GNOMON DIGITAL

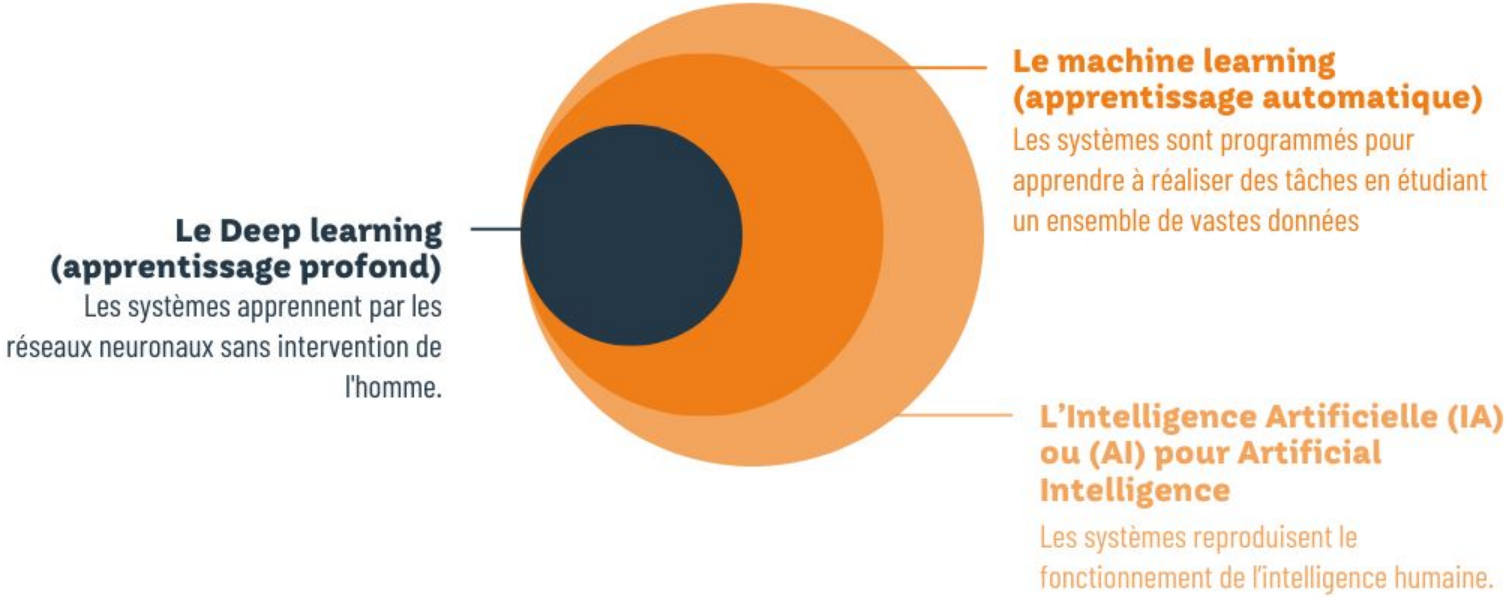
Architecture Deep Learning

Jiqiong QIU

Objectif du cours

- Le but de ce cours est d'apprendre les architectures des modèles deep learning
- Le cours se focalise essentiellement sur les aspects suivants :
  - *Introduction to Deep learning*
  - Typology des méthodologies
  - Structure Classique: MLP, CNN, RNN, Self-Attention

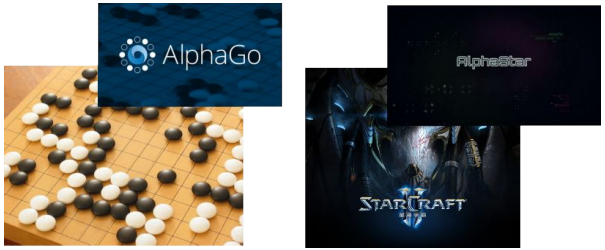
Qu’est-ce que c’est l’intelligence Artificielle



Deep learning

- Pour le moment, un modèle peut souvent résoudre une type de problème

On est la!

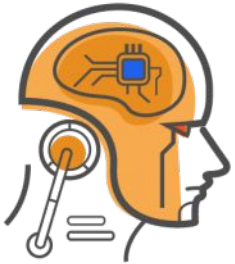


Narrow AI

Dedicated to assist with or take over specific tasks.

General AI

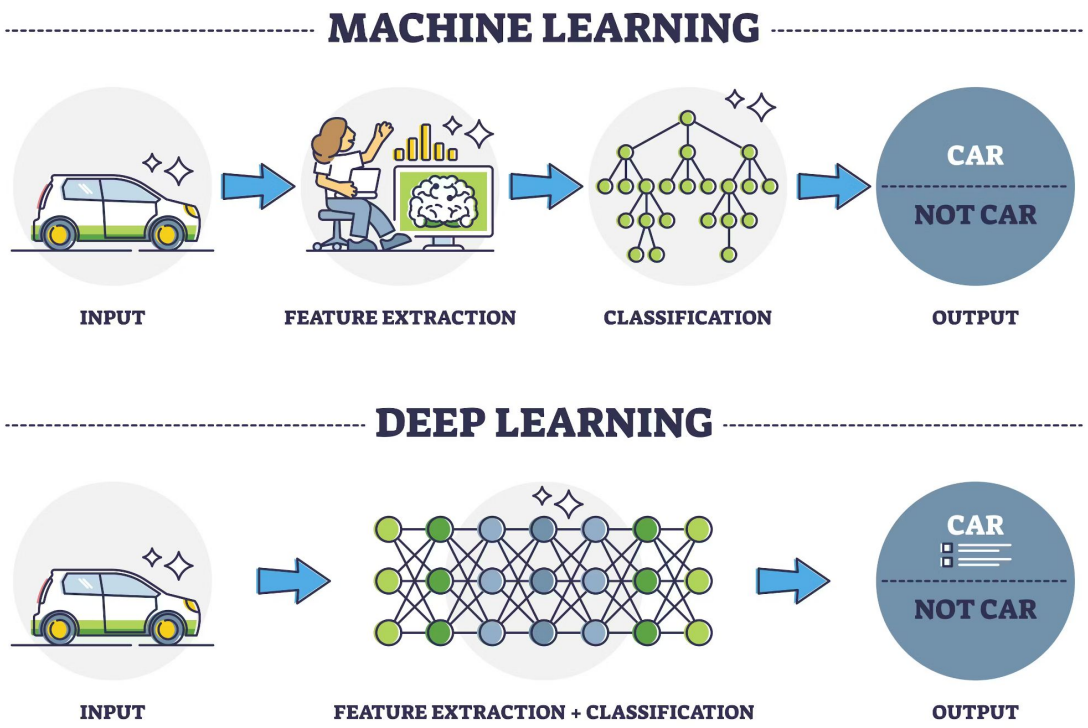
Takes knowledge from one domain, transfers to other domain.



Super AI

Machines that are an order of magnitude smarter than humans.

Quelle est la différence entre DL et ML



Quelle est la différence entre DL et ML

GNOMON DIGITAL®

Machine learning vs. deep learning

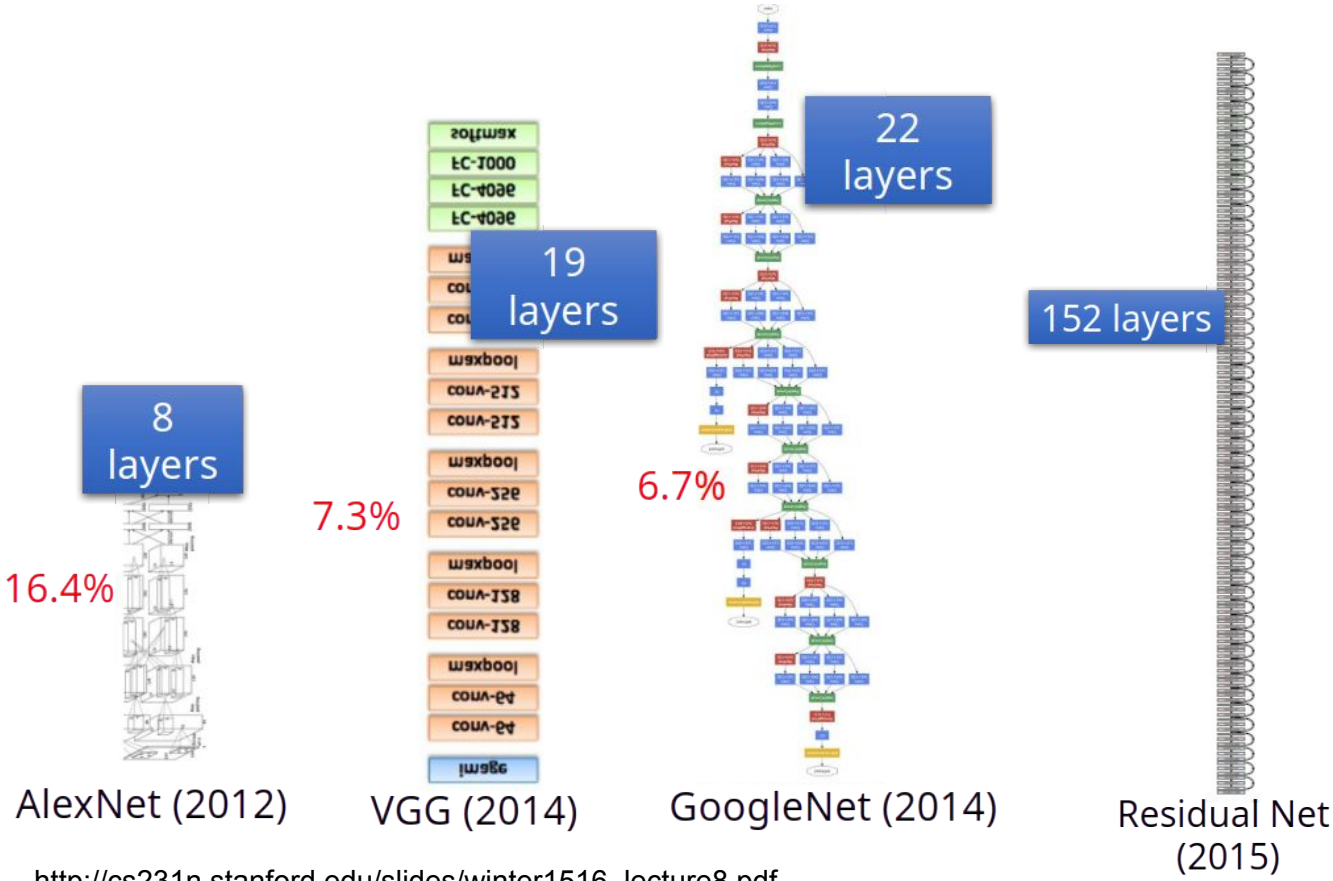
Machine learning

Uses algorithms and learns on its own but may need human intervention to correct errors

Deep learning

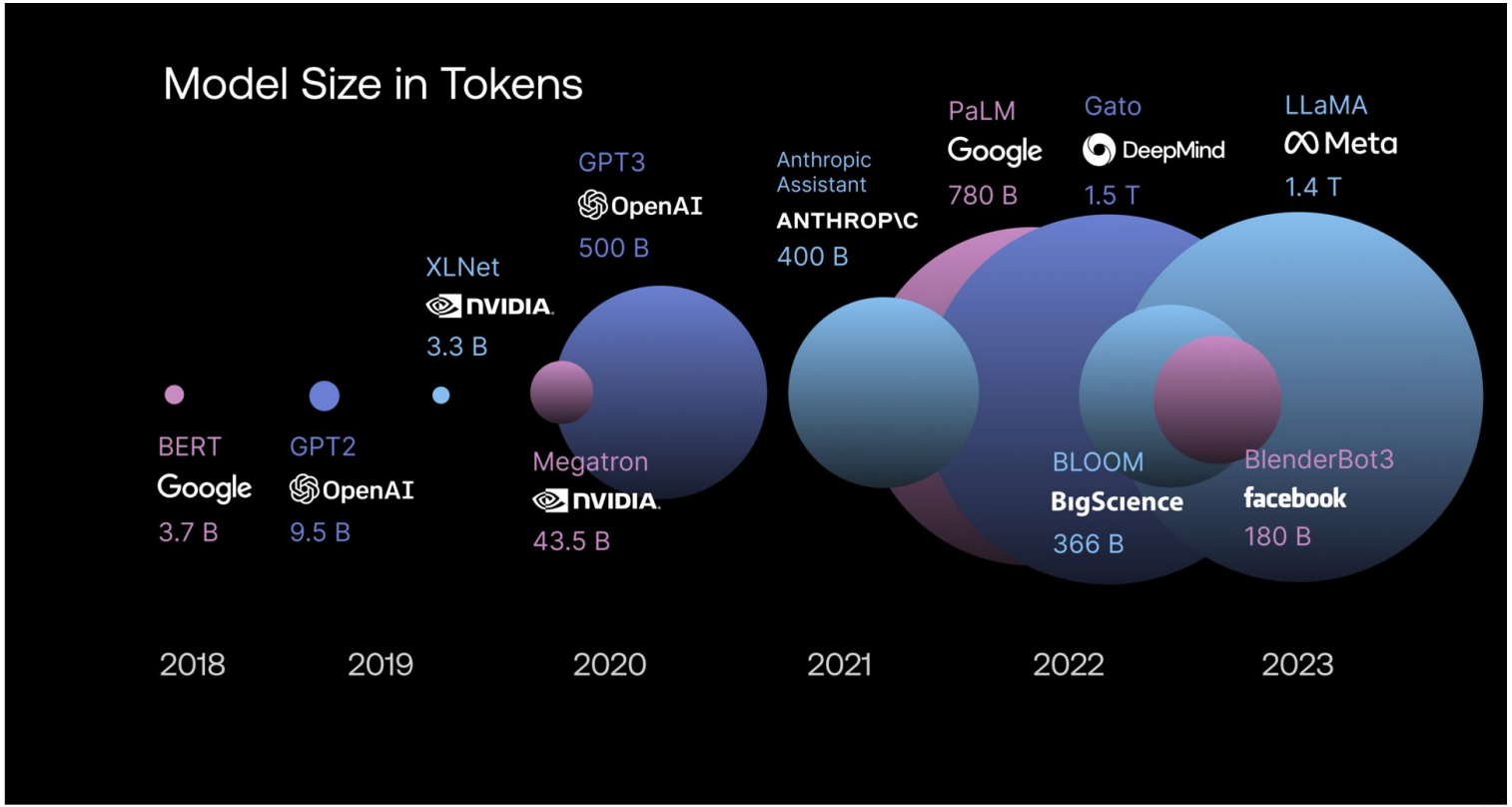
Uses advanced computing, its own neural network, to adapt with little to no human intervention

Quelle est la différence entre DL et NN (before 2010)

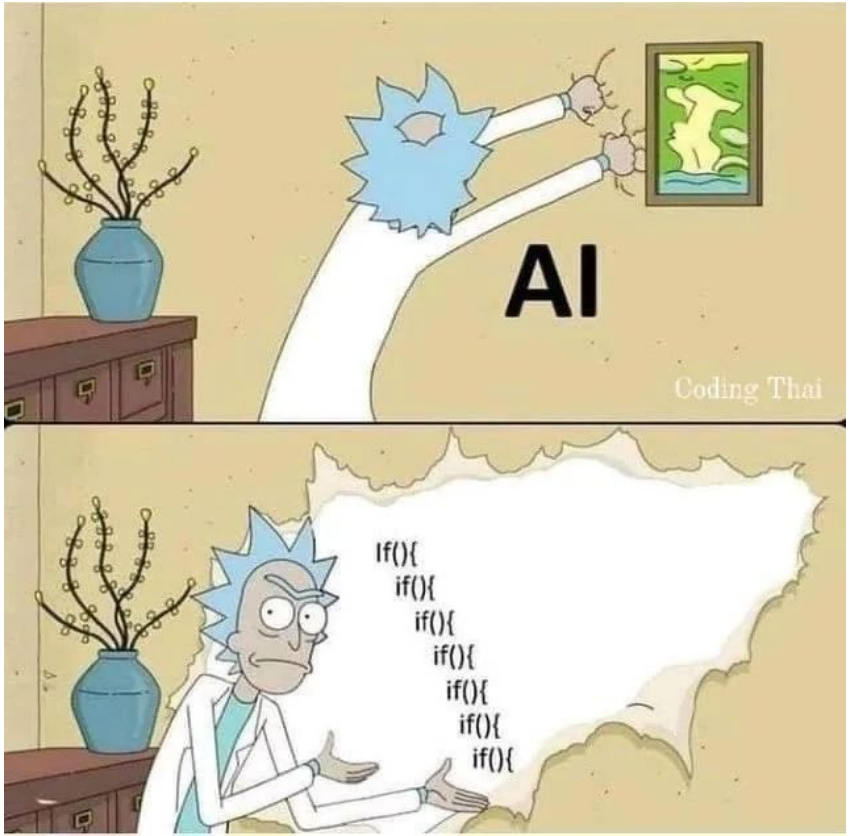


http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf

Quelle est la différence entre DL et NN (before 2010)



Deep learning



ChatGPT 4o >

9.11 and 9.9 – which is bigger

9.11 is bigger than 9.9.

subtract them

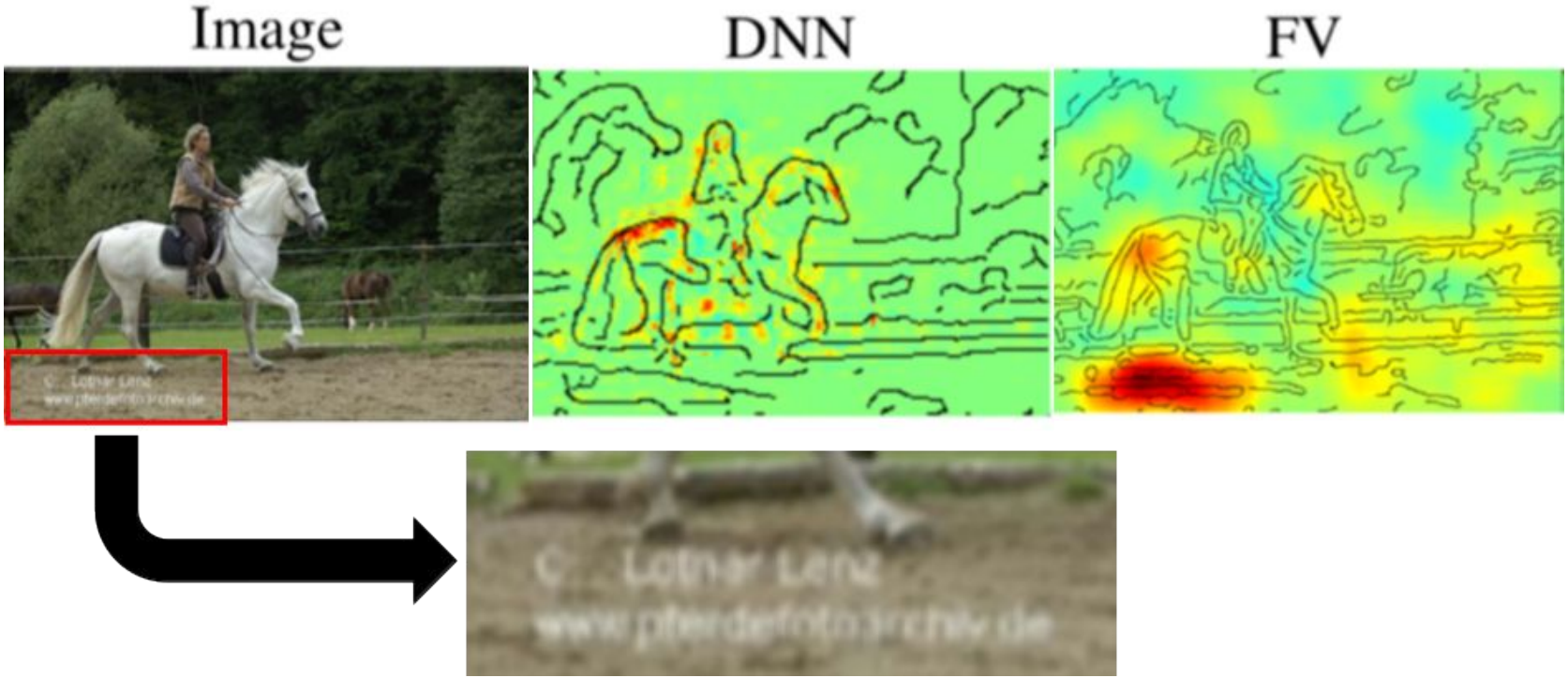
To subtract 9.9 from 9.11, you perform the following calculation:

$$9.11 - 9.9 = 0.21$$

use python

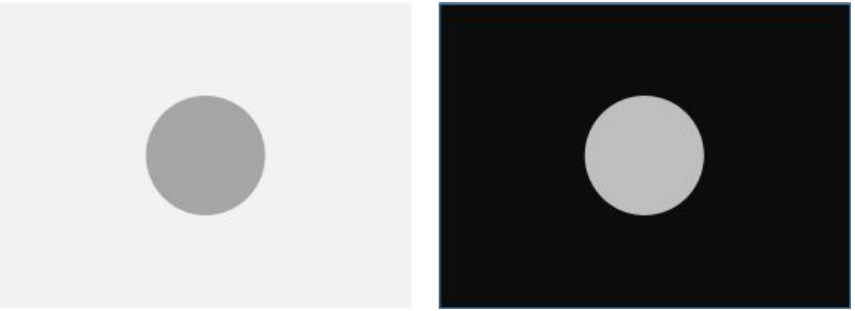
The result of subtracting 9.9 from 9.11 in Python is -0.79. This discrepancy is due to the way floating-point arithmetic is handled in Python, which can lead to small precision errors. The actual intended result should be 0.21.

