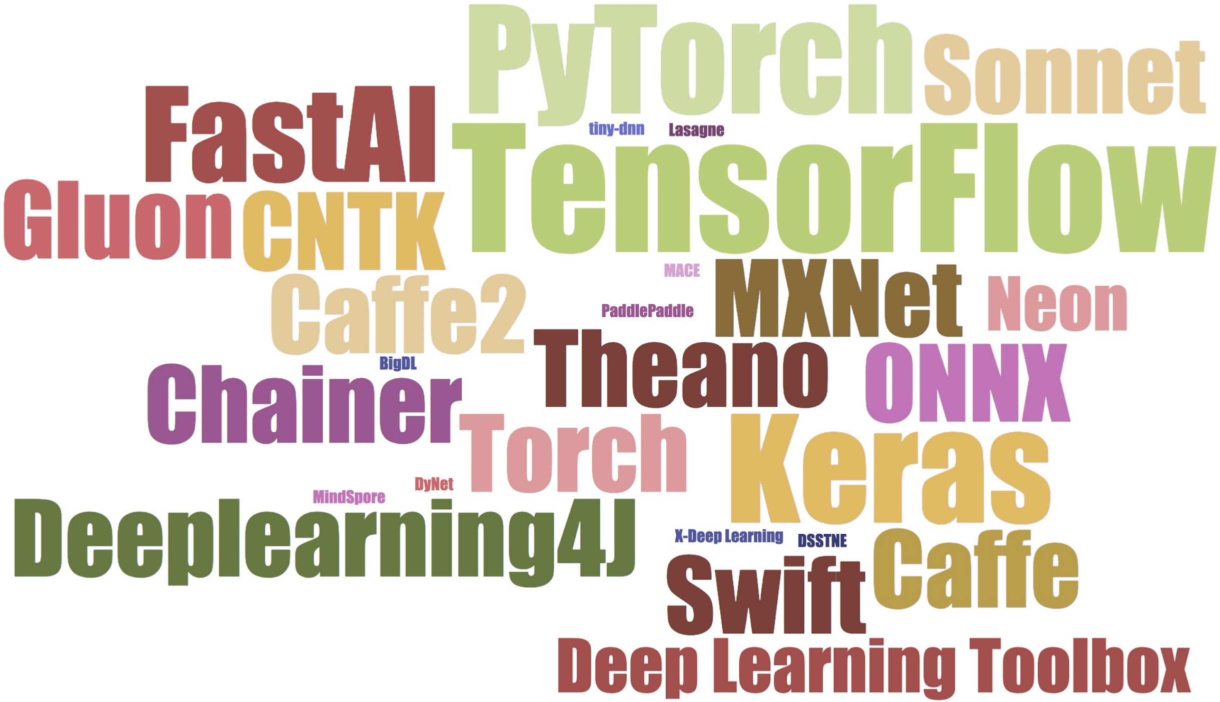
① Forward Pass

$$\frac{\partial z_1}{\partial w_1} = x_1 \longrightarrow \frac{\partial z_2}{\partial w_2} = a_1 \longrightarrow \cdots \longrightarrow \frac{\partial z_{L-1}}{\partial w_{L-1}} = a_{L-2} \longrightarrow \frac{\partial z_L}{\partial w_L} = a_{L-1}$$

② Backward Pass

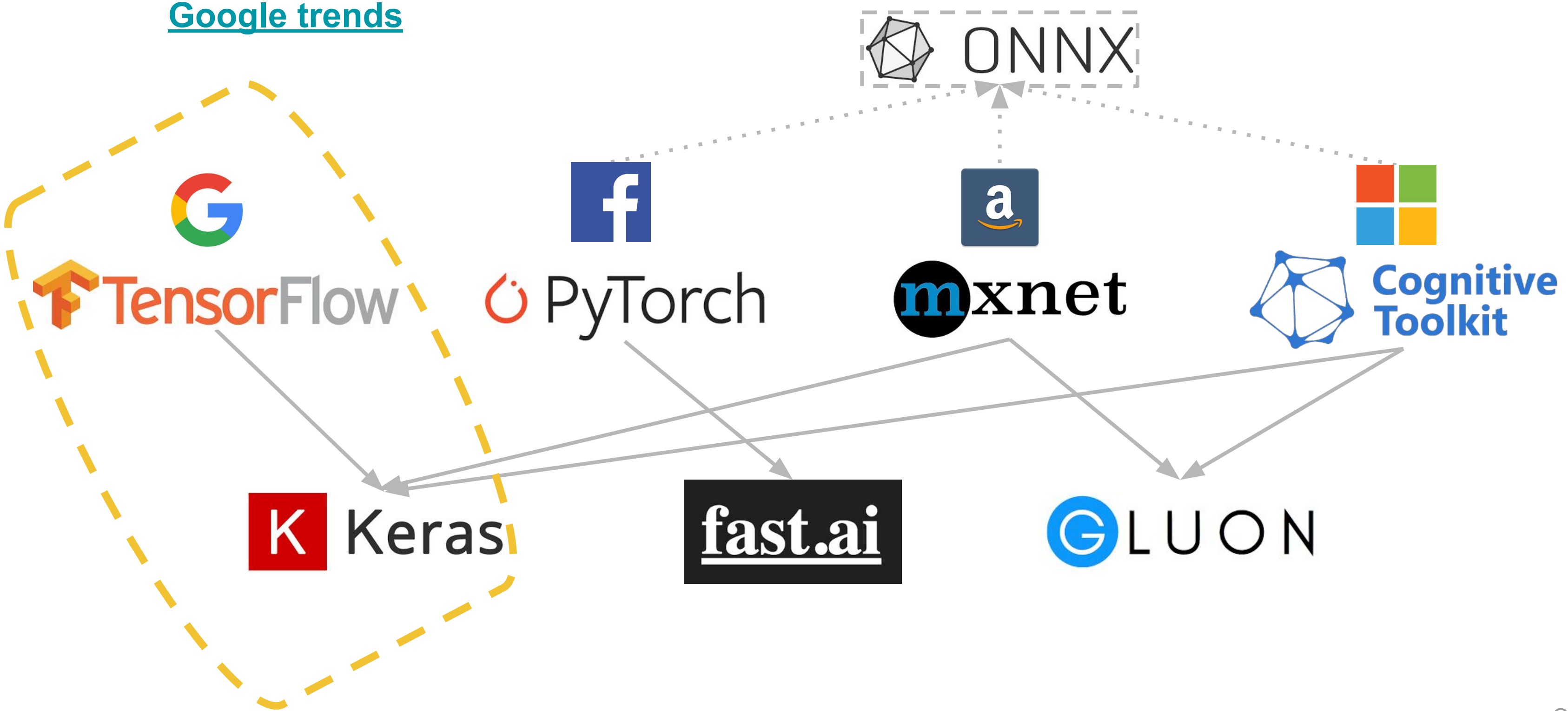
$$\frac{\partial C}{\partial z_1} = \sigma'_1\left[w_2\frac{\partial C}{\partial z_2}\right] \longleftarrow \cdots \longleftarrow \frac{\partial C}{\partial z_{L-1}} = \sigma'_{L-1}\left[w_L\frac{\partial C}{\partial z_L}\right] \longleftarrow \frac{\partial C}{\partial z_L} = \sigma'_L\frac{\partial C}{\partial y} \longleftarrow \frac{\partial C}{\partial y}$$