Deep learning

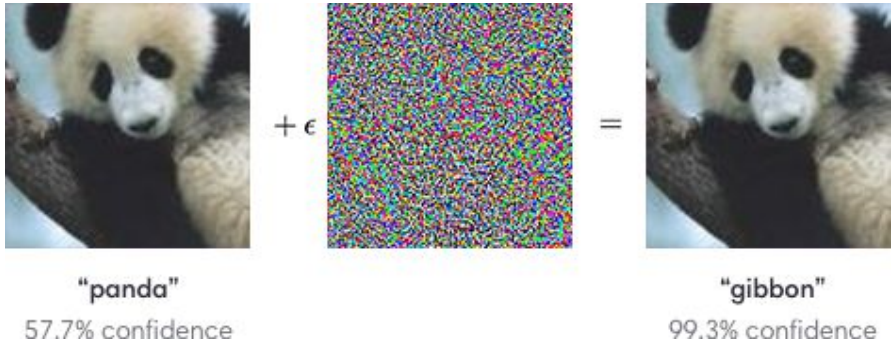


Deep learning

- L’human fais l’erreur:

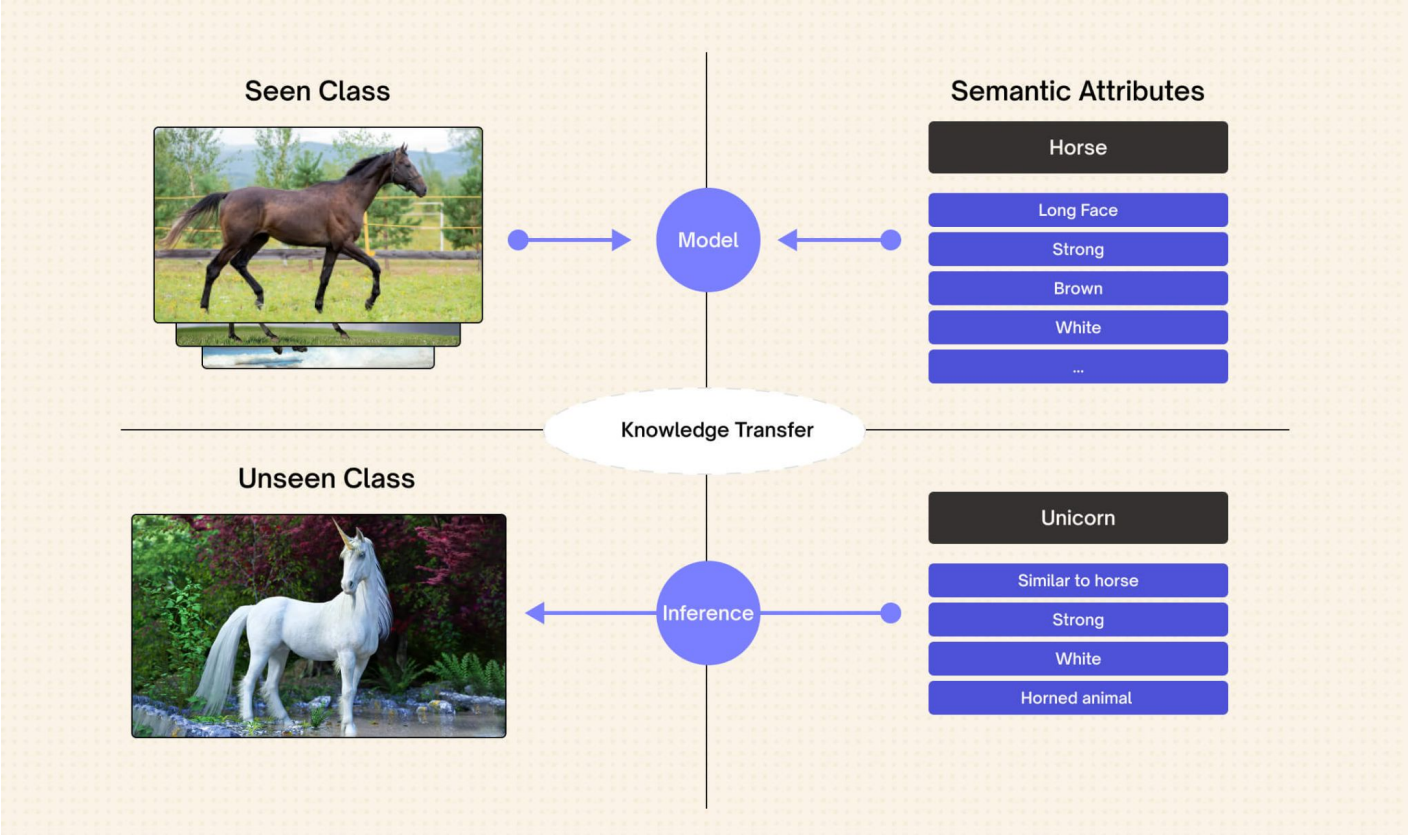


- Le modèle aussi:

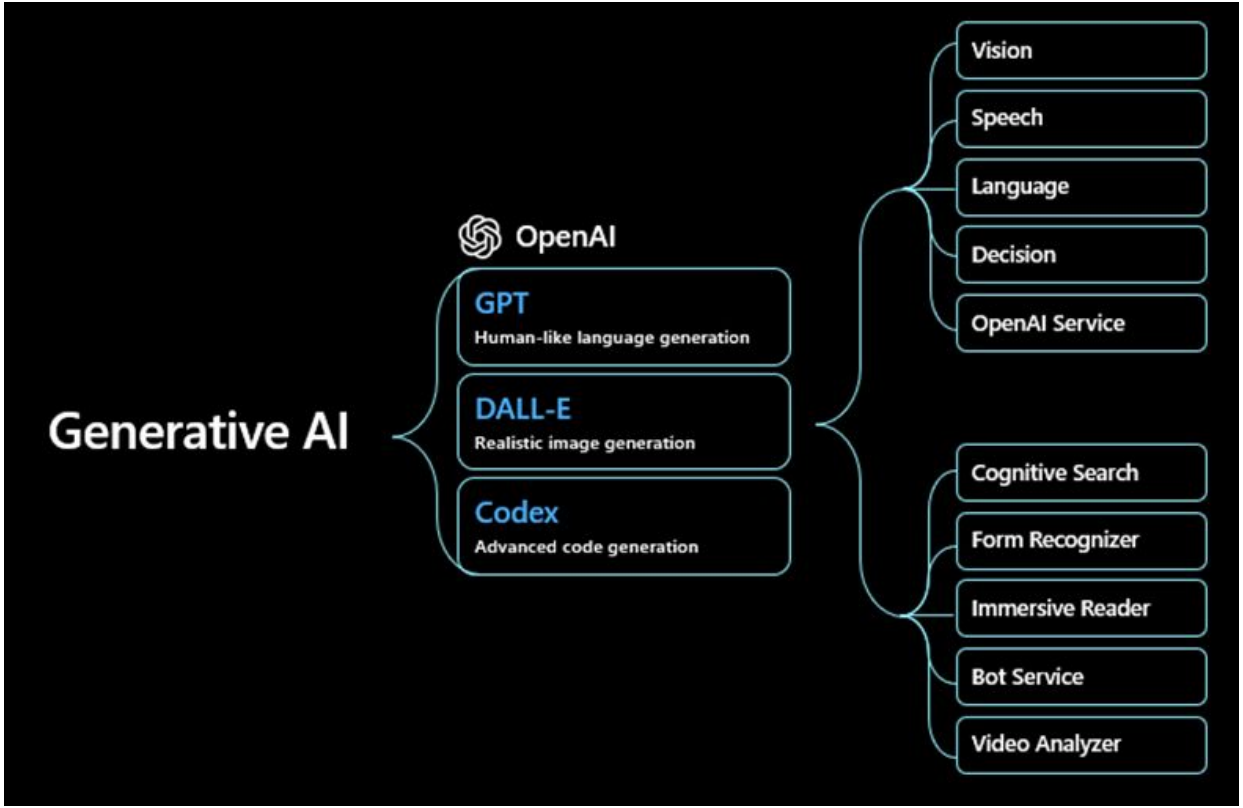


Deep learning

- Few-shot learning
- Zero-shot learning



Deep learning



Deep learning

Pourquoi c’est si difficile? -> Distribution change

Training data

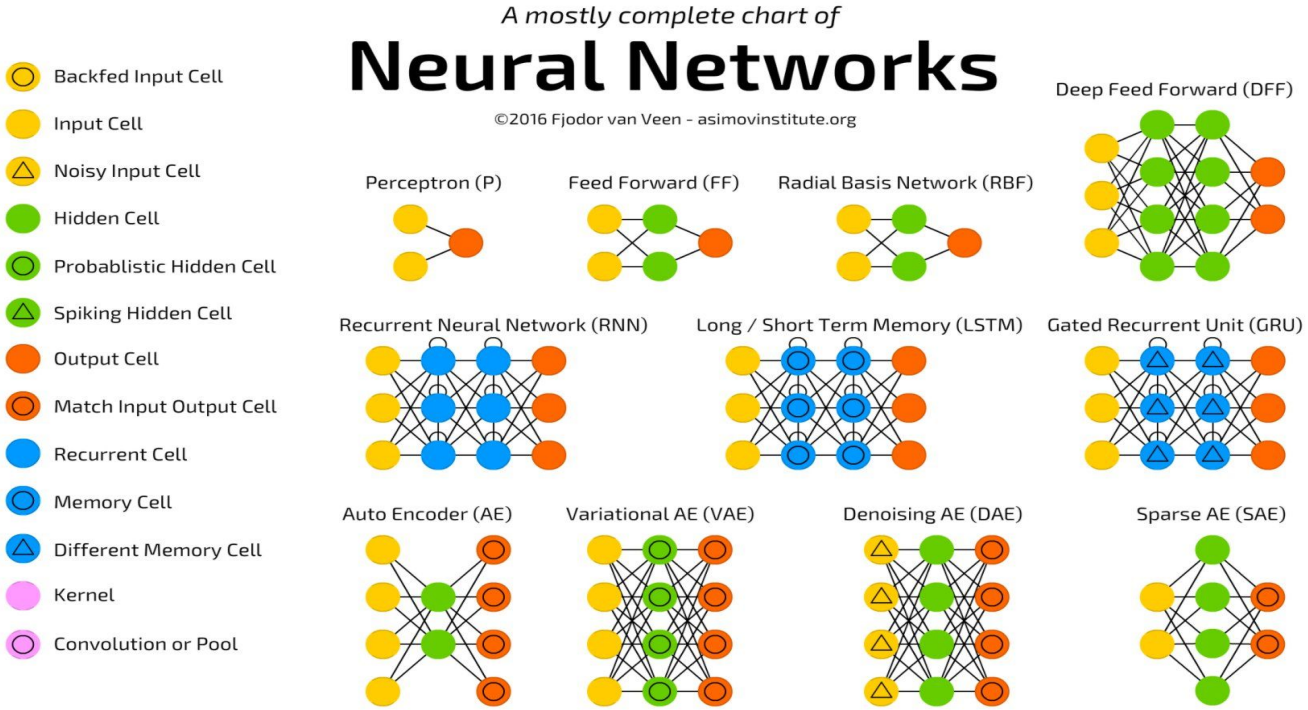
Testing data

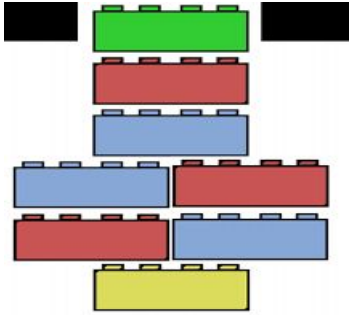
99.5%

57.5%

Structure d'un modèle deep learning

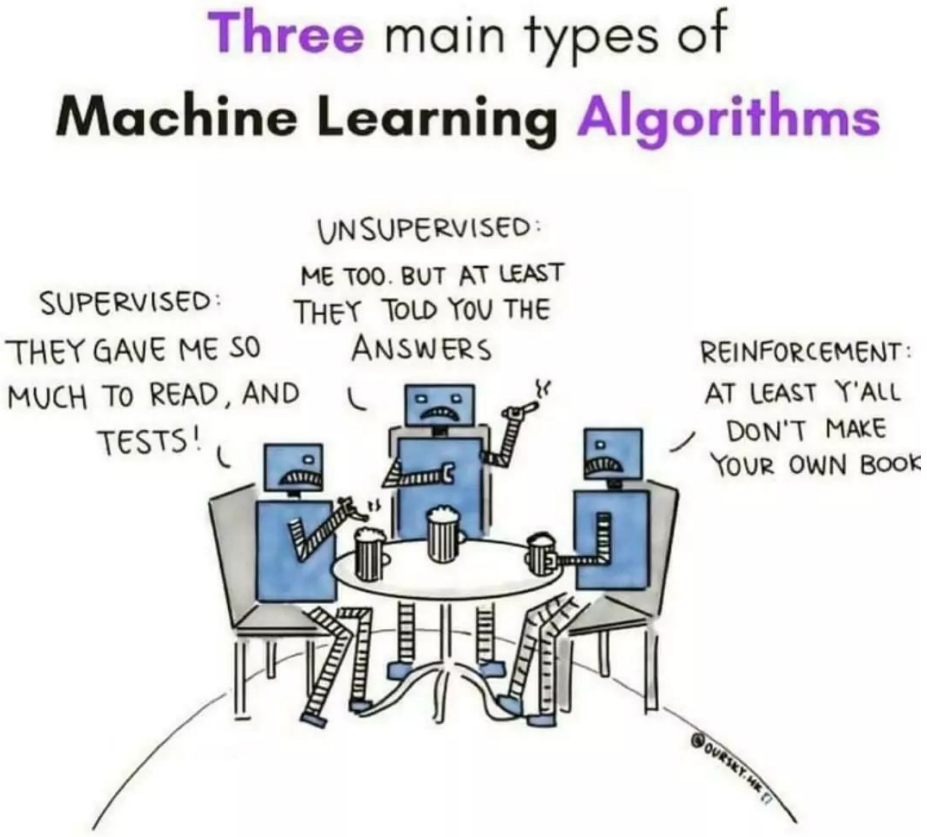
CNN, LSTM ou un LLM sont just des façons differentes de connecter entres des couches des neurons.



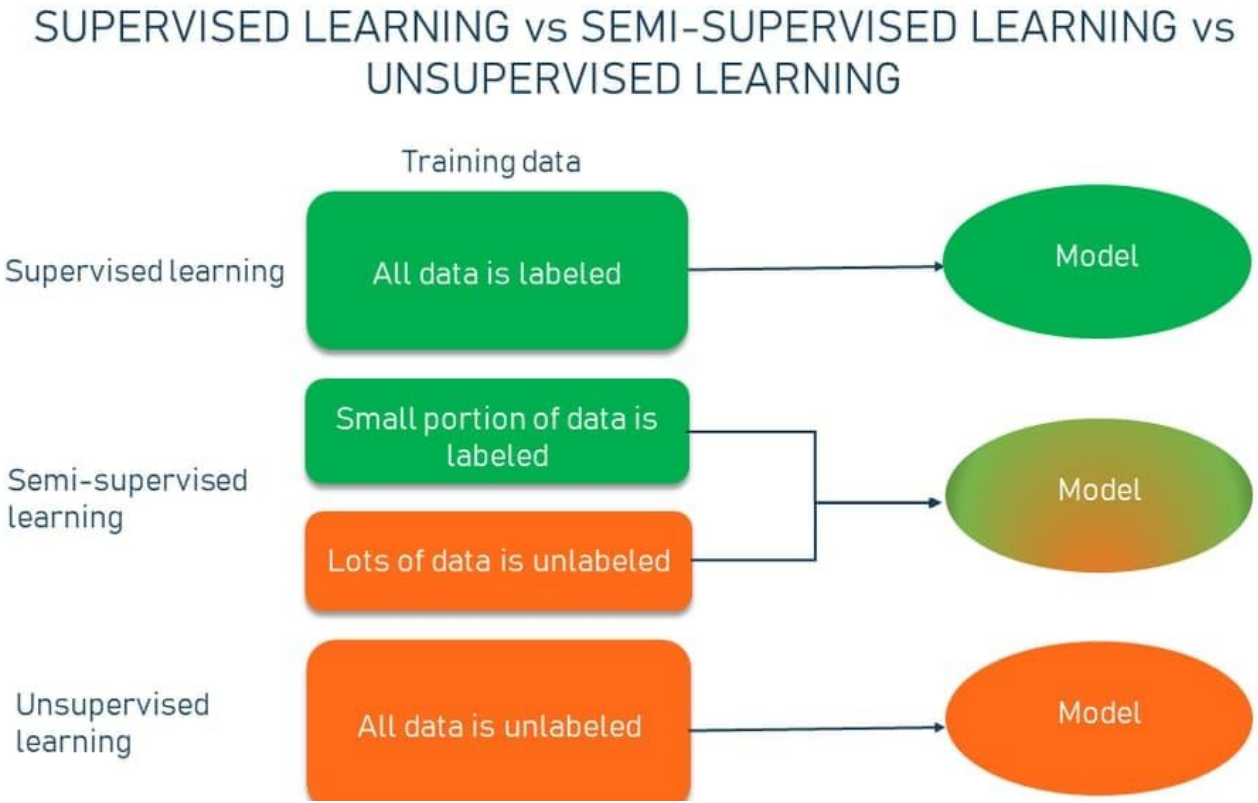


Typologie d’apprentissage

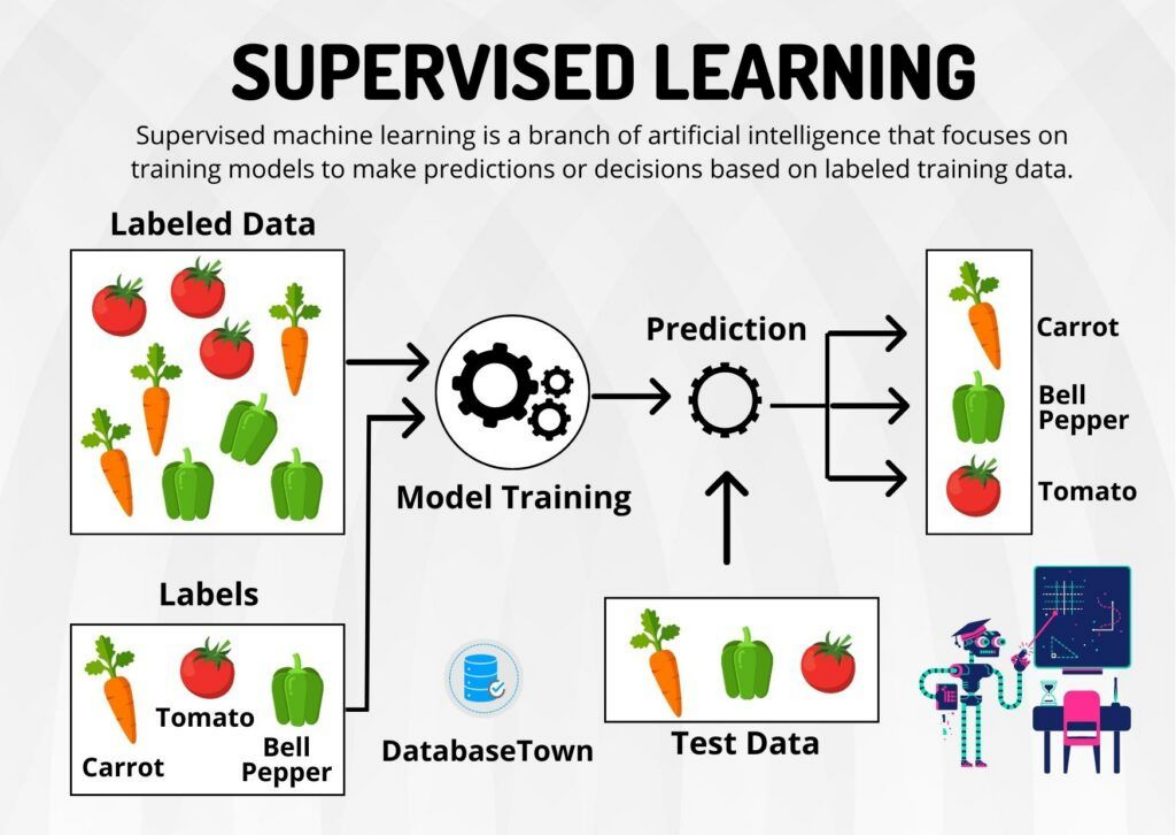
Reinforcement Learning
Apprentissage par renforcement