Deep learning frameworks



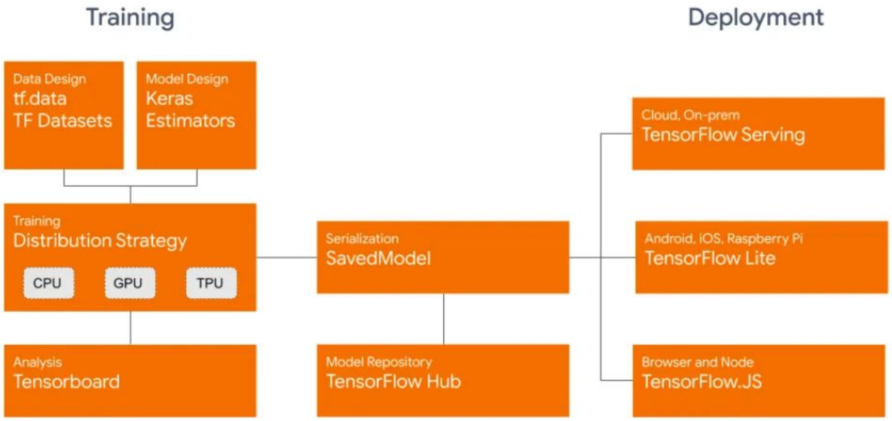
Mainstream Players

Google trends



Tensorflow v2.0 just out on Sep. 30, 2019

- ***BIG*** Changes from v1.x
  - Keras as a core API
  - Eager execution by default
  - tf.function decorator to speed up
  - tf.data to build complex input pipelines
  - Model deployment to various platforms



- Cautions
  - Poor compatibility with v1.x
  - Lots of confusions from API names

- Importing tf2.0 in standalone python
  - ```
    import tensorflow.compat.v2 as tf
    tf.enable_v2_behavior()
    ```

- Importing tf2.0 in Colab
  - ```
    %tensorflow_version 2.x
    ```

Using tf.keras (an 1-neuron 1-layer example)

Load the data

```
celsius_q    = np.array([-40, -10,  0,  8, 15, 22,  38],  dtype=float)
fahrenheit_a = np.array([-40,  14, 32, 46, 59, 72, 100],  dtype=float)
```

Build the model

```
model = tf.keras.Sequential([
                    tf.keras.layers.Dense(units=1,input_shape=[1]) ])
```

Compile the model

```
model.compile( loss='mean_squared_error',
               optimizer=tf.keras.optimizers.Adam(0.1) )
```

Train the model

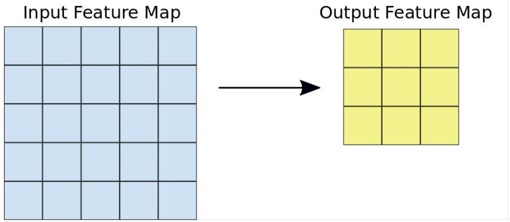
```
model.fit( celsius_q, fahrenheit_a, epochs=500, verbose=False )
```

Predict the model

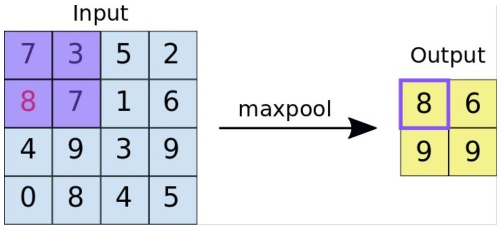
```
model.predict( [100.0] )
```

Convolutional Neural Networks (CNNs)

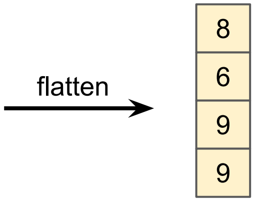
- A special network architecture to reduce parameters
- 3 processes in CNNs:
  - Convolutions to extract as tiles



  - Poolings to downsample

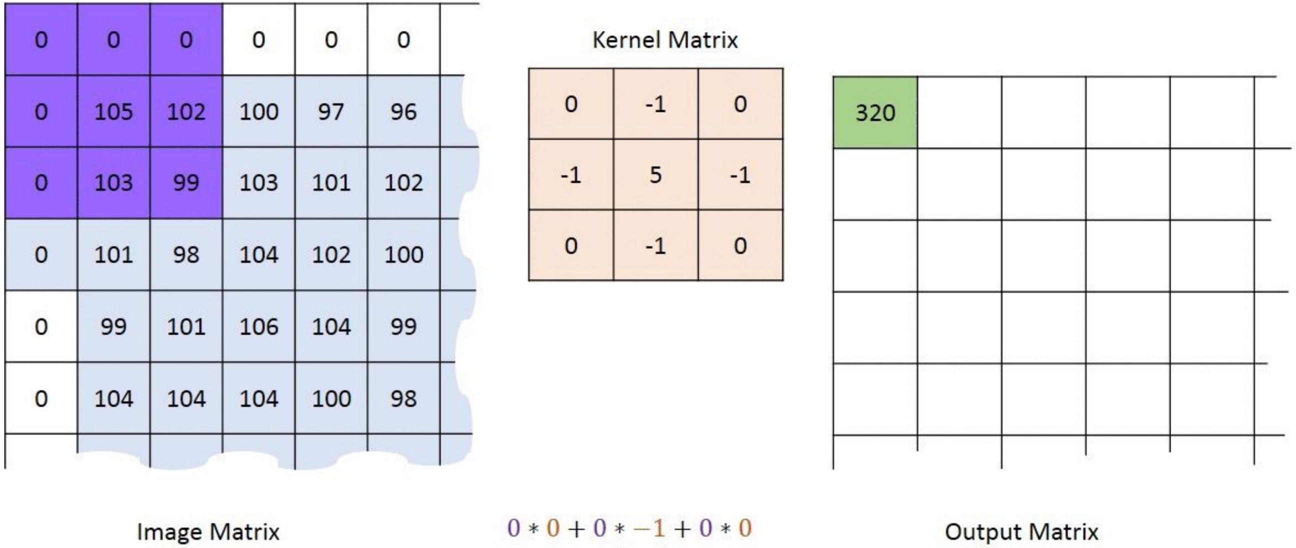


  - Flattening



- Logic behind CNNs
  - Sparse connectivity (characteristic features in smaller local regions)
  - Parameter equivariance & sharing (features appear in different locations)
  - Translation invariance (some sampling will not lose main information)

One Channel, One Filter



$$
\begin{aligned}
&0 * 0 + 0 * -1 + 0 * 0 \\
&+0 * -1 + 105 * 5 + 102 * -1 \\
&+0 * 0 + 103 * -1 + 99 * 0 \quad = 320
\end{aligned}
$$