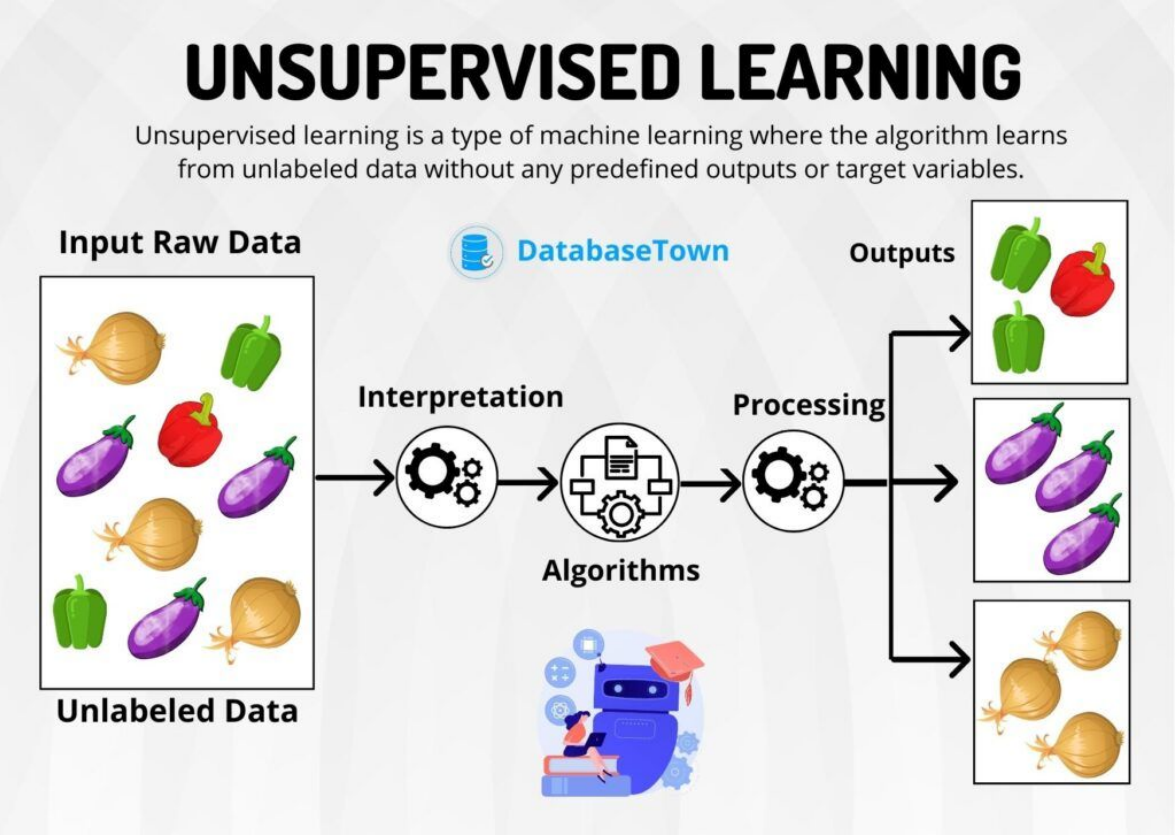
Typologie d'apprentissage



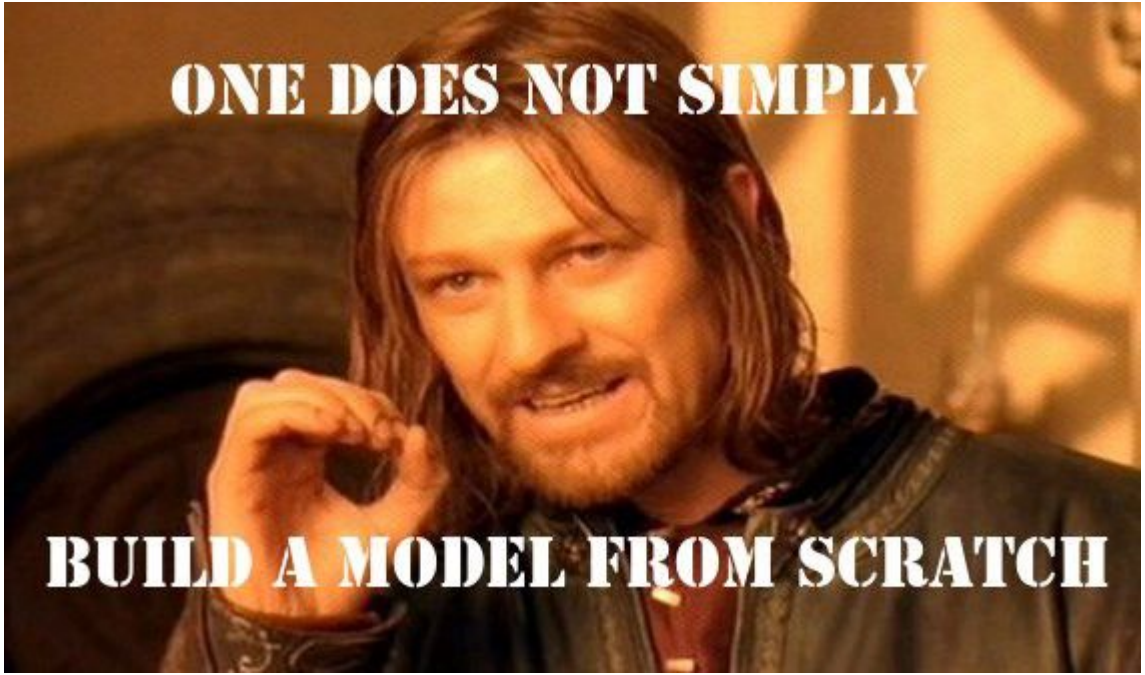
Typologie d’apprentissage



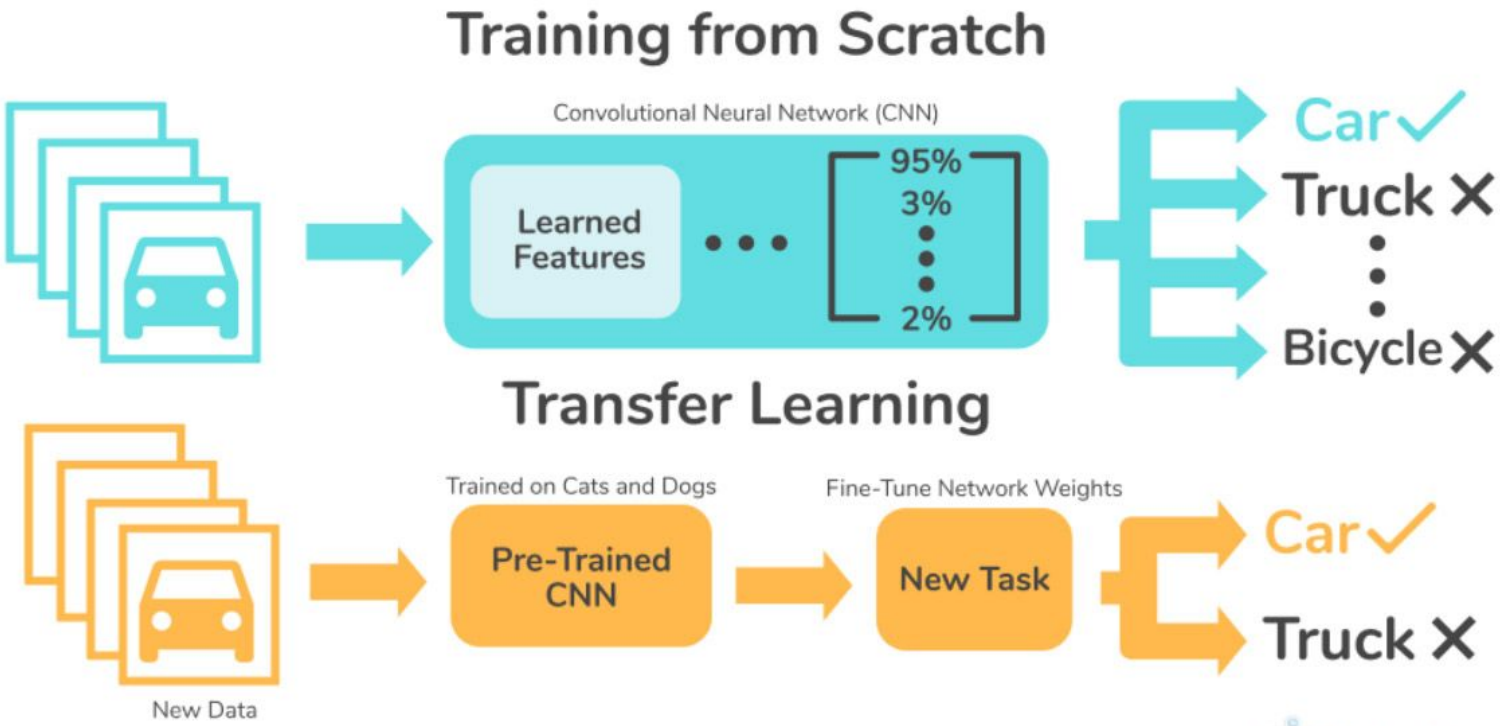
Typologie d’apprentissage



Typologie d’apprentissage

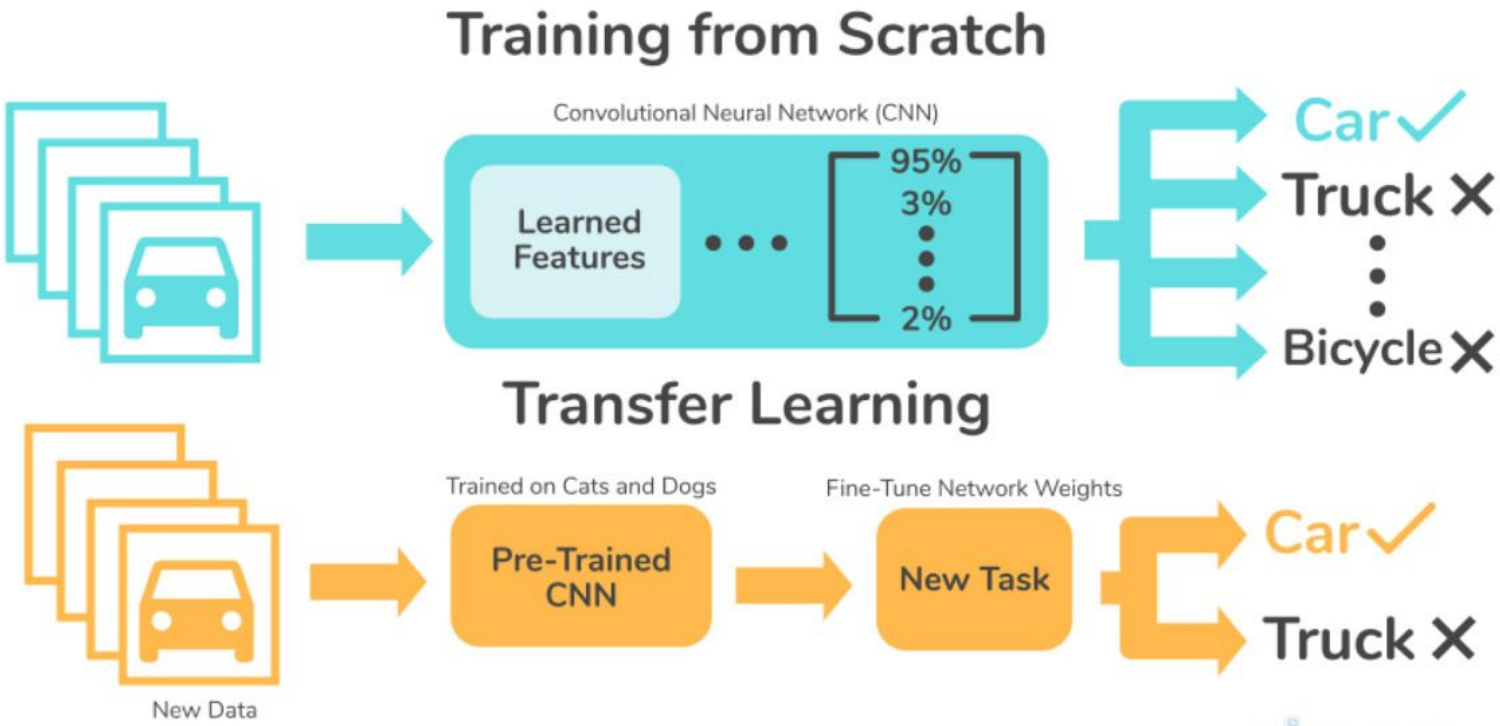


Typologie d'apprentissage



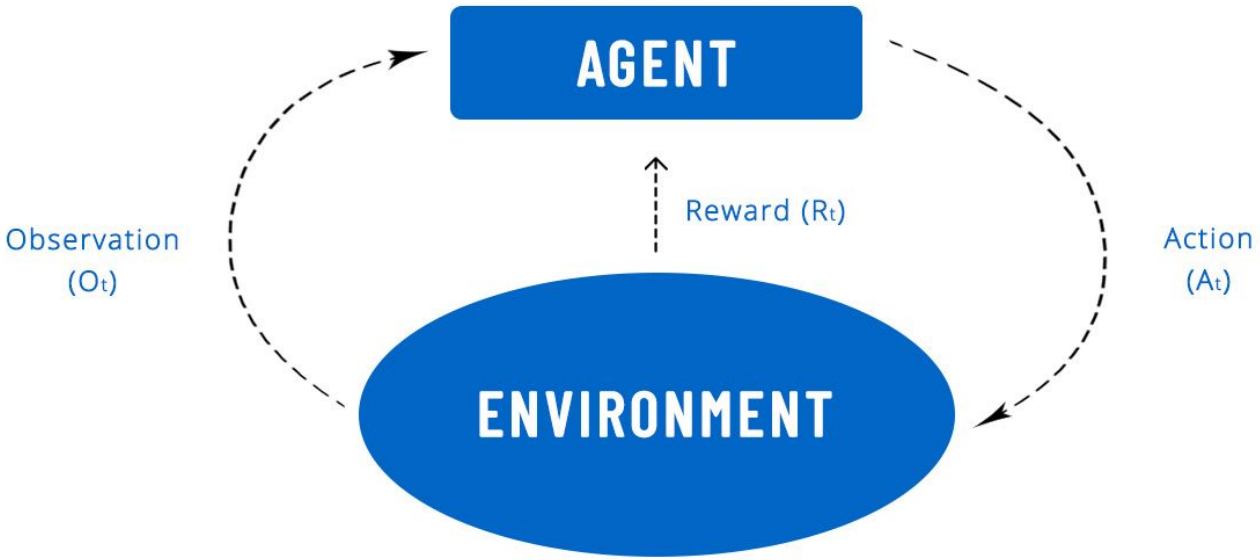
Free Time Learn

Typologie d'apprentissage



Free Time Learn

Typologie d’apprentissage



Typologie d'apprentissage

- Supervised:

Next move: "5-5"

Next move: "3-3"

- Reinforcement Learning

First move → ..... many moves ..... → Win!

Alpha Go is supervised learning + reinforcement learning.

Deep learning

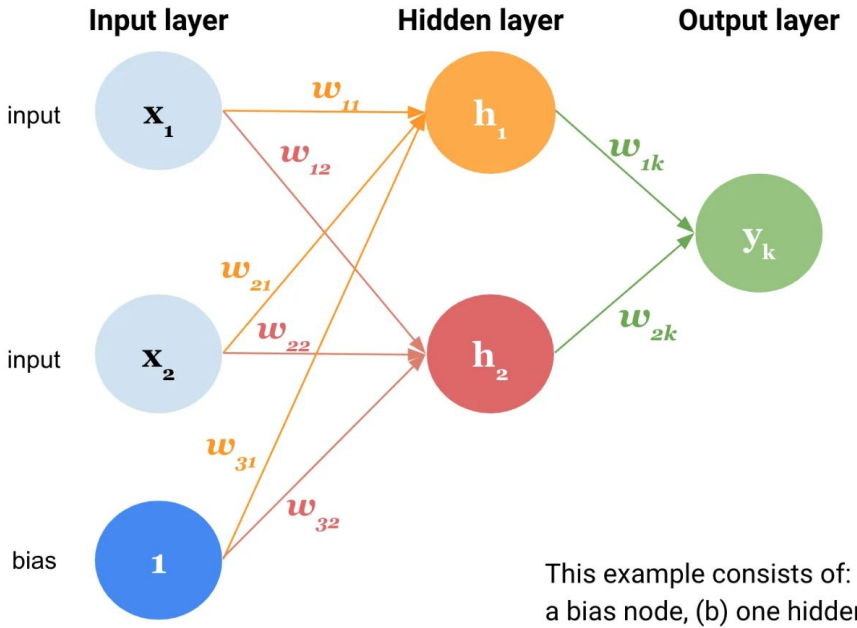
L’architecture classique:

- MLP
- CNN
- Transfer learning
- RNN
- Embedding
- Sequence-to-sequence
- Deep generative model

MLP

GNOMON DIGITAL

Illustrative example of Multilayer perceptron, a Feedforward neural network



x_1 , x_2 : *input data features*

w_{ij} : *weights of the network*

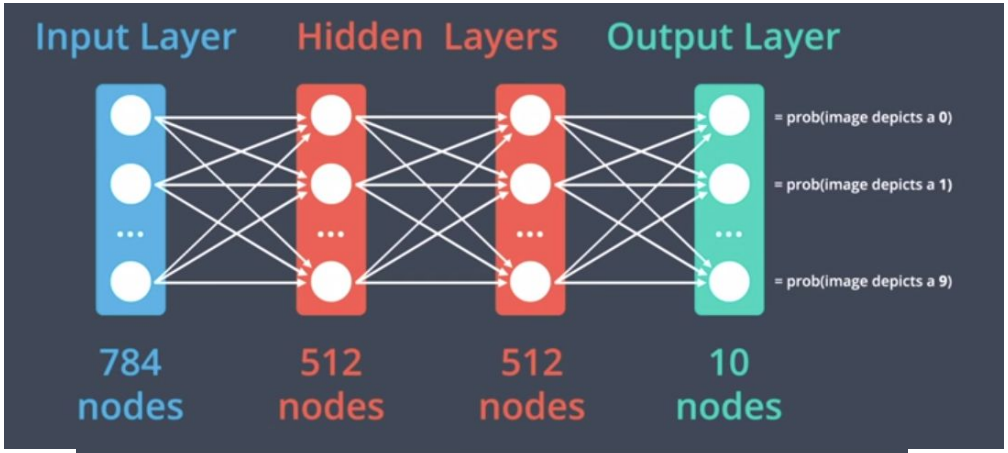
h_1 , h_2 : *nodes in the hidden layer*

y_k : *output variable*

© AIML.com Research

This example consists of: (a) an input layer with two input nodes and a bias node, (b) one hidden layer with two neurons, and (c) an output layer with one neuron

MLP

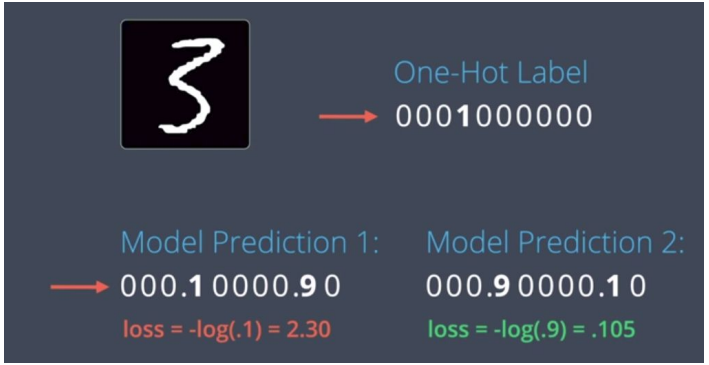


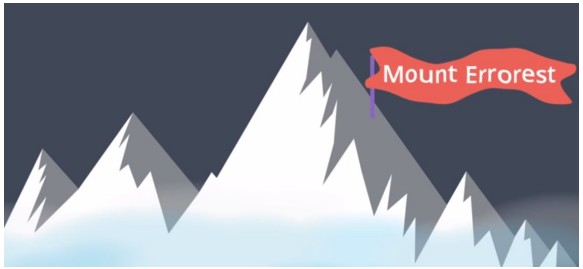

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten

# define the model
model = Sequential()
model.add(Flatten(input_shape=X_train.shape[1:]))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