Convolution with horizontal and vertical strides = 1, with 'same' padding

Multiple Channels

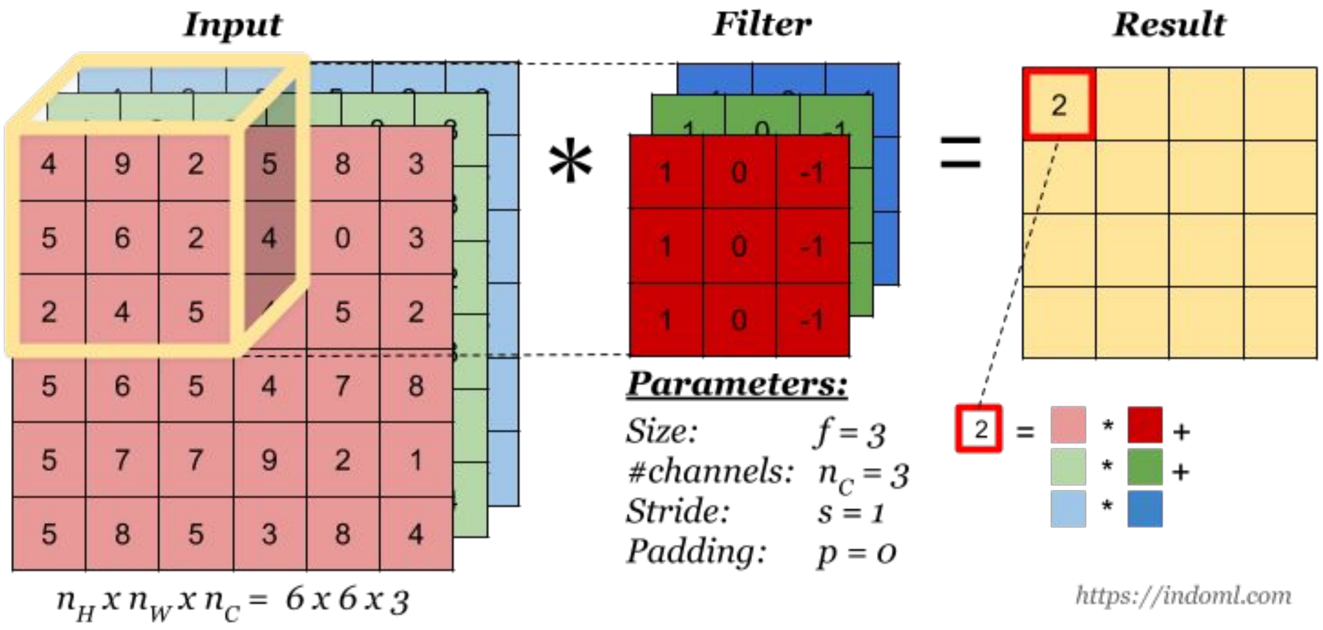
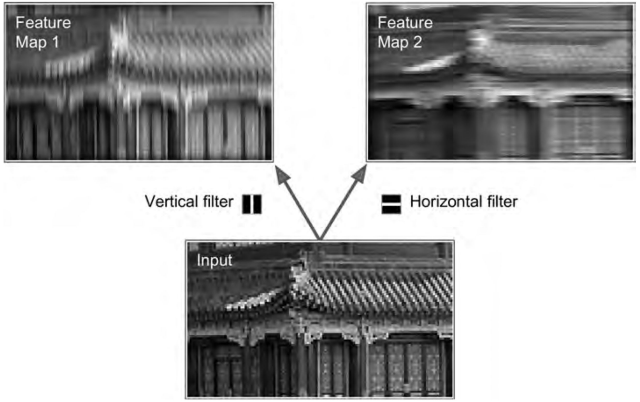
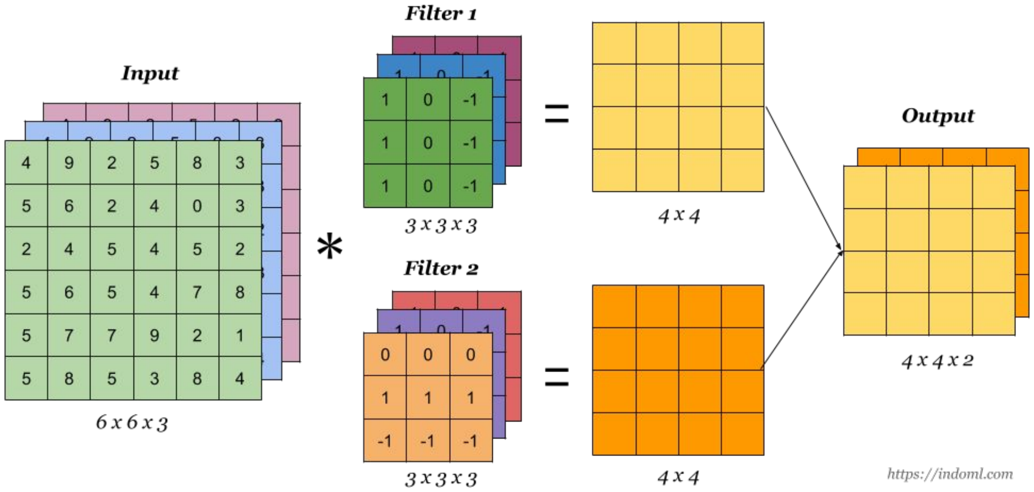


Figure Source

Multiple Filters





Figures from Aurélien Géron's [1st Ed. Book](#)

Figure [Source](#)