# summarize the model
model.summary()
```

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_1 (Dense)	(None, 512)	401920
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262656
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 10)	5130

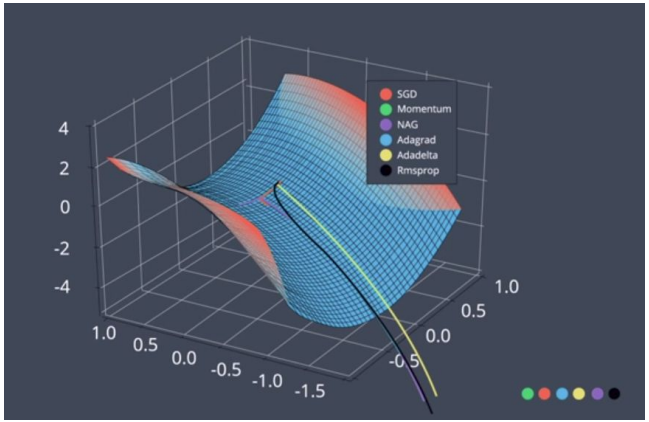
Total params: 669,706.0
Trainable params: 669,706.0
Non-trainable params: 0.0

MLP

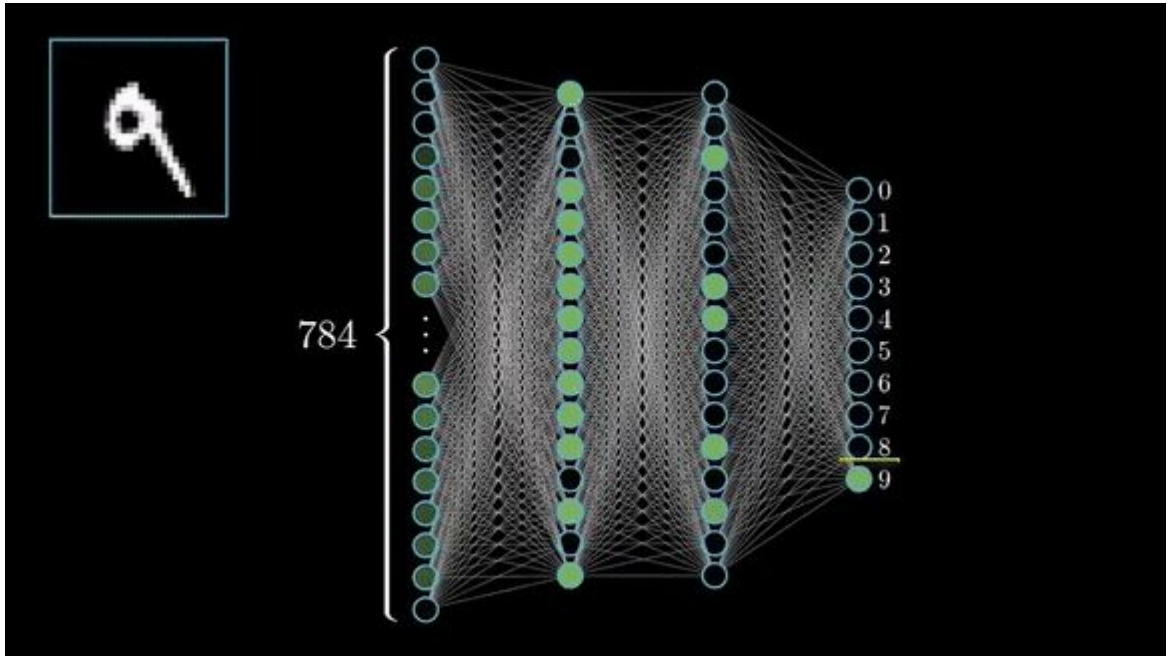




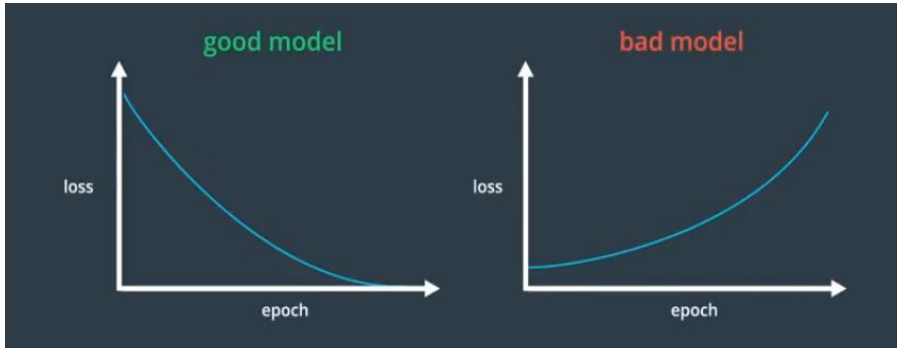
Categorical Cross-Entropy

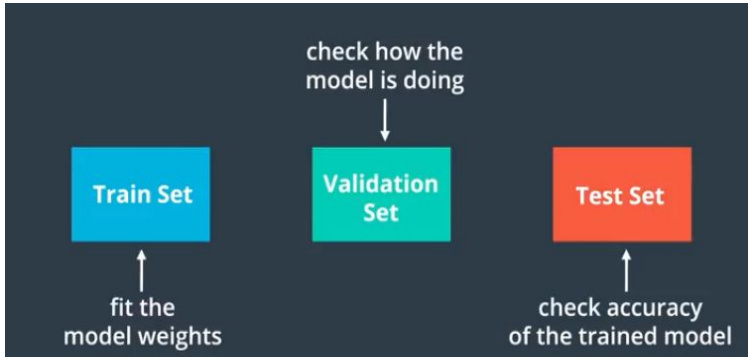


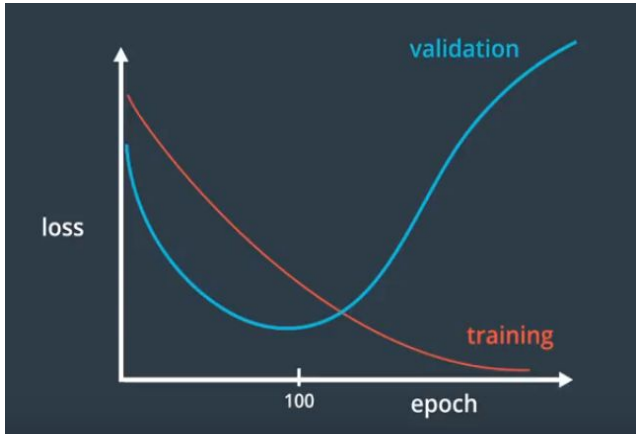
MLP



MLP

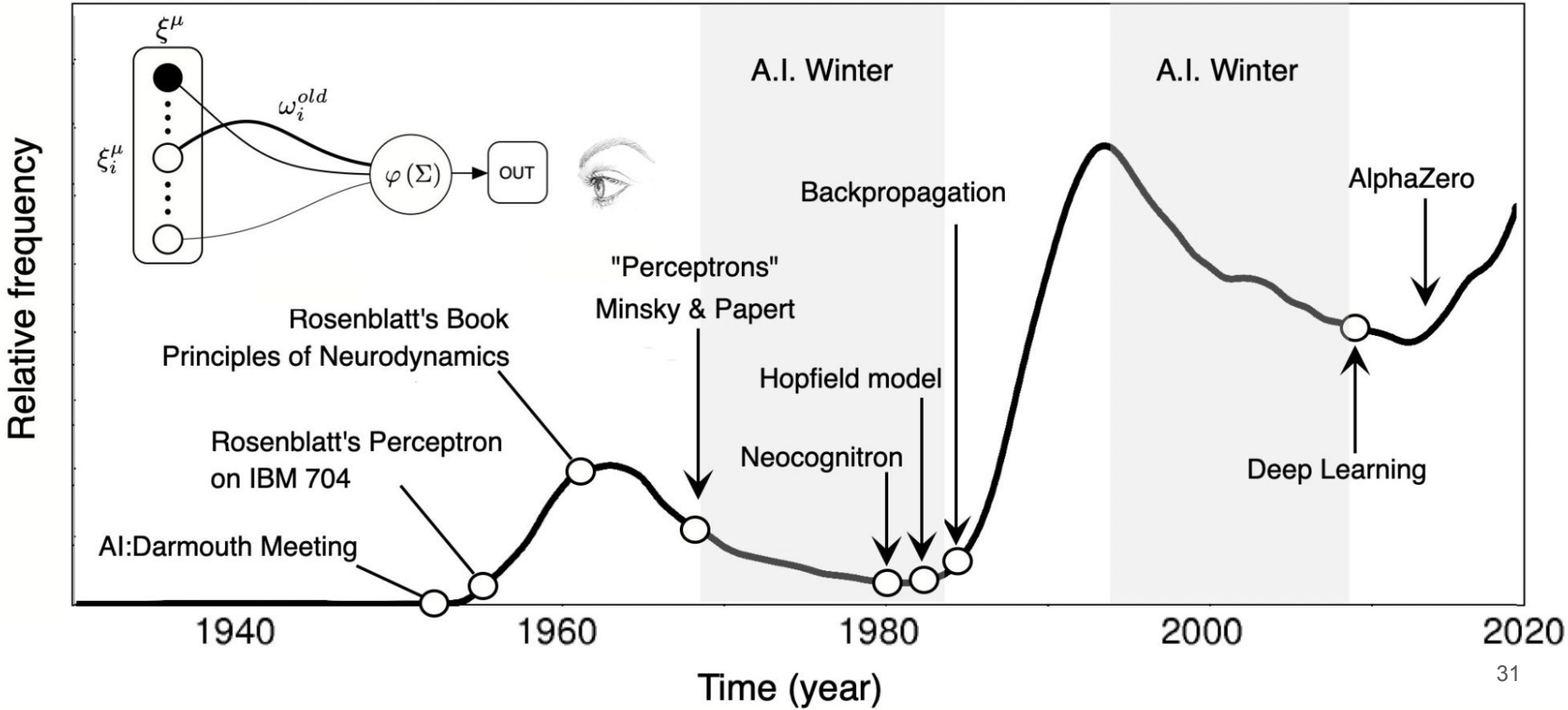

good model
loss
epoch
bad model
loss
epoch


check how the model is doing
Train Set
Validation Set
Test Set
fit the model weights
check accuracy of the trained model

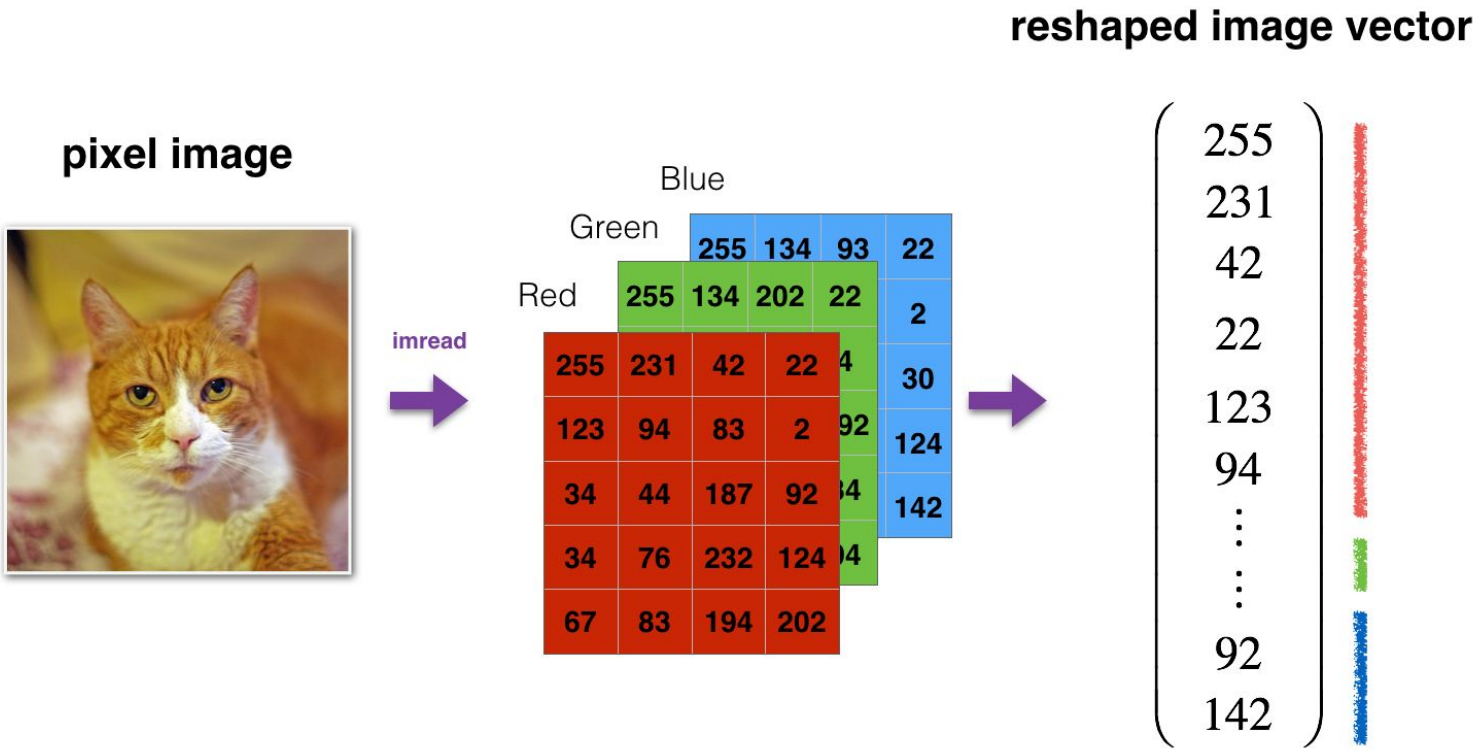

validation
loss
training
100
epoch

BEST MODEL!
Model 1
val_loss = 0.9
Model 2
val_loss = 0.5
Model 3
val_loss = 1.8

CNN: Convolutional Neural Network



CNN: Convolutional Neural Network



Données non tabular sont difficiles à comprendre pour la machine

CNN: Convolutional Neural Network

MLP ne fonctionne bien sur la classification d'images

THE MNIST DATABASE

of handwritten digits

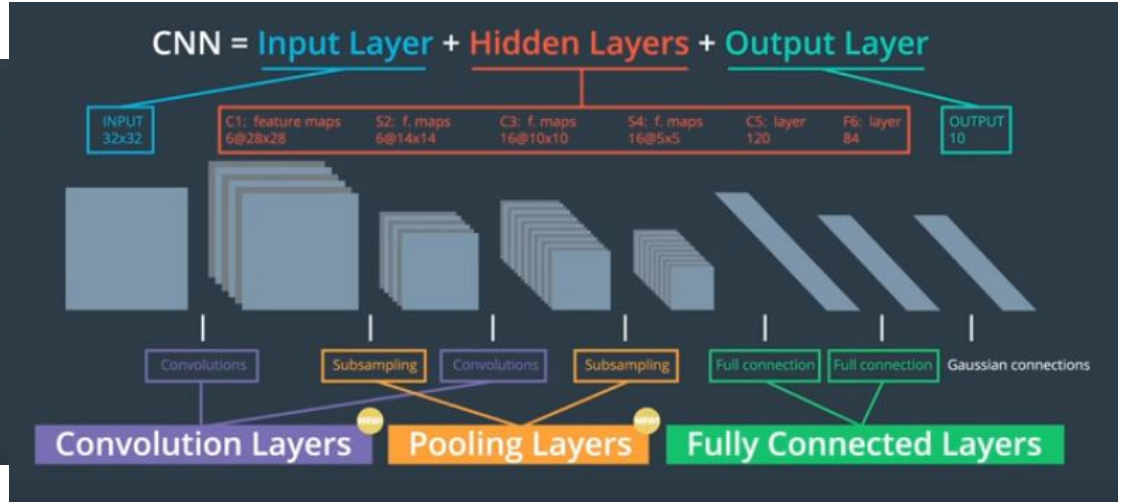
Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

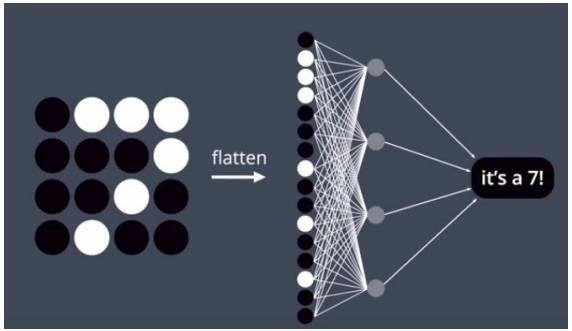
It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

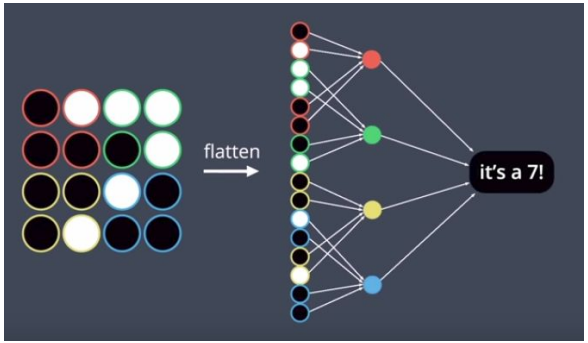
Four files are available on this site:

train-images-idx3-ubyte.gz: training set images (9912422 bytes)

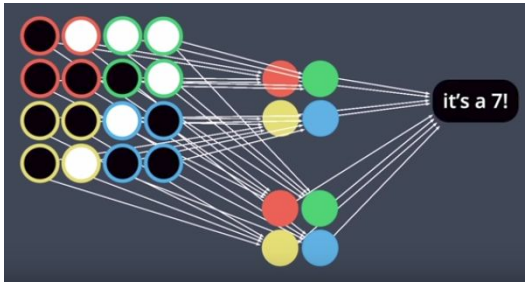


CNN: Convolutional Neural Network



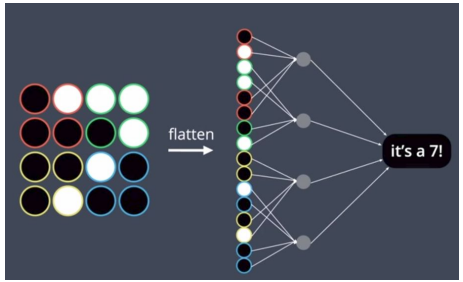


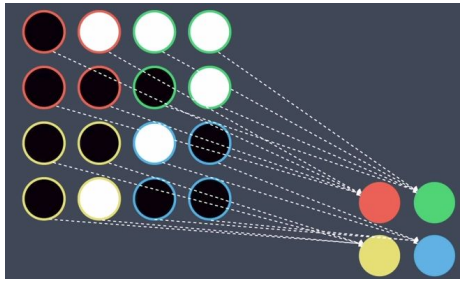
Local Connectivity

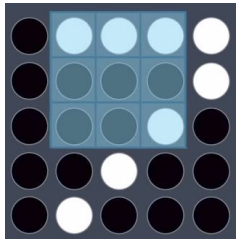


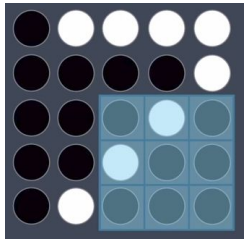
CNN: Convolutional Neural Network

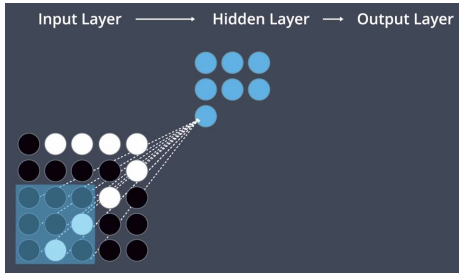
GNOMON DIGITAL®

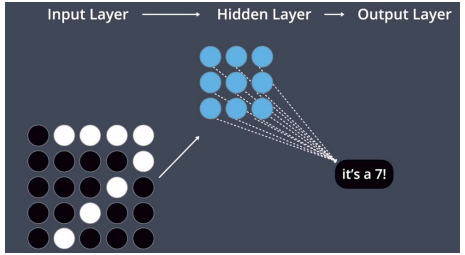


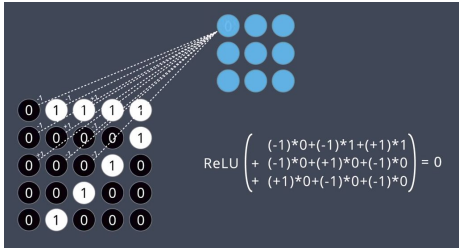


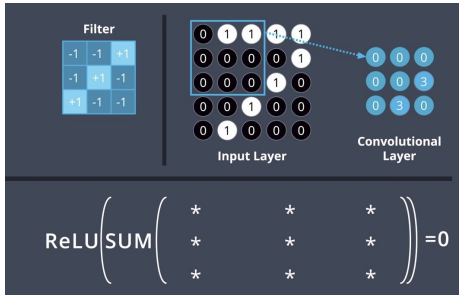


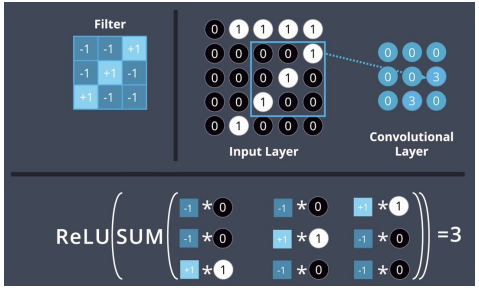




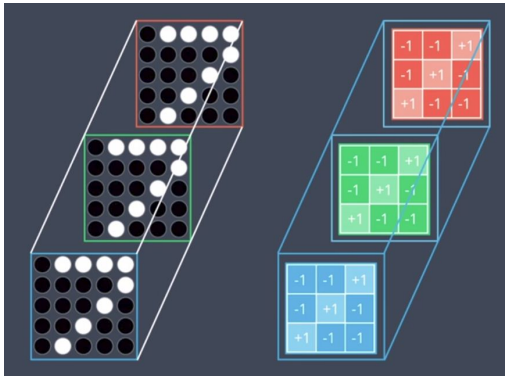


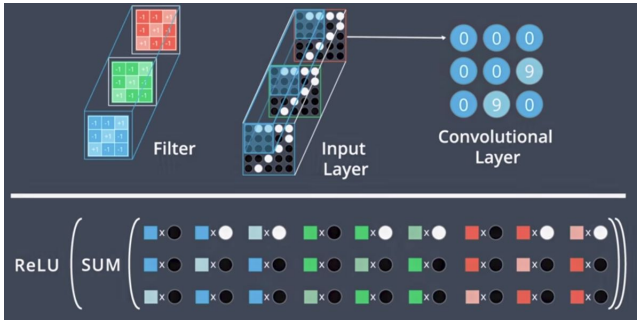


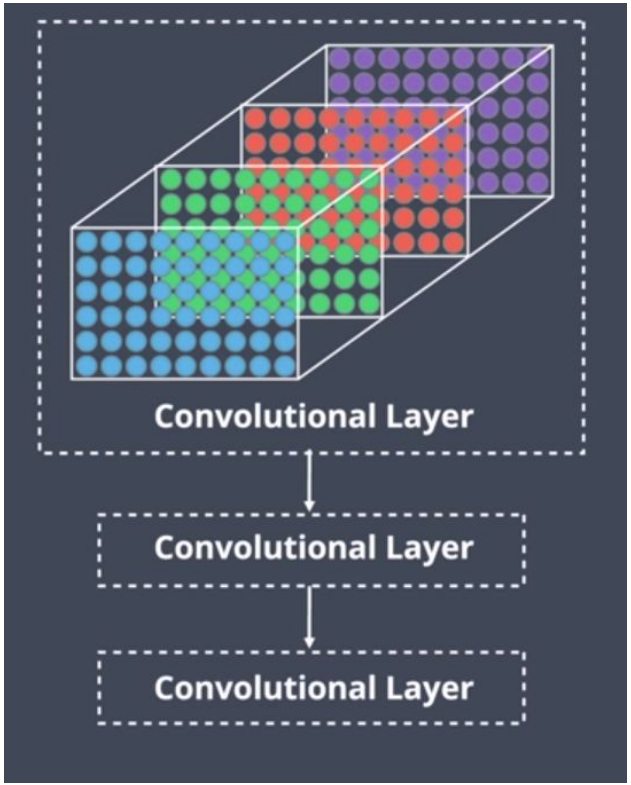




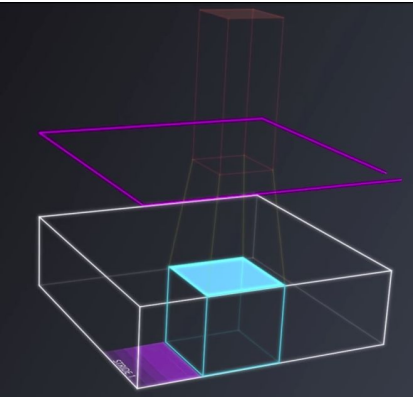
CNN: Convolutional Neural Network

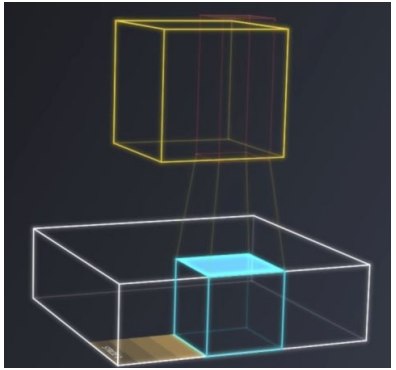


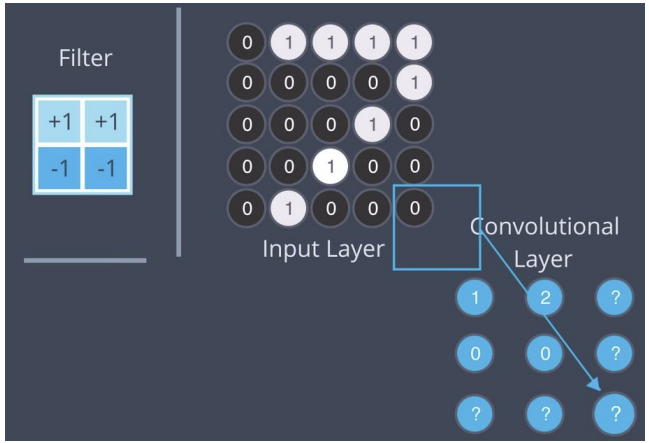


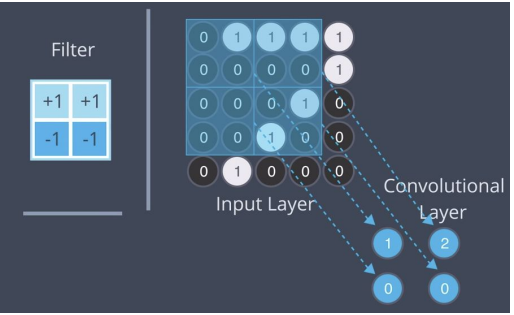


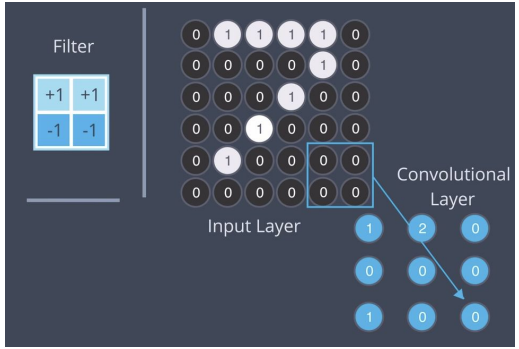
CNN: Convolutional Neural Network





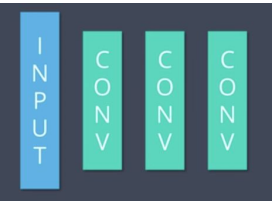


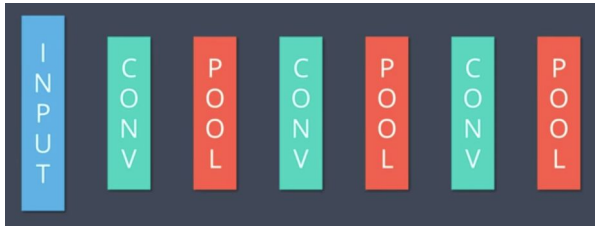


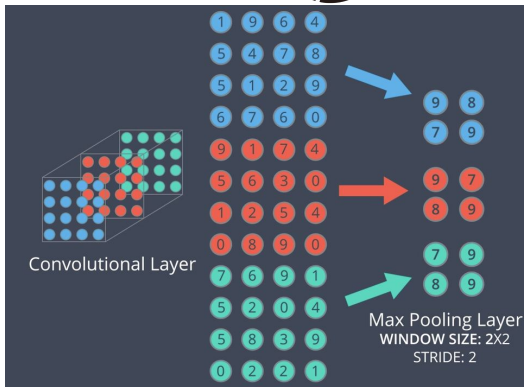


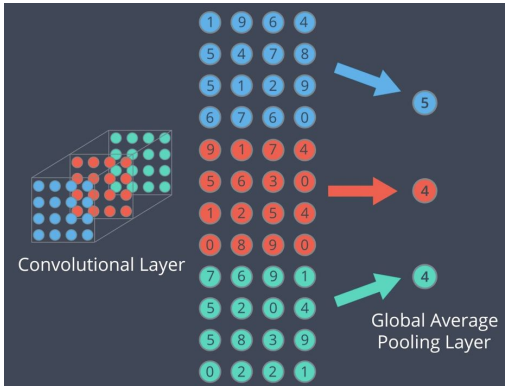
Stride and Padding

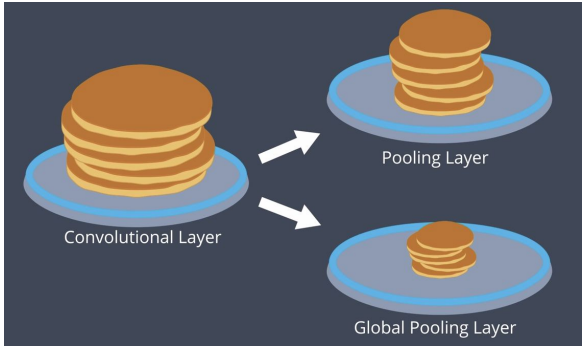
CNN: Convolutional Neural Network



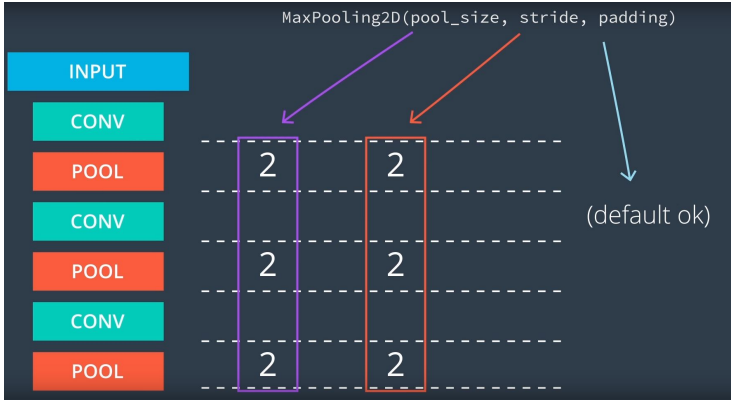


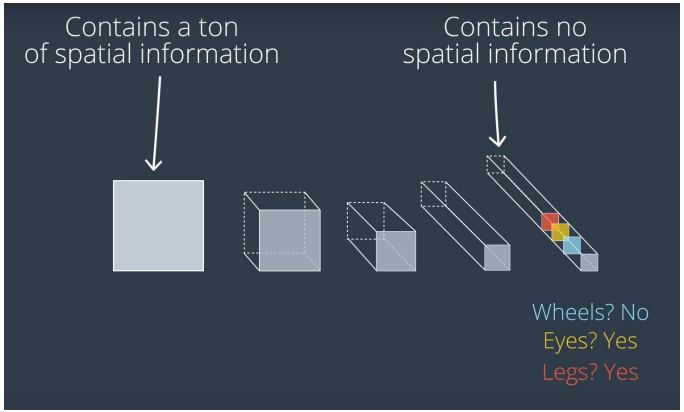






CNN: Convolutional Neural Network

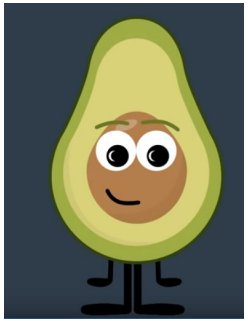


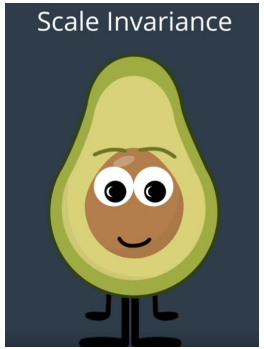


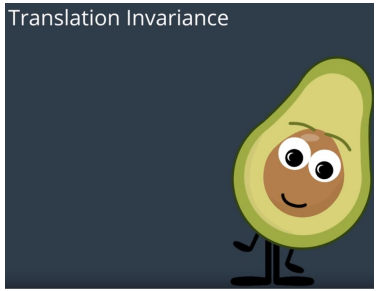

```python
from keras.models import Sequential
from keras.layers import Convolution2D, MaxPooling2D

model = Sequential()
model.add(Convolution2D(filters=16, kernel_size=2, padding='same', activation='relu',
                        input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Convolution2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Convolution2D(filters=64, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 32, 32, 16)        208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 16, 16, 32)        2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 64)          8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0
=================================================================
Total params: 10,544.0
Trainable params: 10,544.0
Non-trainable params: 0.0
```

CNN: Convolutional Neural Network







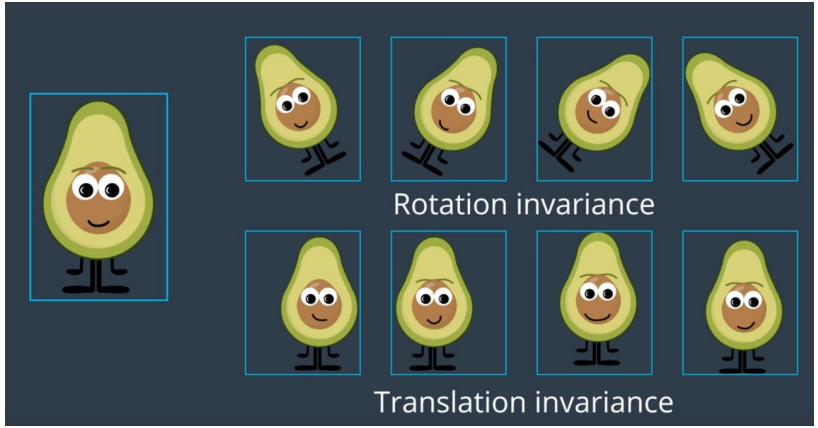
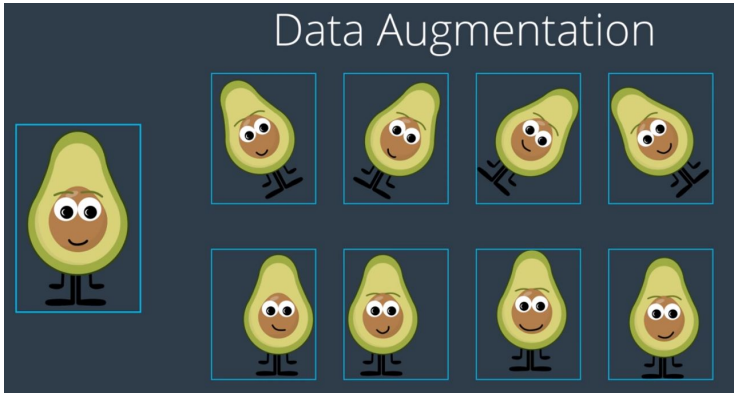


Image Augmentation

CNN: Convolutional Neural Network

GNOMON DIGITAL



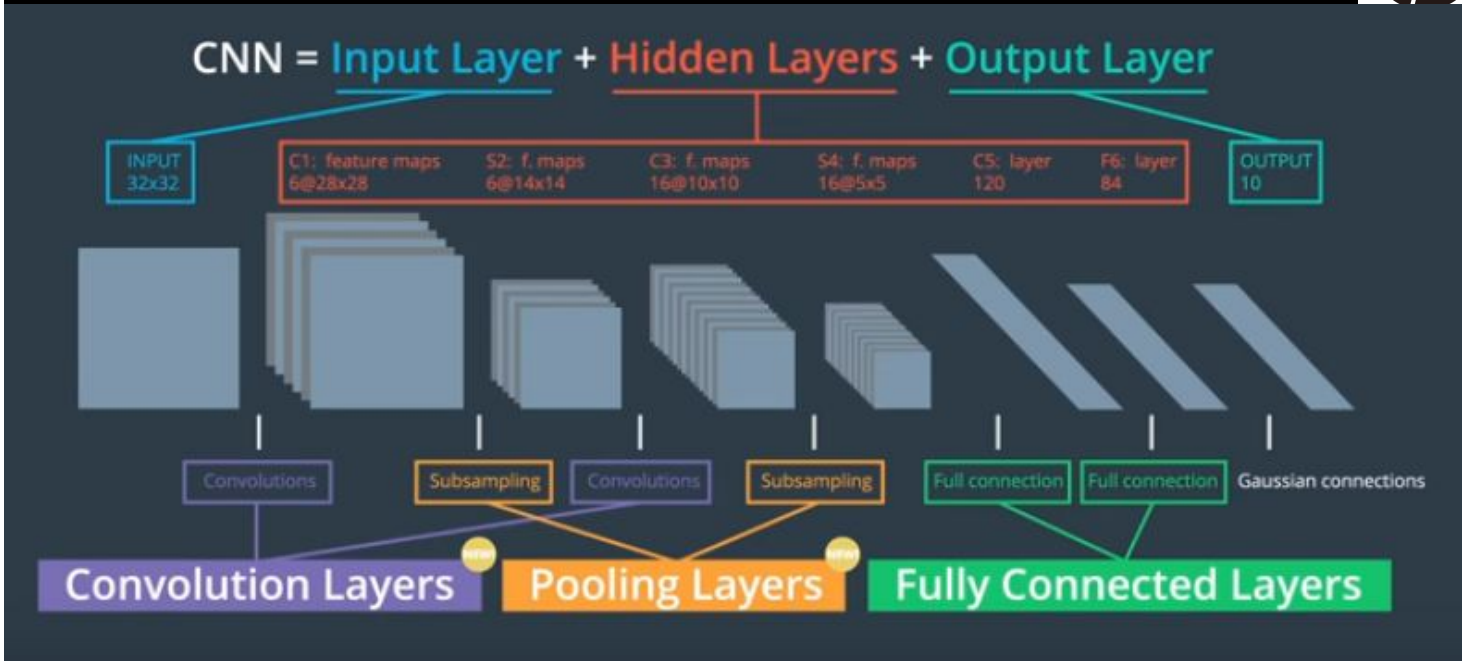
5. Create and configure augmented image generator

```
from keras.preprocessing.image import ImageDataGenerator

# create and configure augmented image generator
datagen = ImageDataGenerator(
    width_shift_range=0.1, # randomly shift images horizontally (10% of total width)
    height_shift_range=0.1, # randomly shift images vertically (10% of total height)
    horizontal_flip=True) # randomly flip images horizontally

# fit augmented image generator on data
datagen.fit(x_train)
```

6. Visualize original and augmented images

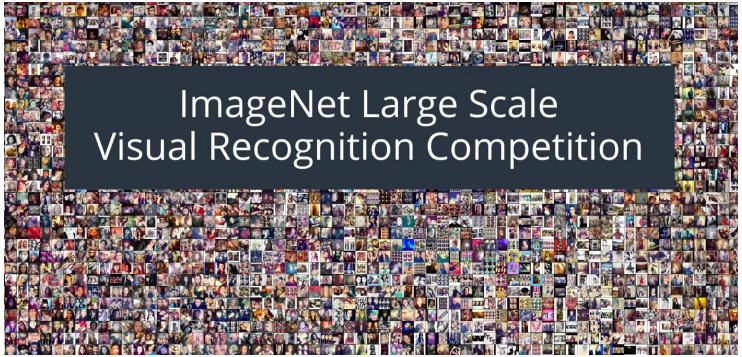
```
import matplotlib.pyplot as plt

# take subset of training data
x_train_subset = x_train[:12]

# visualize subset of training data
fig = plt.figure(figsize=(20,2))
for i in range(0, len(x_train_subset)):
    ax = fig.add_subplot(1, 12, i+1)
    ax.imshow(x_train_subset[i])
fig.suptitle('Subset of Original Training Images', fontsize=20)