A Convolutional layer

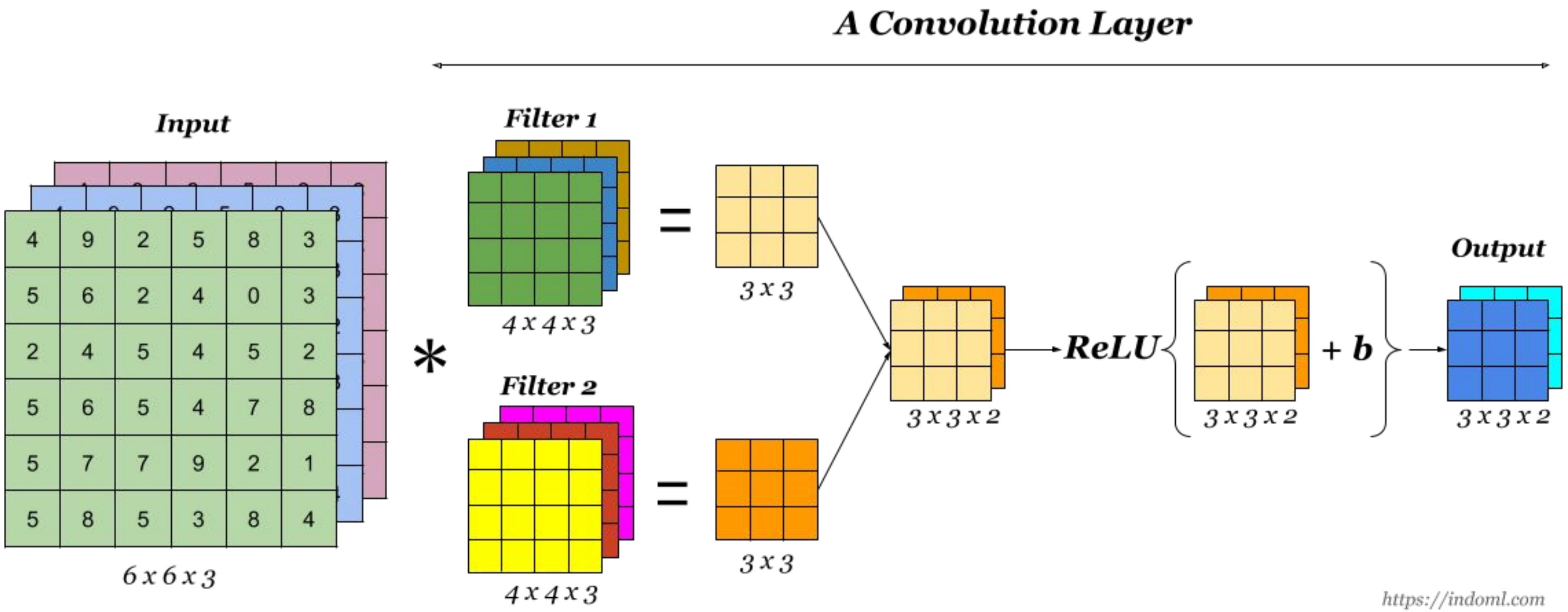


Figure Source

Pooling Layer

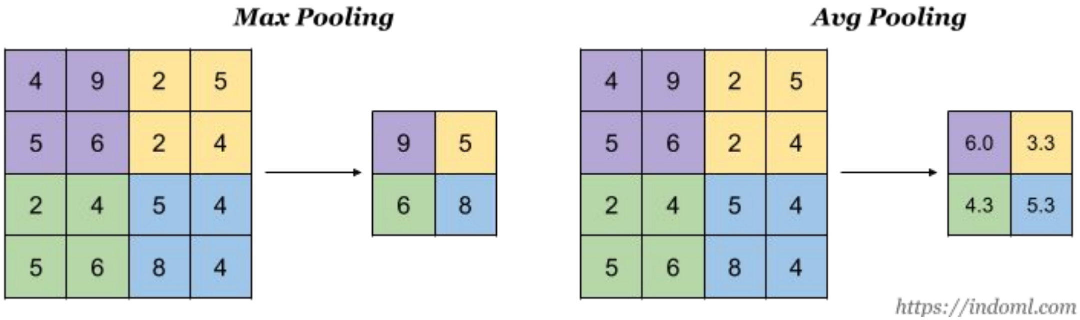
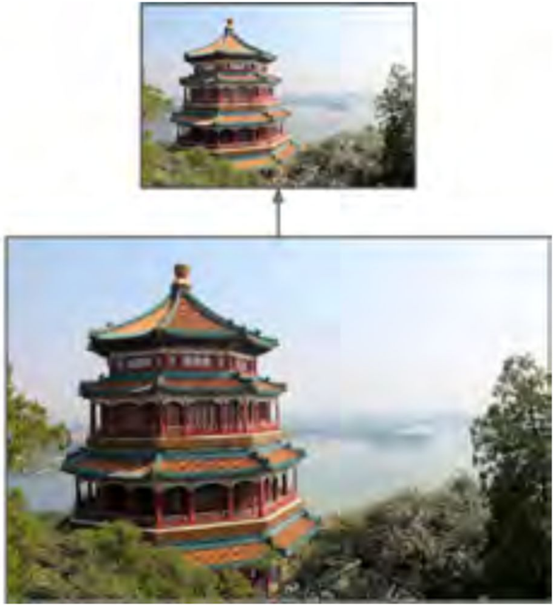