# visualize augmented images
fig = plt.figure(figsize=(20,2))
for x_batch in datagen.flow(x_train_subset, batch_size=12):
    for i in range(0, 12):
        ax = fig.add_subplot(1, 12, i+1)
        ax.imshow(x_batch[i])
    fig.suptitle('Augmented Images', fontsize=20)
    plt.show()
    break;
```

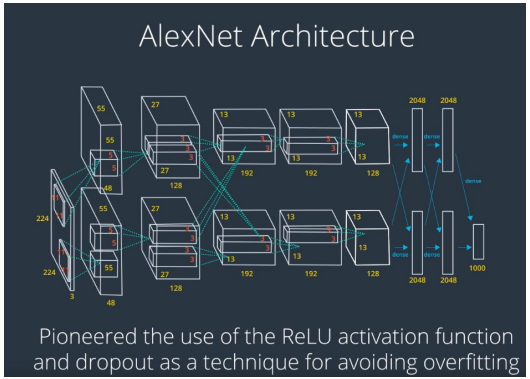
CNN: Convolutional Neural Network

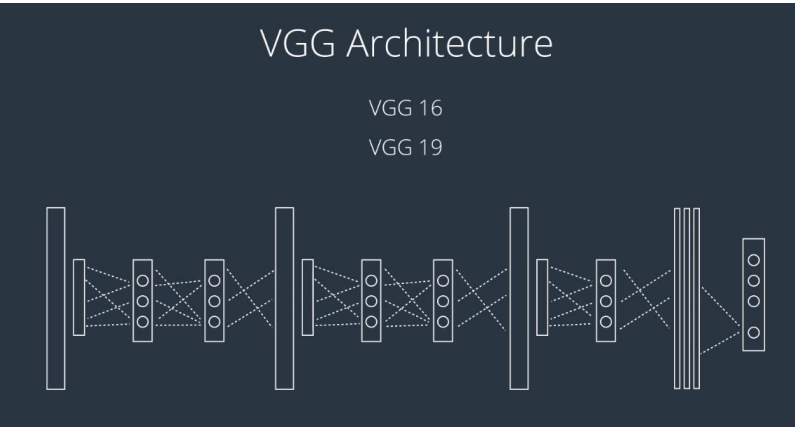


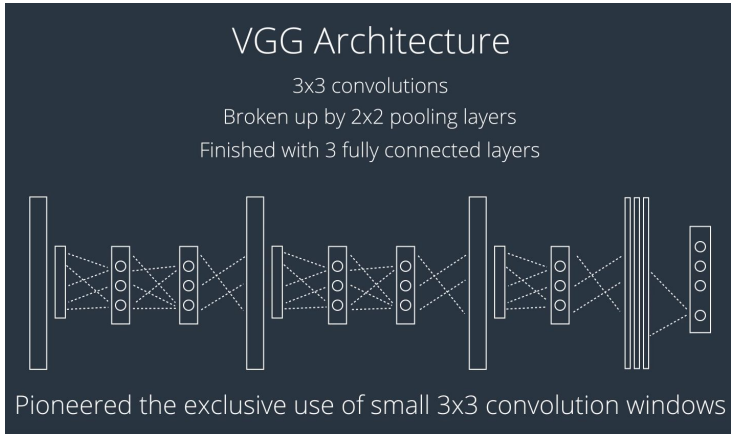
x N fois

CNN: Variances

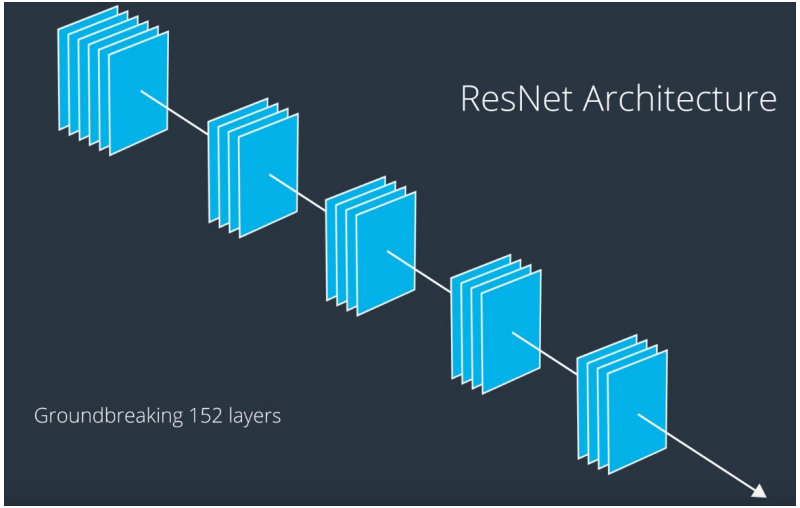


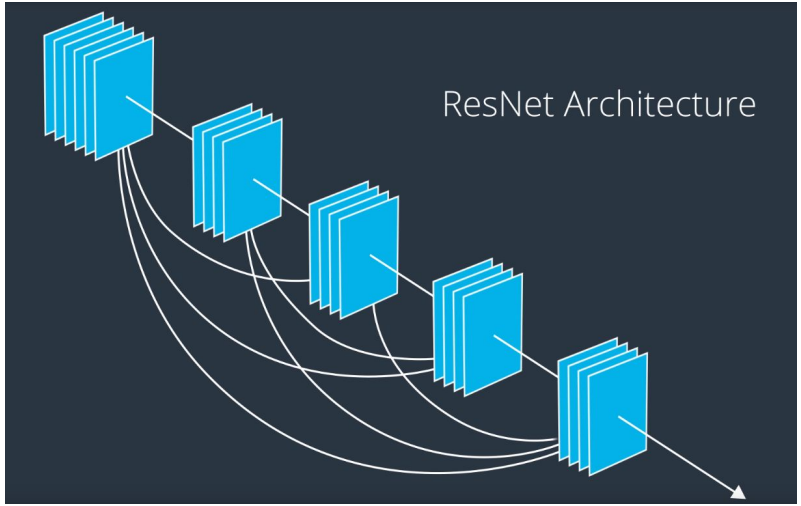






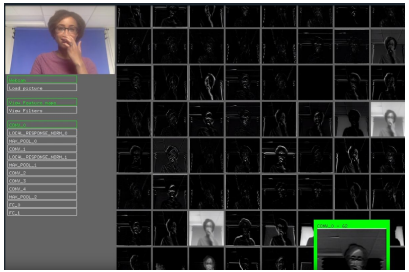
CNN: Variances

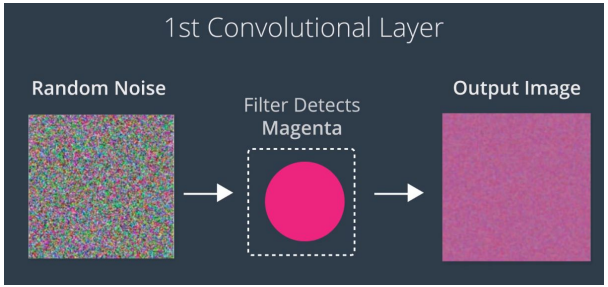


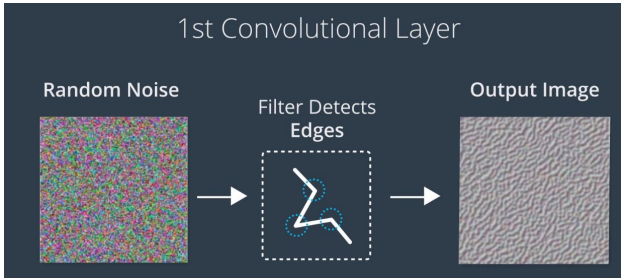


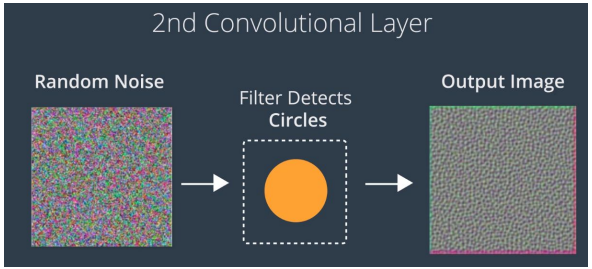
CNN

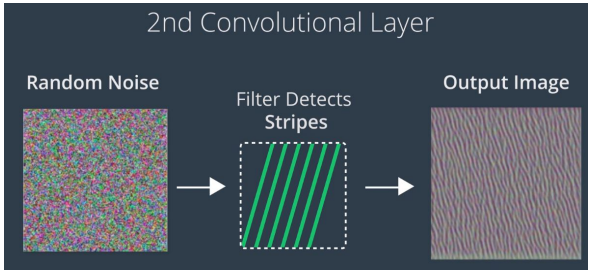
GNOMON DIGITAL®

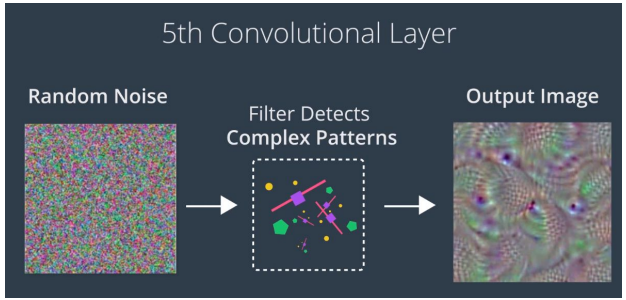


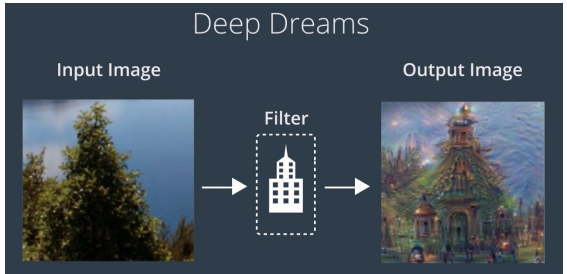




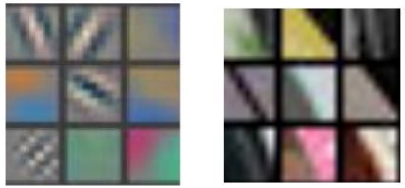


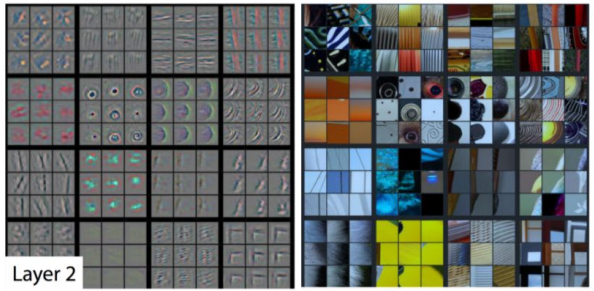


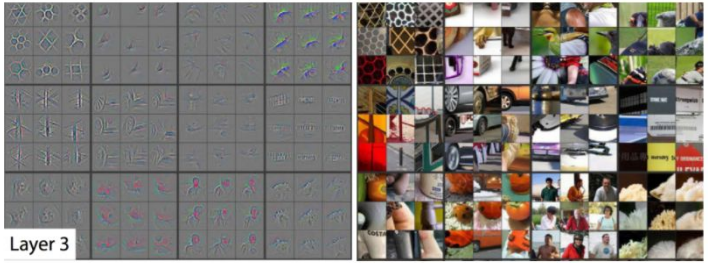




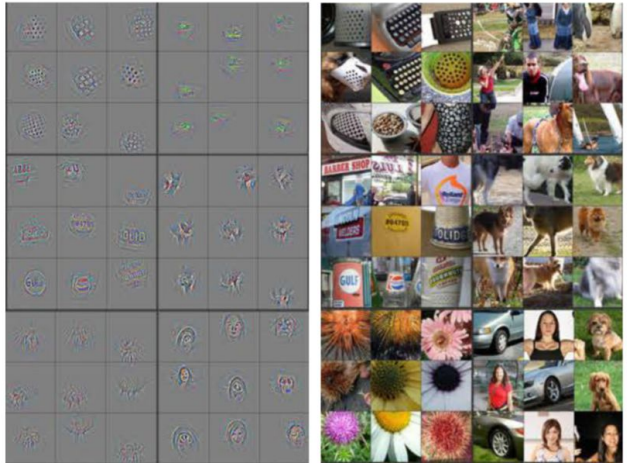
CNN

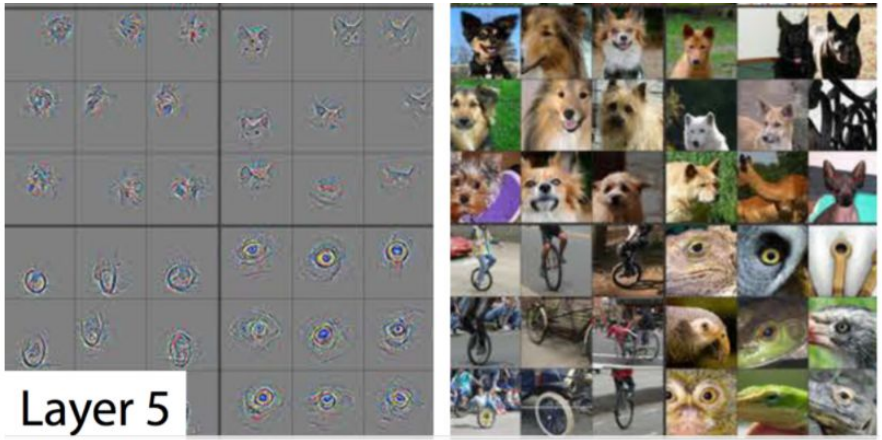






Layer 5





CNN

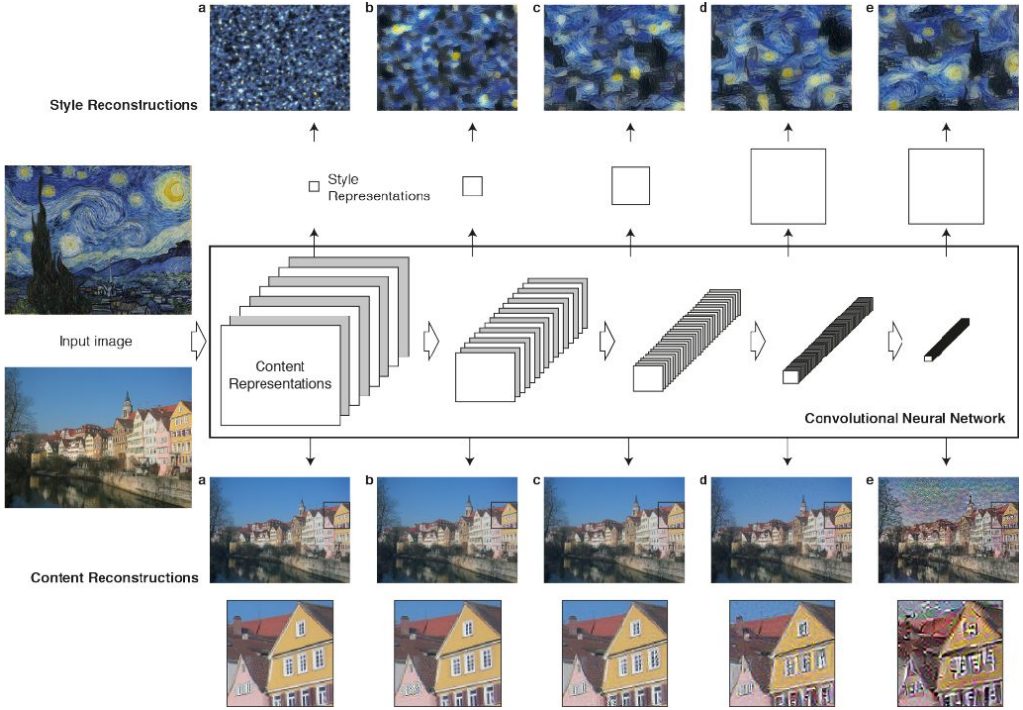
GNOMON DIGITAL

On peut déjà générer des images avec un modèle CNN

tjwhitaker/**a-neural-algorithm-of-artistic-...**

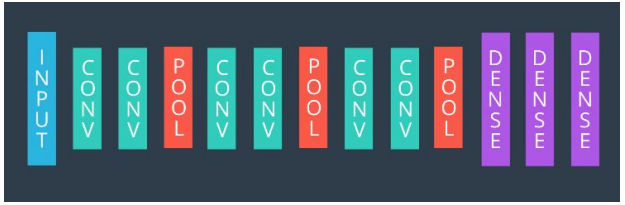
Neural Style Transfer

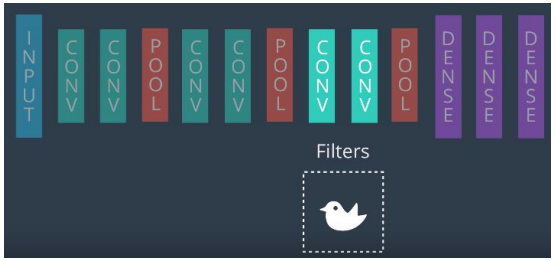
1 Contributor | 0 Issues | 4 Stars | 2 Forks

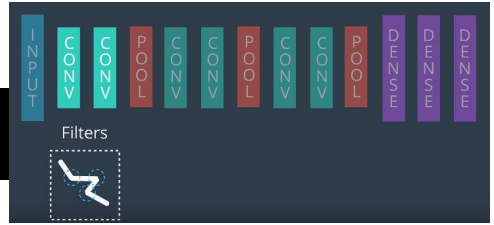


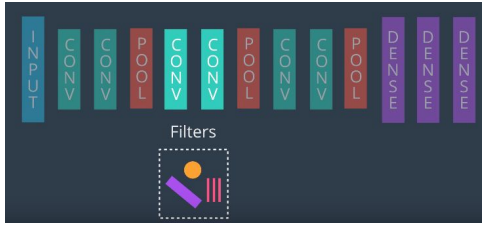
https://arxiv.org/abs/1508.06576

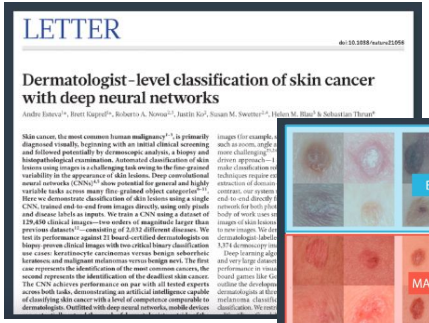
Transfer learning



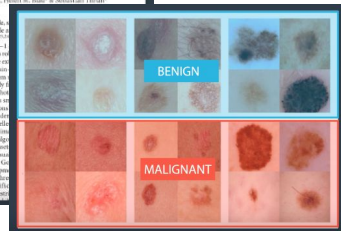


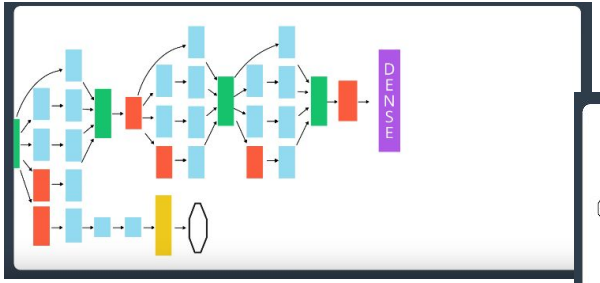


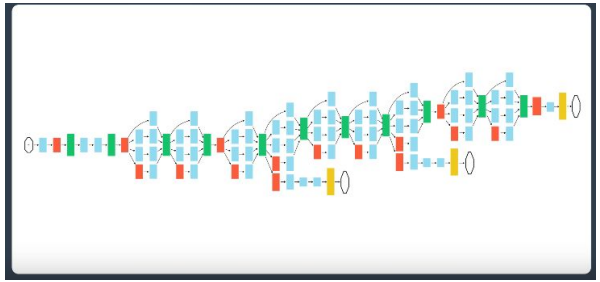


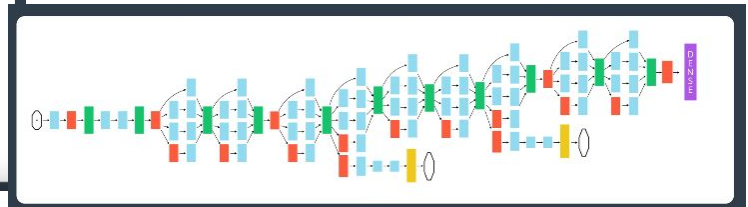

LETTER

Dermatologist-level classification of skin cancer with deep neural networks



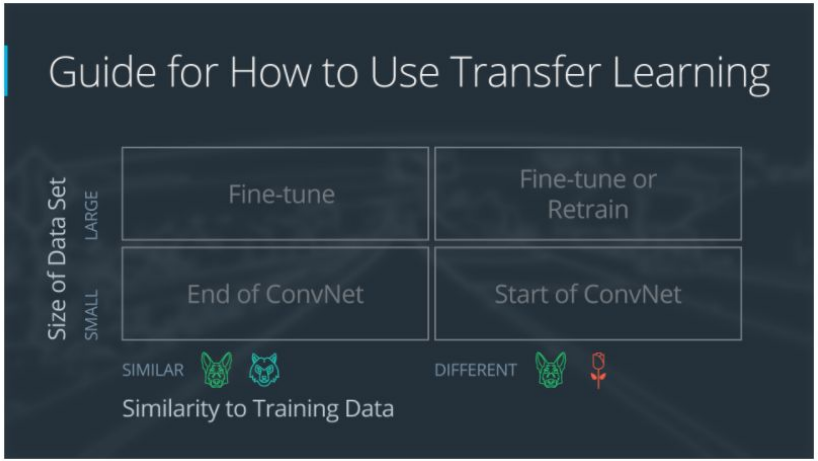


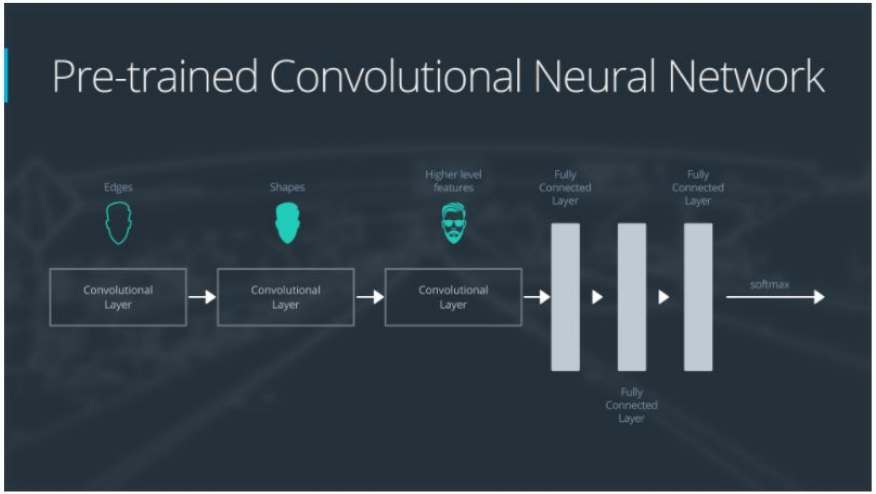




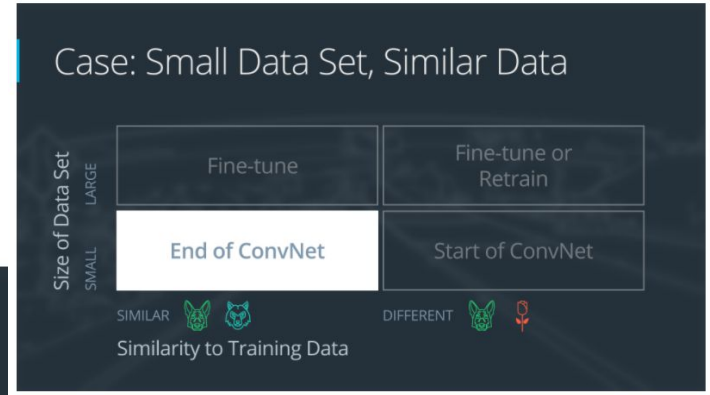
- Randomly initialize the weights in the new fully connected layer
- Initialize the rest of the weights using the pre-trained weights
- Re-train the entire neural network

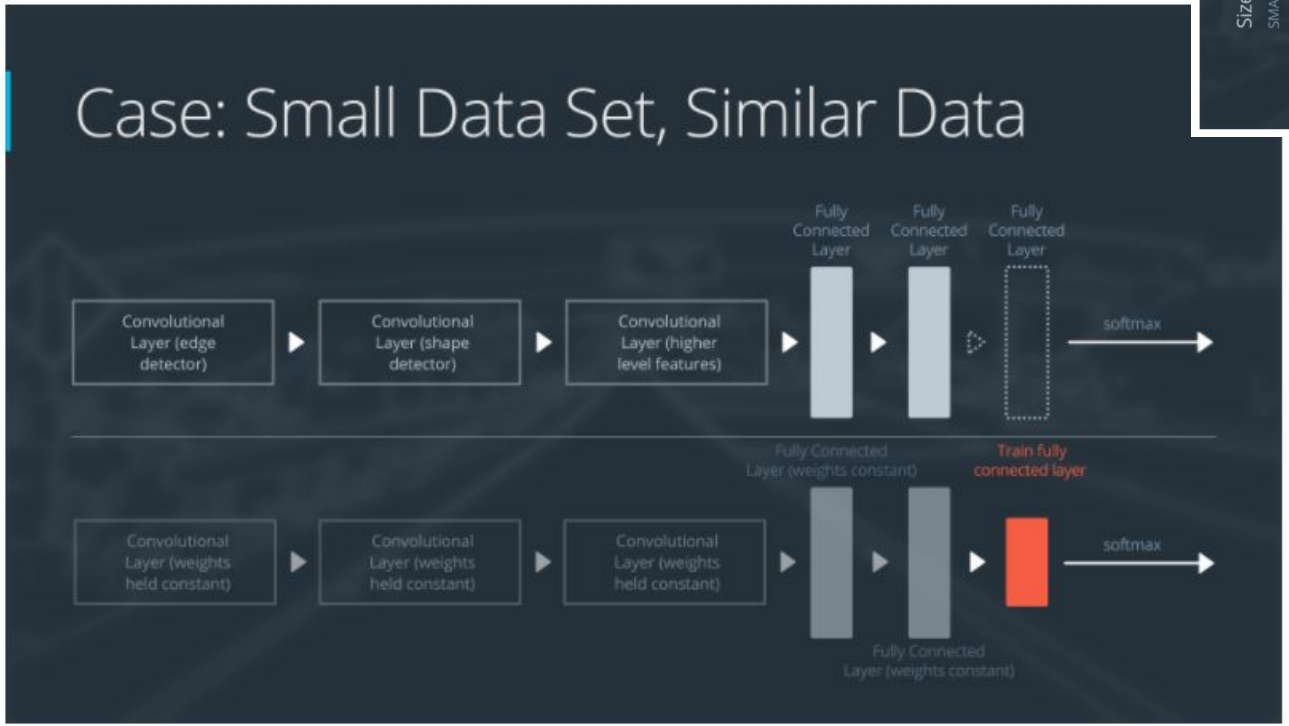
Transfer learning



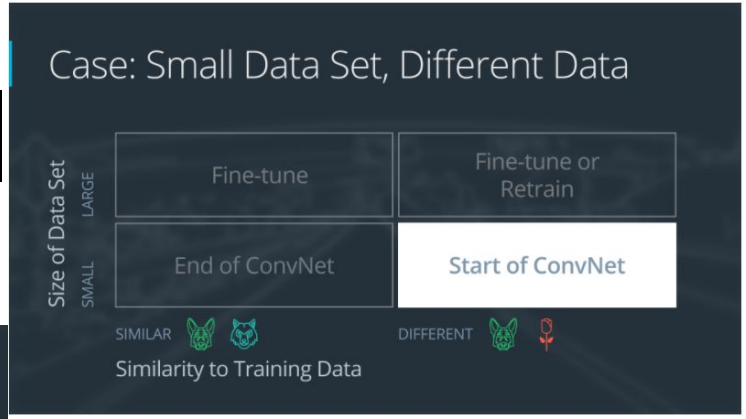


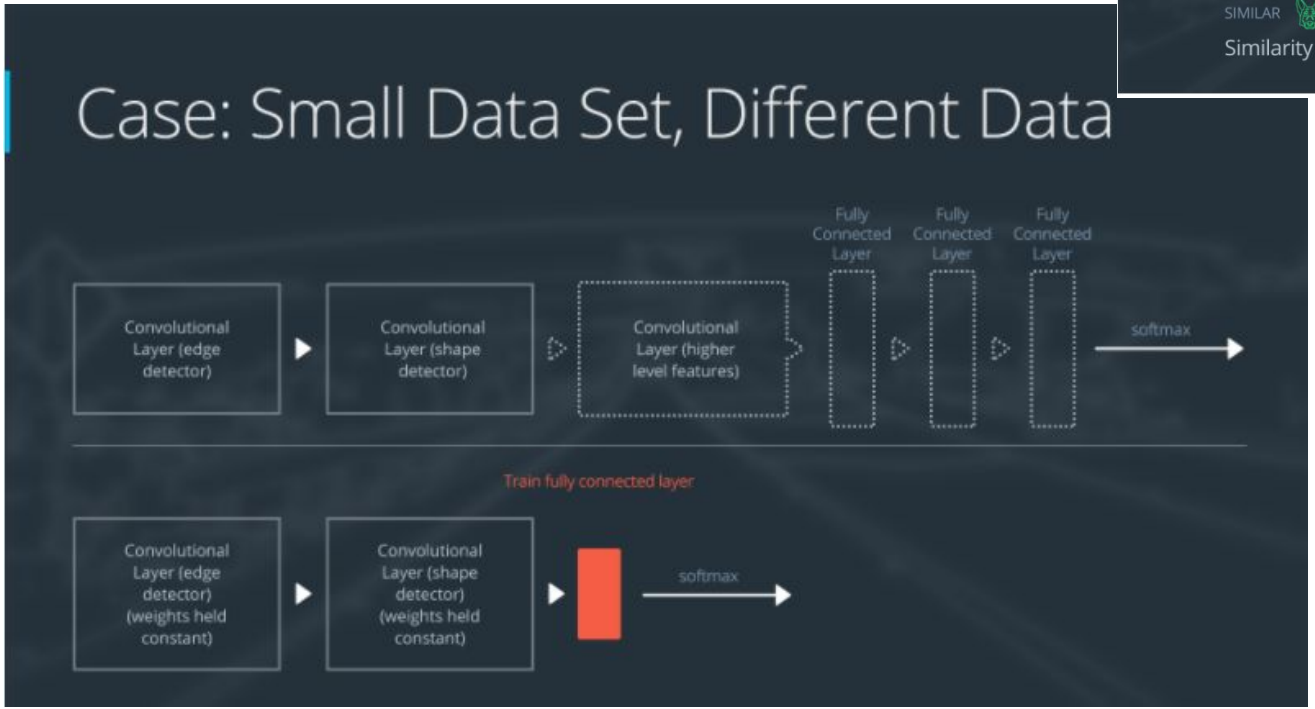
Transfer learning



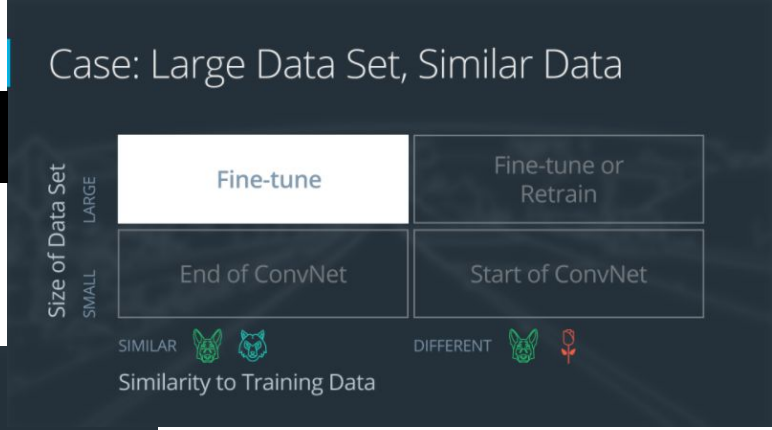


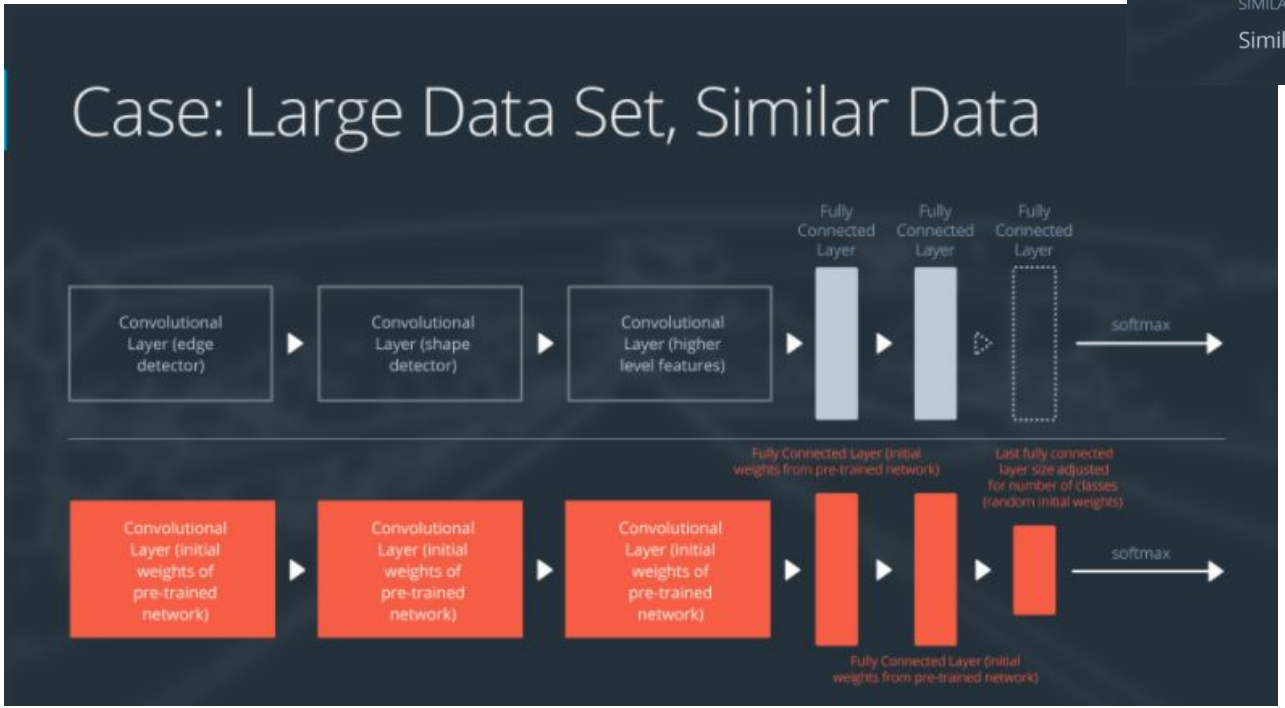
Transfer learning



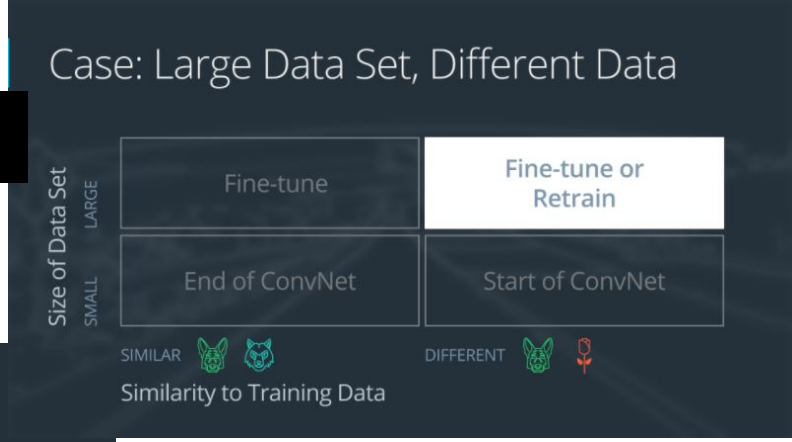


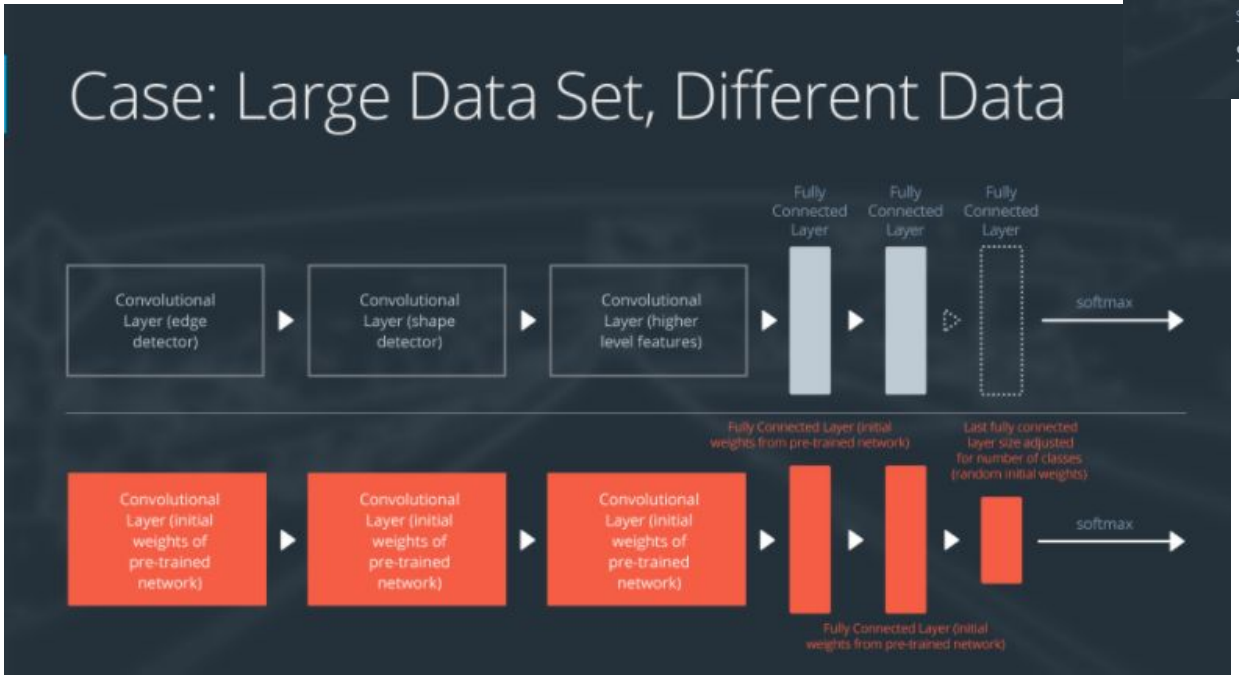
Transfer learning



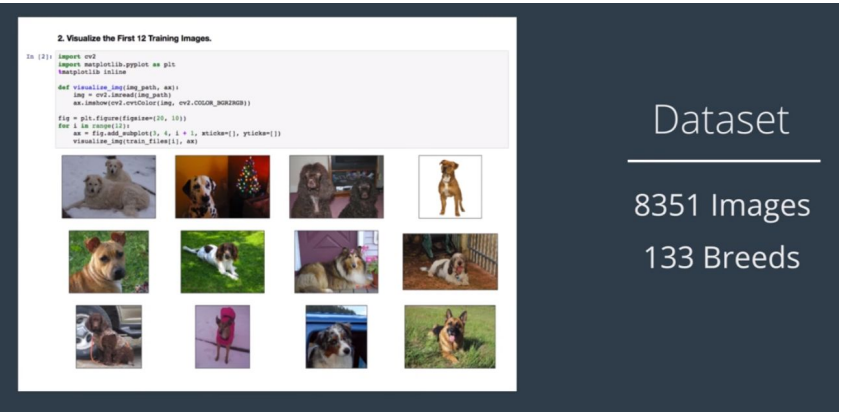


Transfer learning





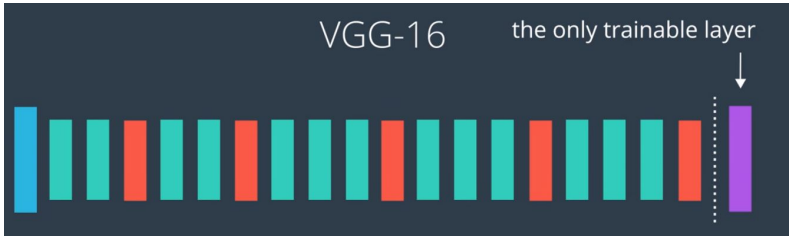
Transfer learning

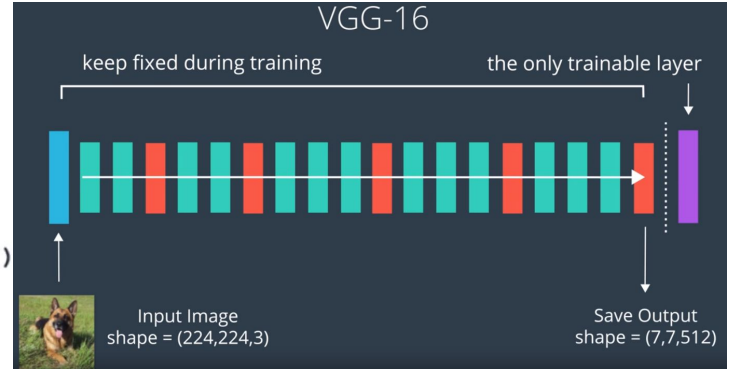


Dataset

8351 Images

133 Breeds





3. Obtain the VGG-16 Bottleneck Features

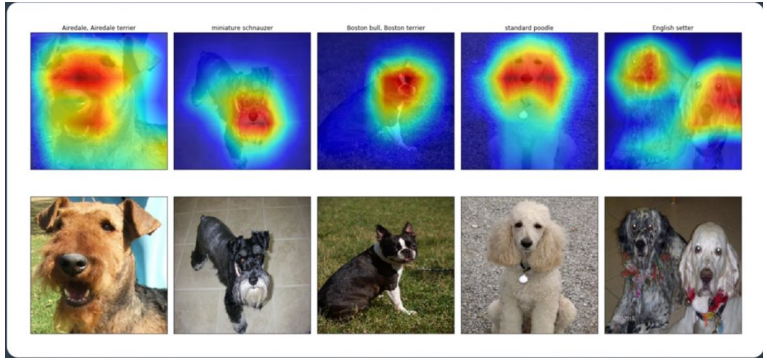
```
In [3]: bottleneck_features = np.load('bottleneck_features/DogVGG16Data.npz')
        train_vgg16 = bottleneck_features['train']
        valid_vgg16 = bottleneck_features['valid']
        test_vgg16 = bottleneck_features['test']
```

Transfer learning