Figure Source



Figures from Aurélien Géron’s 1st Ed. Book

- Assuming downsampling will not lose the major information.

Architecture of Convolutional Neural Networks

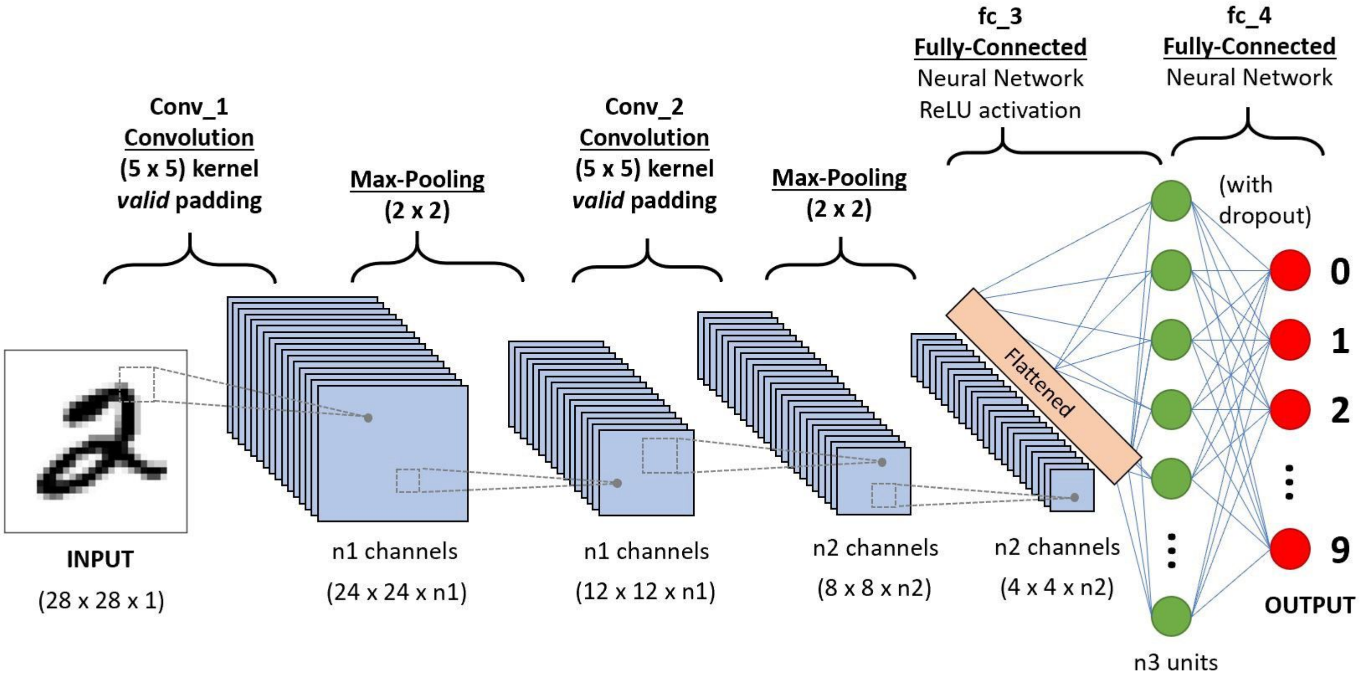


Figure [Source]