```
from keras.layers import Dense, Flatten
from keras.models import Sequential

model = Sequential()
model.add(Flatten(input_shape=(7,7,512)))
model.add(Dense(133, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
              metrics=['accuracy'])
model.summary()
```

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 133)	3336837

Total params: 3,336,837.0
Trainable params: 3,336,837.0
Non-trainable params: 0.0




```
from keras.layers import GlobalAveragePooling2D

model = Sequential()
model.add(GlobalAveragePooling2D(input_shape=(7,7,512)))
model.add(Dense(133, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
global_average_pooling2d_1 (	(None, 512)	0
dense_2 (Dense)	(None, 133)	68229

Total params: 68,229.0
Trainable params: 68,229.0
Non-trainable params: 0.0

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Inspect Model

```
>>> model.output_shape              # Model output shape
>>> model.summary()                 # Model summary representation
>>> model.get_config()              # Model configuration
>>> model.get_weights()             # List all weight tensors in the model
```

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```


GNOMON DIGITAL®

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Inspect Model

```
>>> model.output_shape              # Model output shape
>>> model.summary()                 # Model summary representation
>>> model.get_config()              # Model configuration
>>> model.get_weights()             # List all weight tensors in the model
```

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

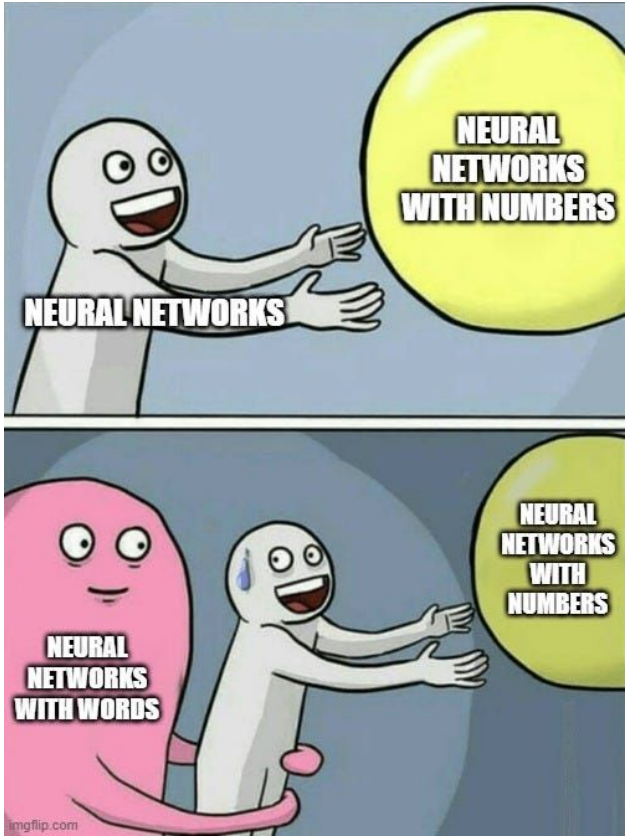
```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

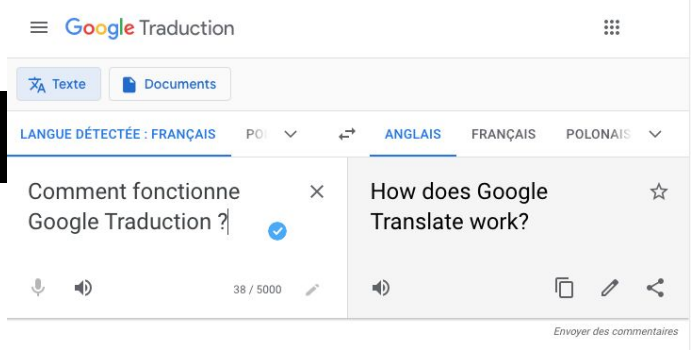
```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```


GNOMON DIGITAL®

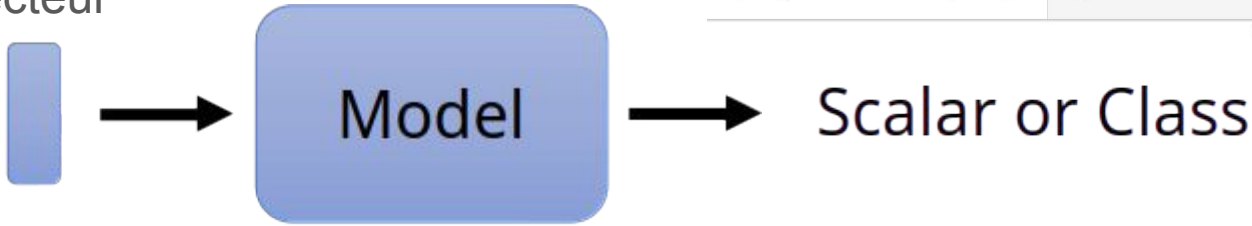
NLP


NEURAL NETWORKS
NEURAL NETWORKS WITH NUMBERS
NEURAL NETWORKS WITH WORDS
NEURAL NETWORKS WITH NUMBERS
imgflip.com

NLP

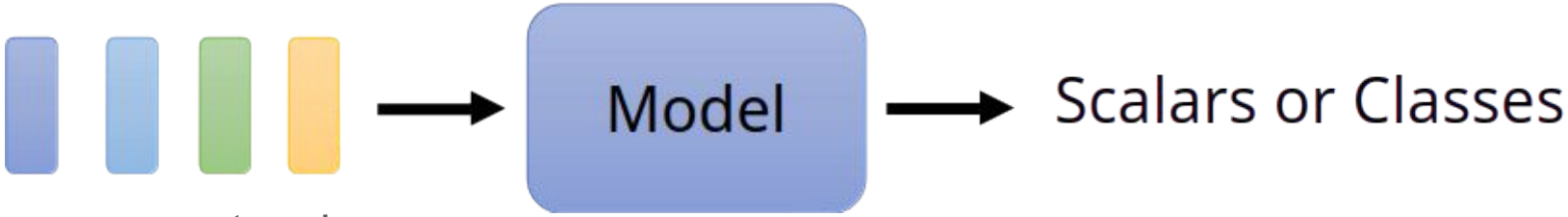


- L'entrée est un vecteur



Longueur fixé ou uniformisé

- L'entrée est une listes de vecteurs



Longueur peut varier

NLP

GNOMON DIGITAL

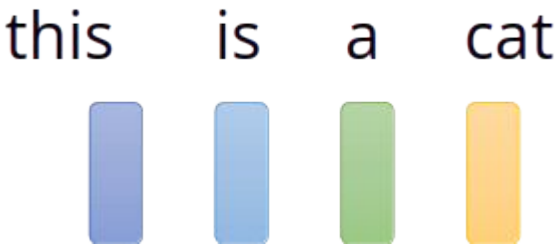
Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50



NLP

GNOMON DIGITAL

Label Encoding

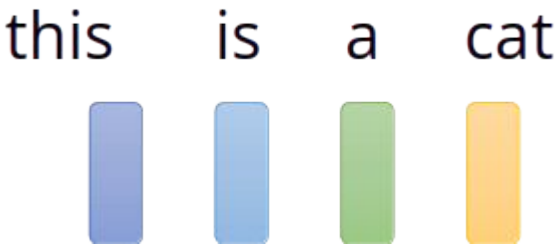
Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

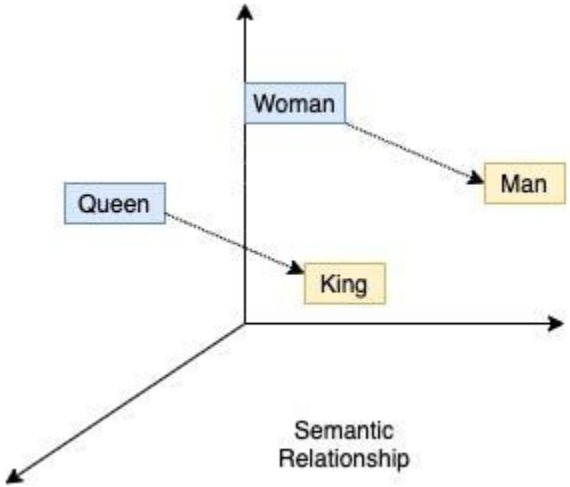
One Hot Encoding

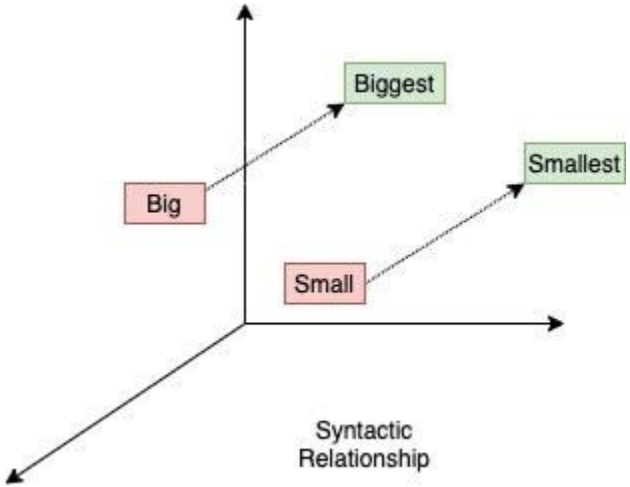
Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

PB: I saw a saw.



NLP



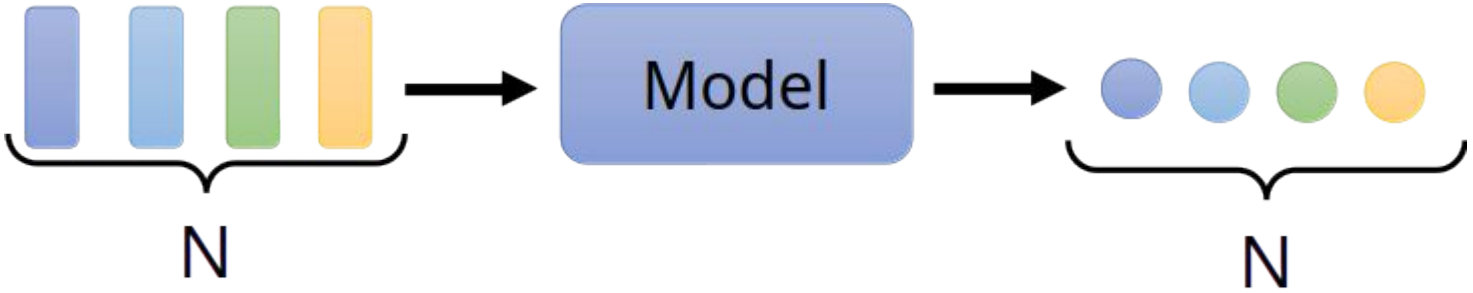


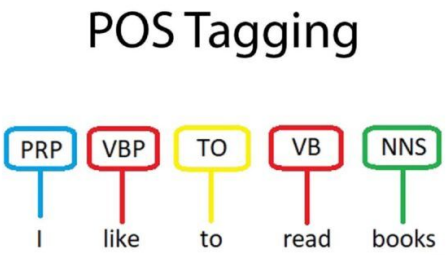
word2vec

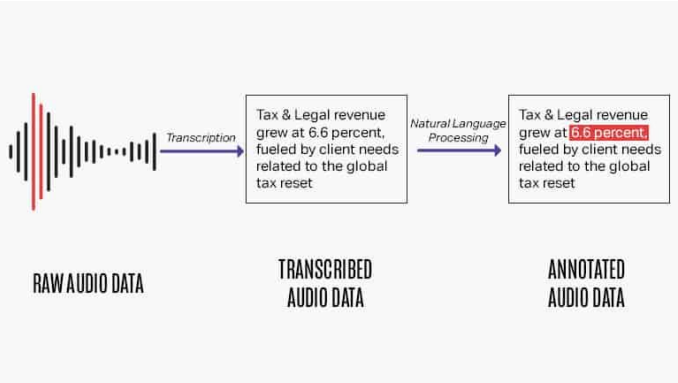
https://medium.com/@fraidoonomarzai99/word2vec-cbow-skip-gram-in-depth-88d9cc340a50

NLP

Chaque vecteur a un label

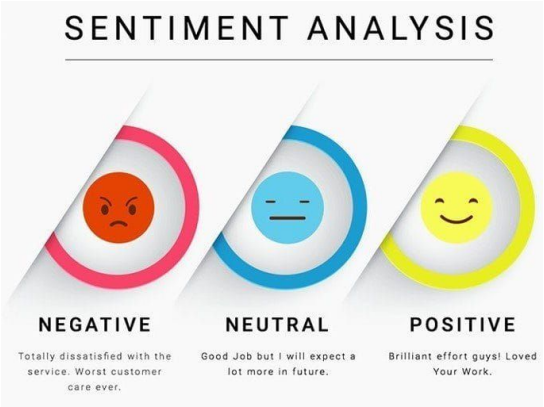


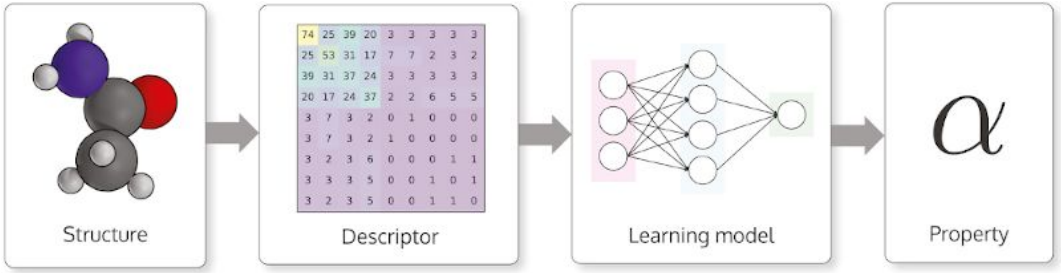




NLP

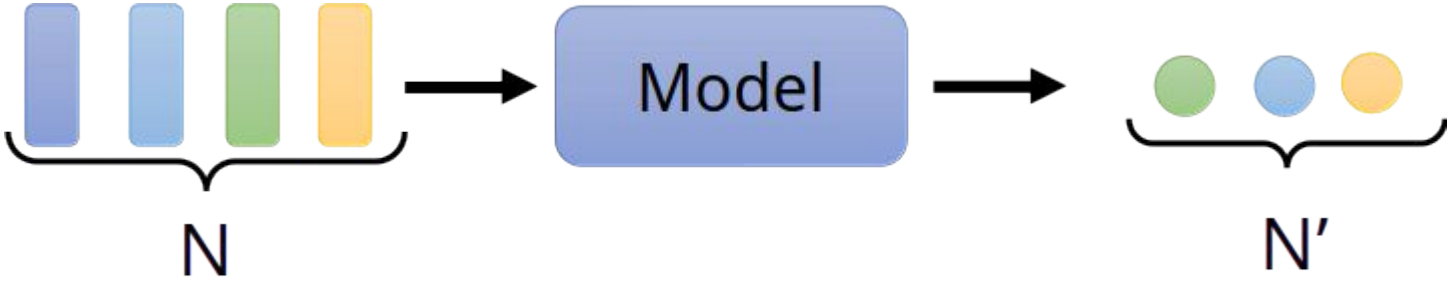
La séquence entière correspond un label.





NLP

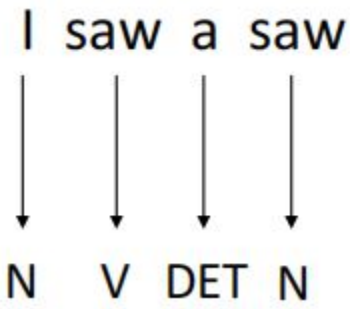
Le modèle va déterminer lui même la sortie (seq2seq)

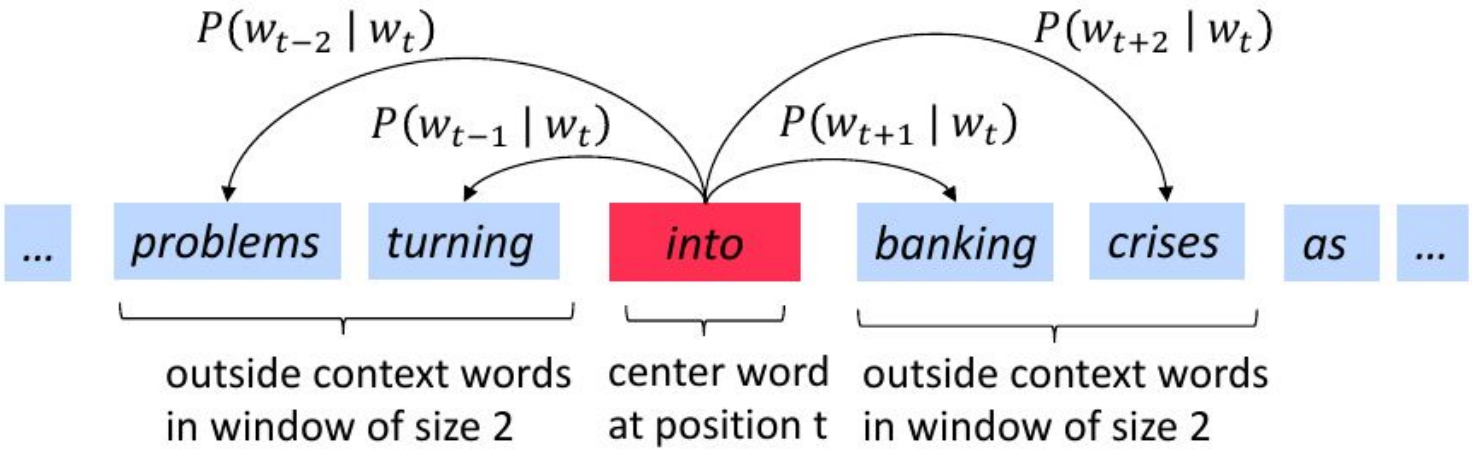


Google Traduction

ChatGPT

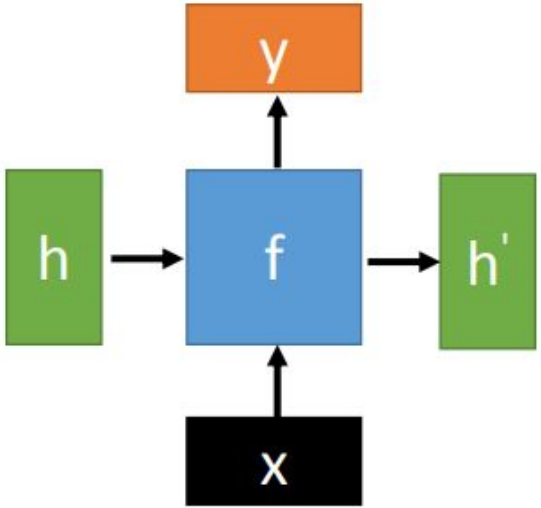
NLP


I saw a saw
N V DET N


$P(w_{t-2} \mid w_t)$
$P(w_{t+2} \mid w_t)$
$P(w_{t-1} \mid w_t)$
$P(w_{t+1} \mid w_t)$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