Qiyang Hu

Dogs vs. Cats Kaggle Challenge

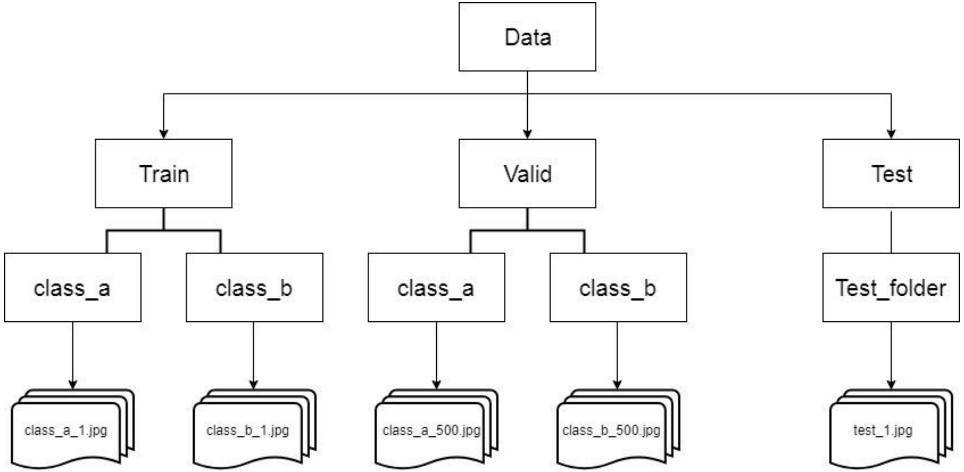
- Redux: Kernels Edition
  - Submission scored by the probability of dogs using log loss

$$L = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

- Dataset
  - Training set: 25,000 dogs and cats images
  - Testing set: 12,500 images
- Two concerns:
  - Images with different sizes
    - Neural network needs fixed sized input.
    - We will resize images to 150x150 pixels
  - Images are colored
    - Represented by Red-Green-Blue channels
    - One image ⇒ 150x150x3 matrices

Import image data using tf.keras

- tf.keras.preprocessing.image.ImageDataGenerator.flow_from_directory




```
train_img_gen  = ImageDataGenerator(rescale=1./255)
train_data_gen = train_img_gen.flow_from_directory(batch_size=BATCH_SIZE,
                                                   directory=train_dir,
                                                   shuffle=True,
                                                   target_size=(IMG_SHAPE,IMG_SHAPE), #(150,150)
                                                   class_mode='binary')
```

Construct CNN architecture using tf.keras

- Convolution layer:
  [tf.keras.layers.Conv2D(filters, kernel_size, activation, ...)]
  - Filters (feature maps): 32 or 64 or 128 …
  - Kernel_size: (3,3)
  - Activation function: 'relu'
  - Input_shape: (150, 150, 3)
- MaxPooling layer:
  [tf.keras.layers.MaxPooling2D(pool_size, strides, …)]
  - Pool_size: 2
  - Strides: 2
- Flattened layer
  [tf.keras.layers.Flatten()]
- Dense layer
  [tf.keras.layers.Dense(units, activation, …)]
  - Units: 512 and 2
  - Activation: 'relu' and 'softmax'

4 Conv blocks with maxpool in each

```
model = tf.keras.models.Sequential([                                  (148,148,32)
    tf.keras.layers.Conv2D(32, (3,3), activation='relu',
                           input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),                               (74,74,32)

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),                                (72,72,64)

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),                                (7,7,128)

    tf.keras.layers.Flatten(),                                        6272
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')
])
```

Don’t forget to

- Sign in your info to the class
  - To get the email notifications
- Contact me for questions or discussions
  - huqy@idre.ucla.edu
  - Office: Math Sci #3330
  - Phone: 310-825-2011
- Fill out the survey for comments:
  - https://forms.gle/t3f8CztFQpeFFksy6