RNN: Recurrent Neural Networks

- Given function f: $h', y = f(h, x)$

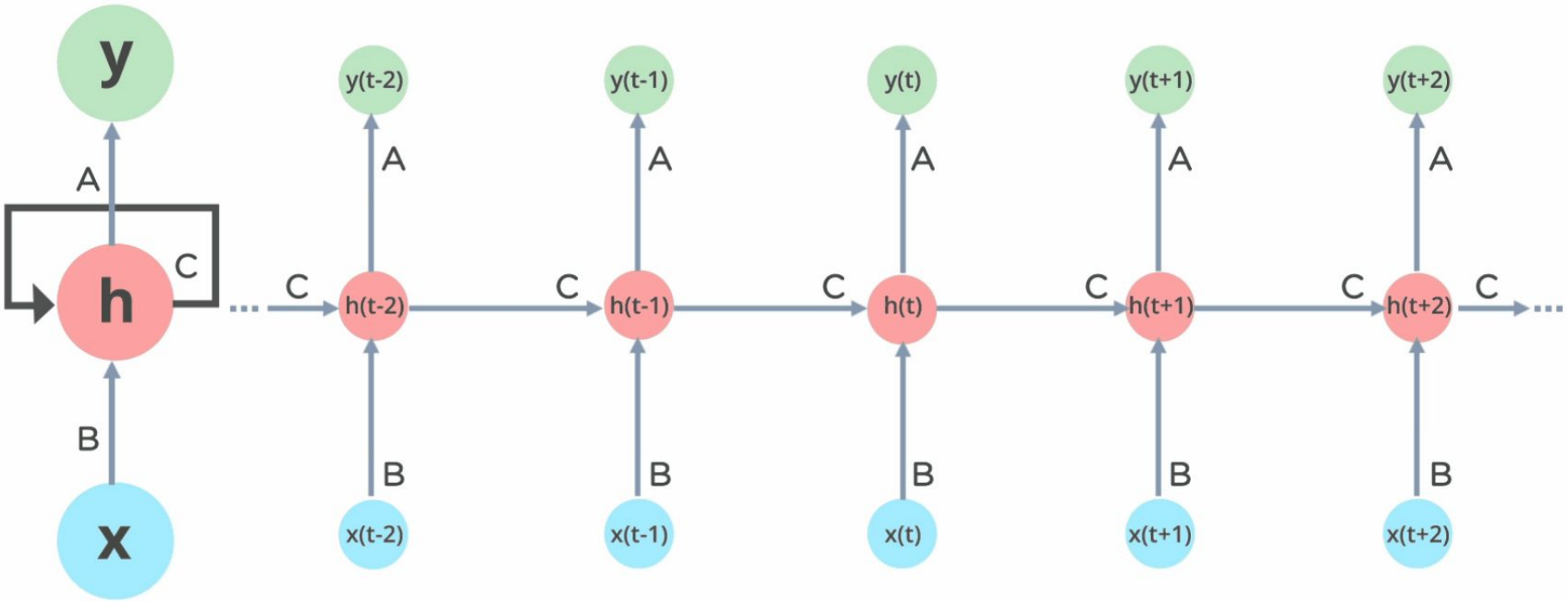


$$h' = \sigma(W^h h + W^i x)$$

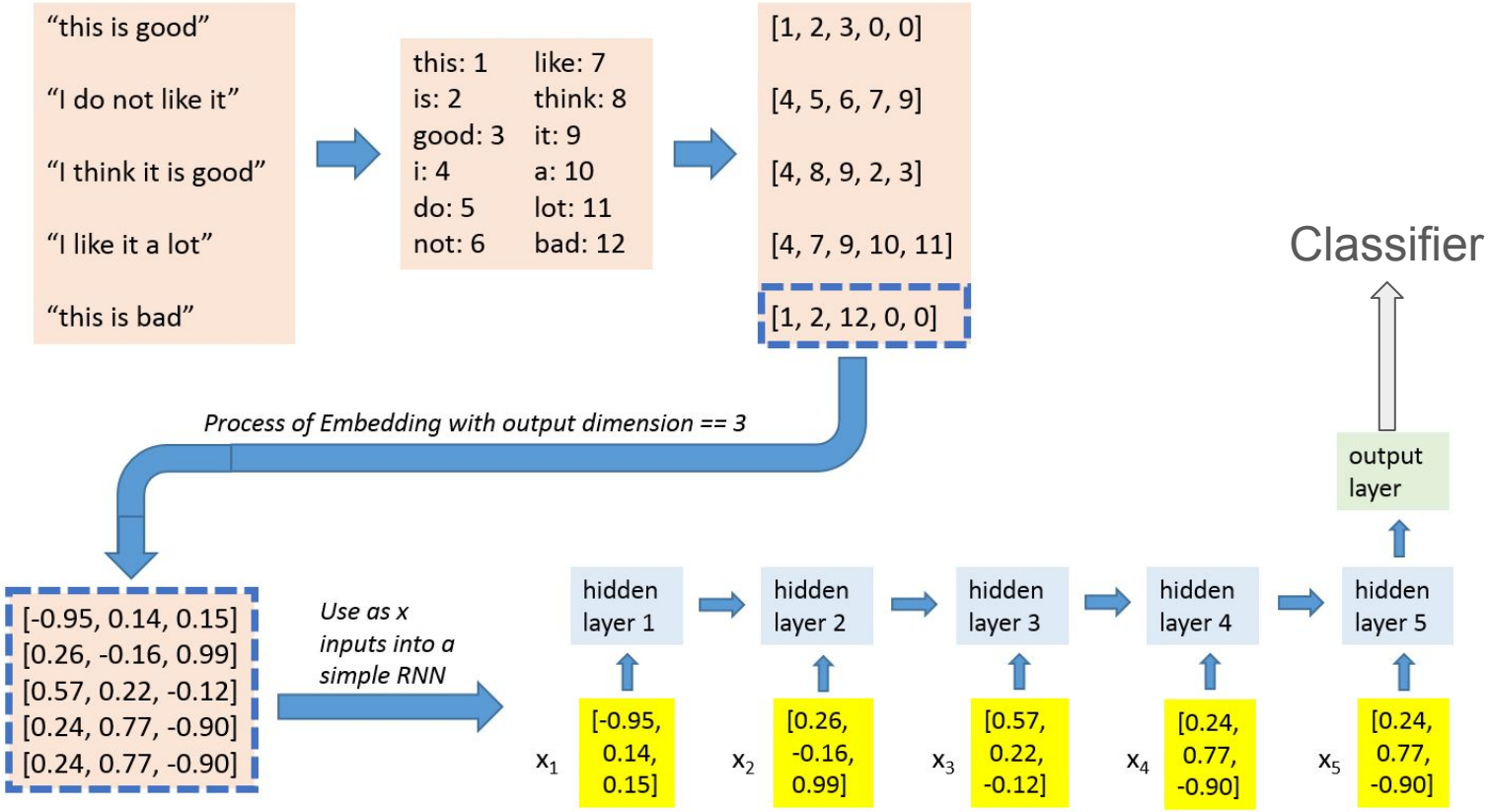
$$y = \sigma(W^o h')$$

softmax

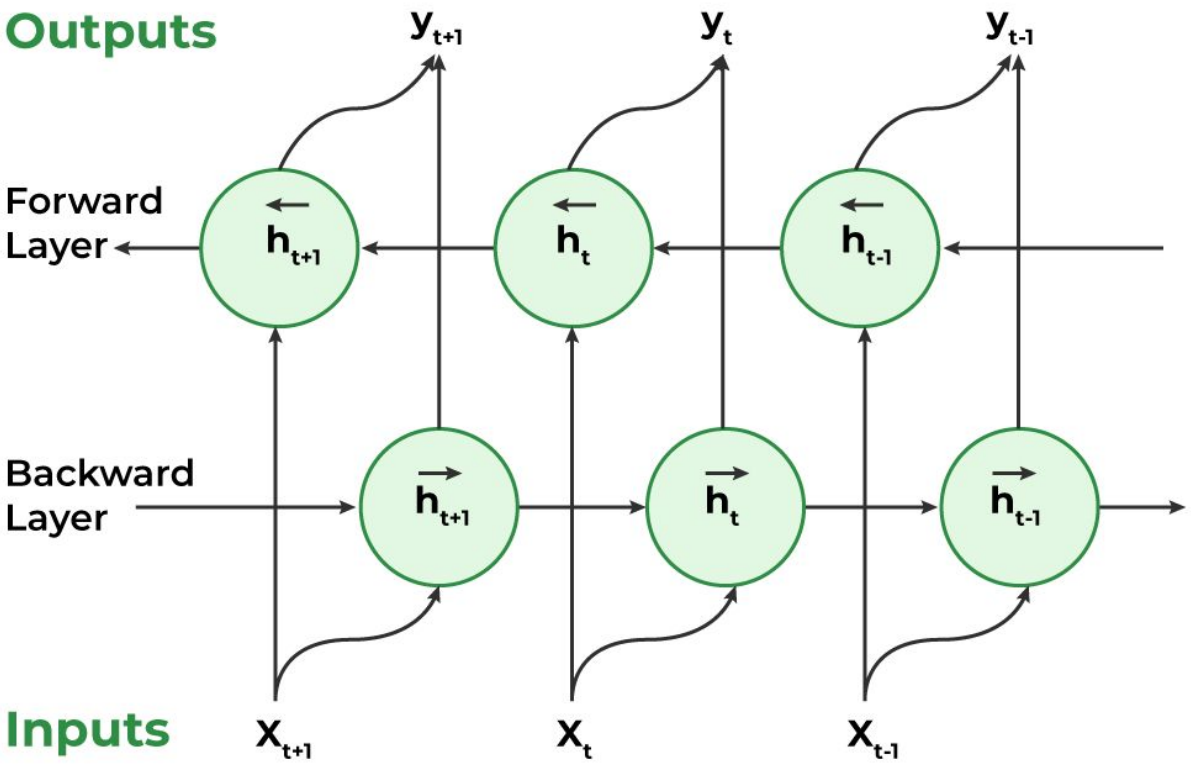
RNN: Recurrent Neural Networks



RNN: Recurrent Neural Networks



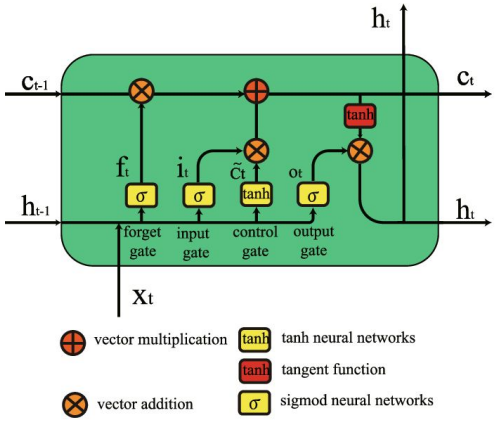
RNN: Recurrent Neural Networks

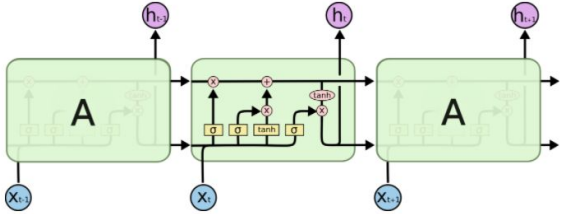


Bidirectional RNN

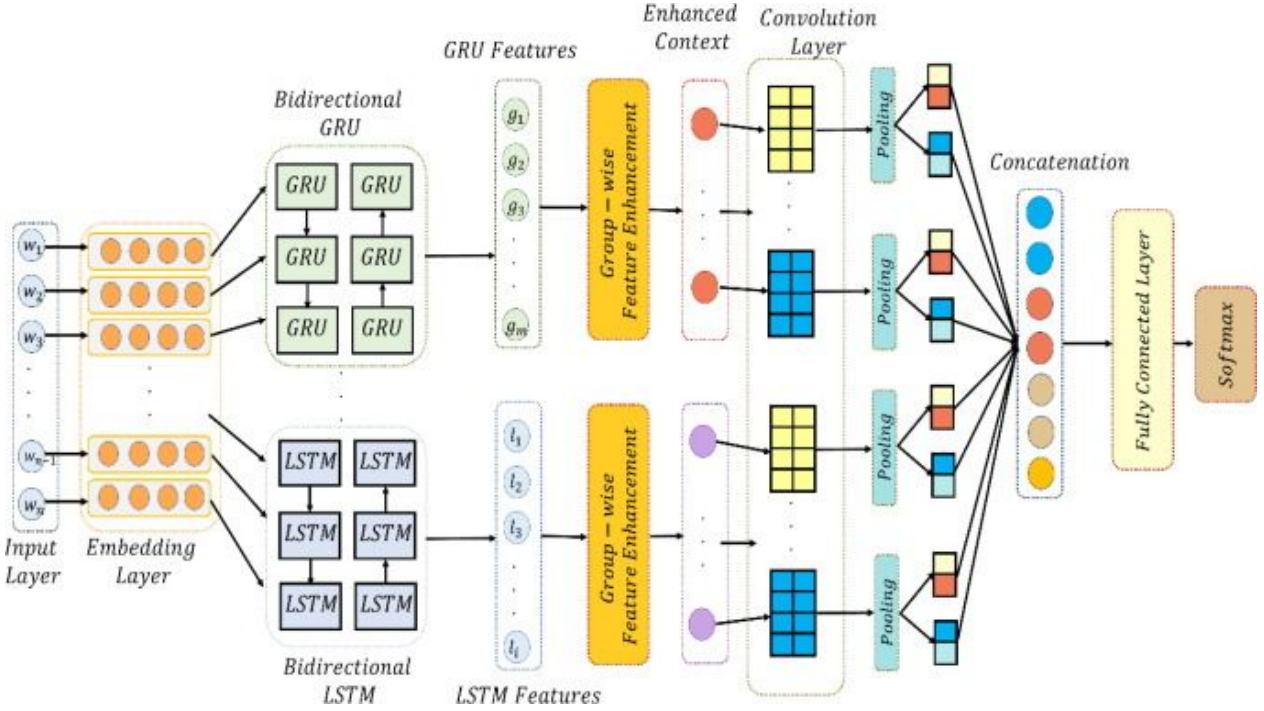
RNN: Recurrent Neural Networks

Keras peut gérer pour toi

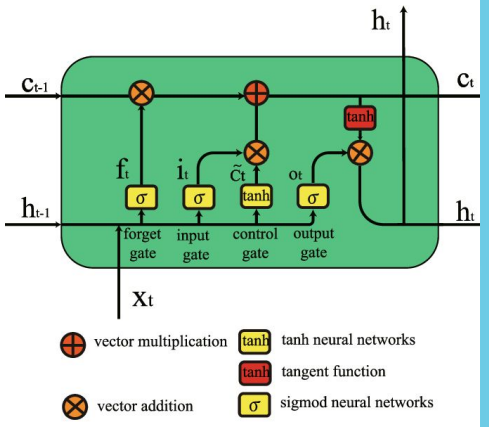


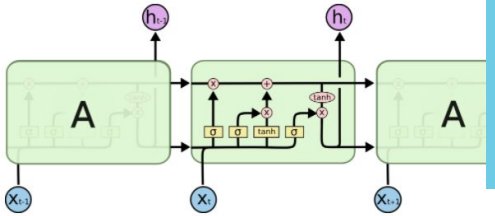


The repeating module in an LSTM contains four interacting layers.

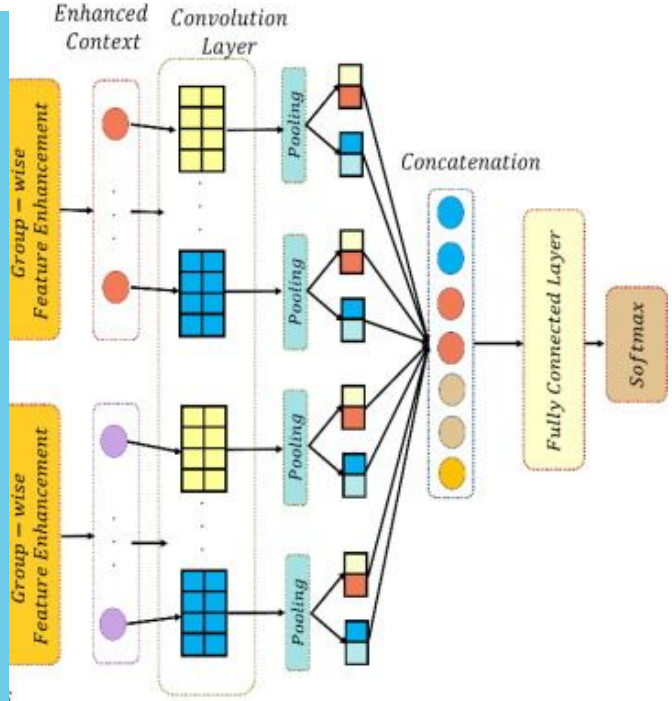


RNN: Recurrent Neural Networks





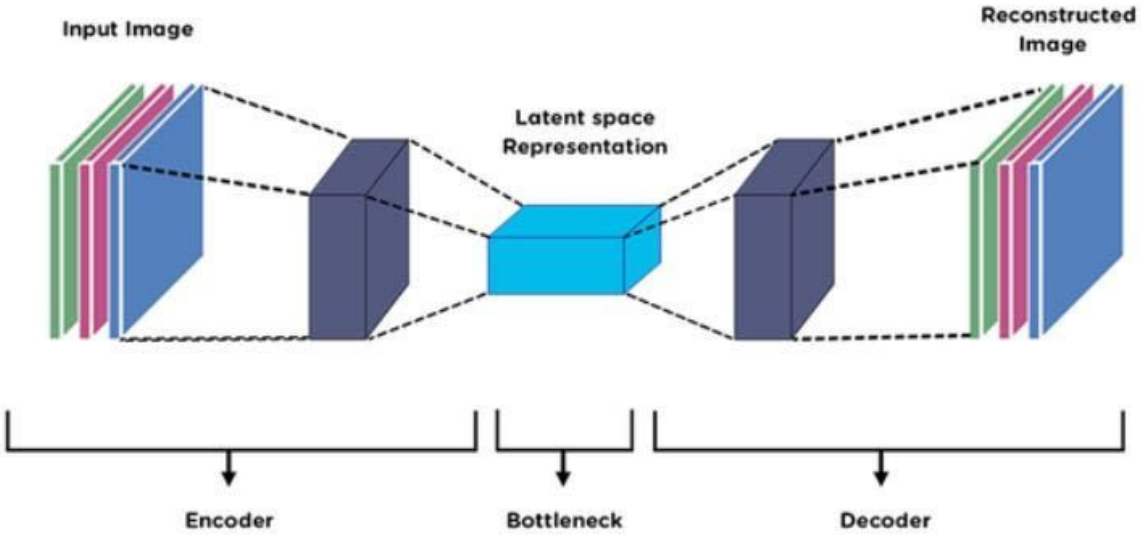
The repeating module in an LSTM contains four interacting layers.



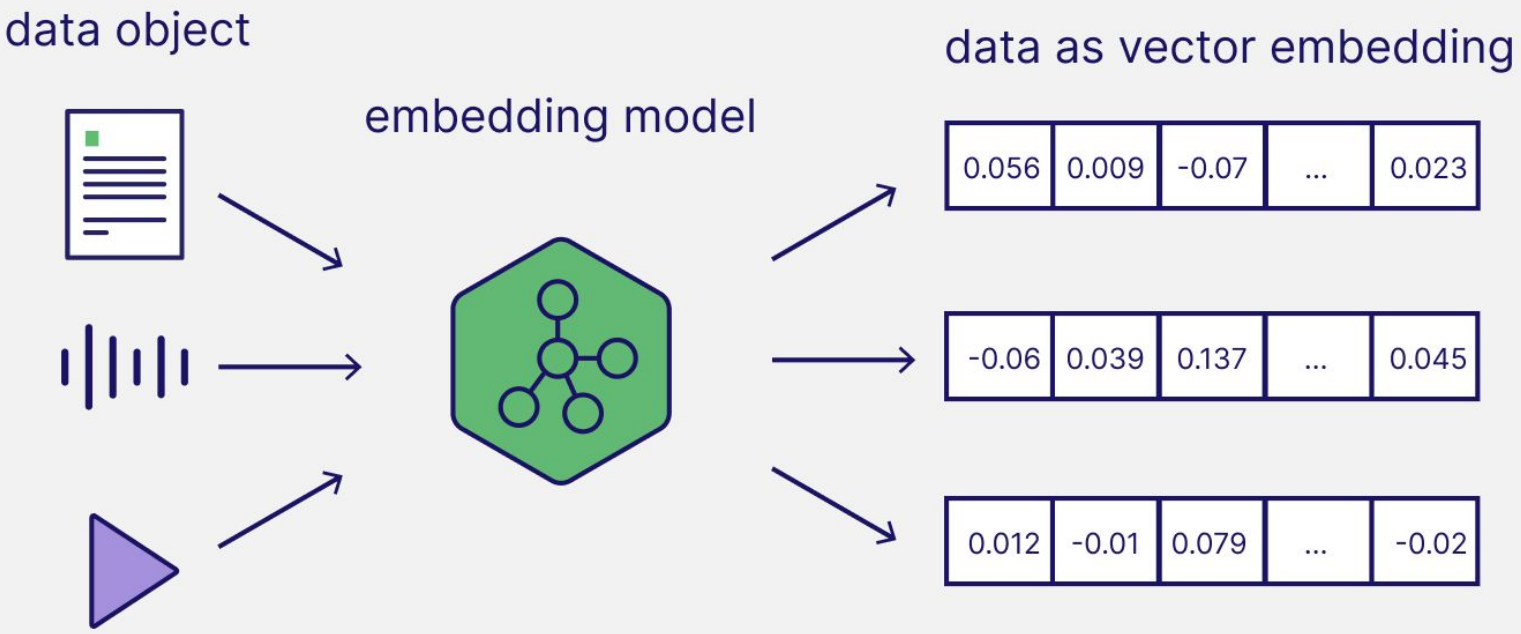
S2S

Sequence-to-sequence (Encoder-Decoder Architecture)

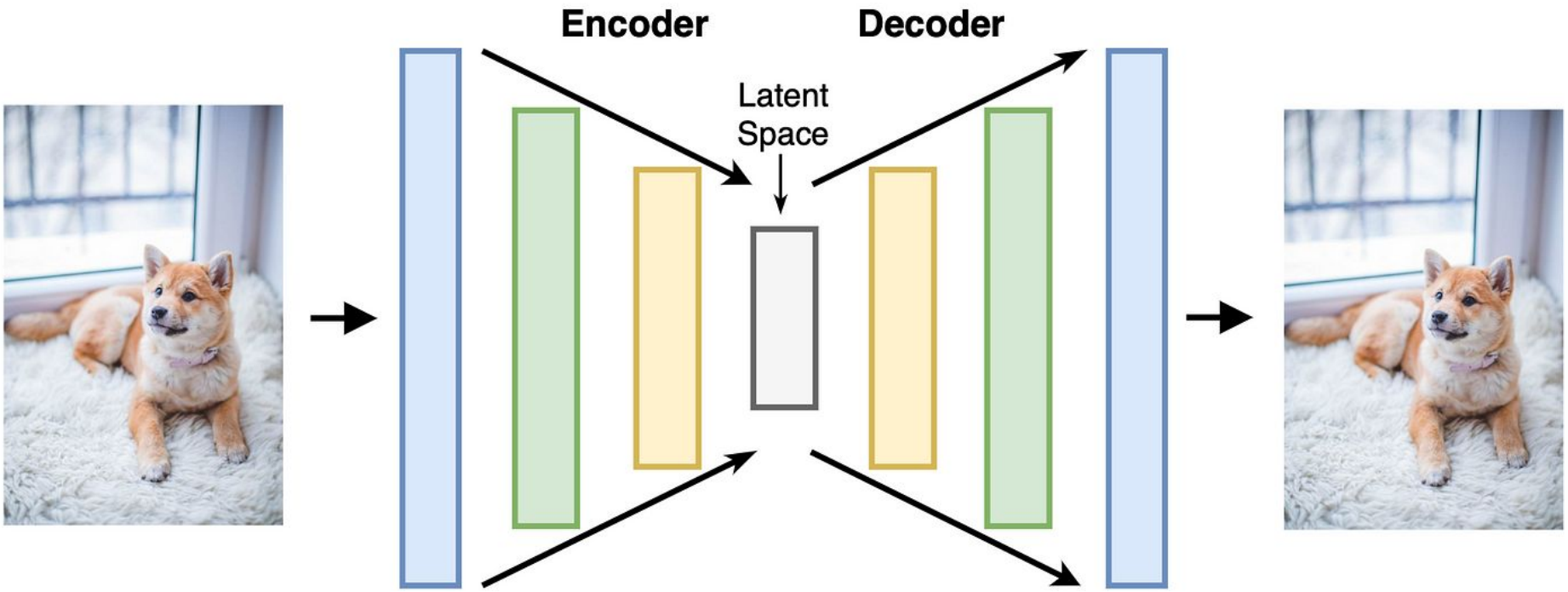
Les entrées et les sorties sont des séquences avec des différentes longueurs (token).



Embedding

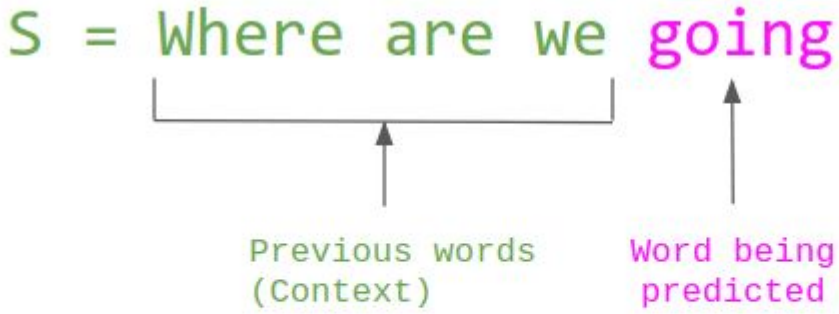

data object
embedding model
data as vector embedding
0.056
0.009
-0.07
...
0.023
-0.06
0.039
0.137
...
0.045
0.012
-0.01
0.079
...
-0.02

Embedding



Embedding

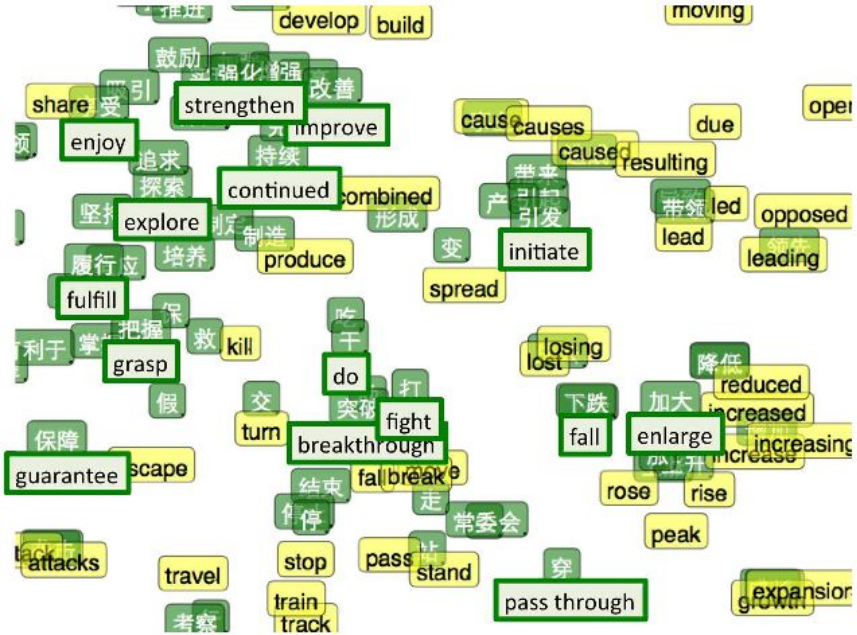
Language model



P(S) = P(Where) x P(are | Where) x P(we | Where are) x P(going | Where are we)

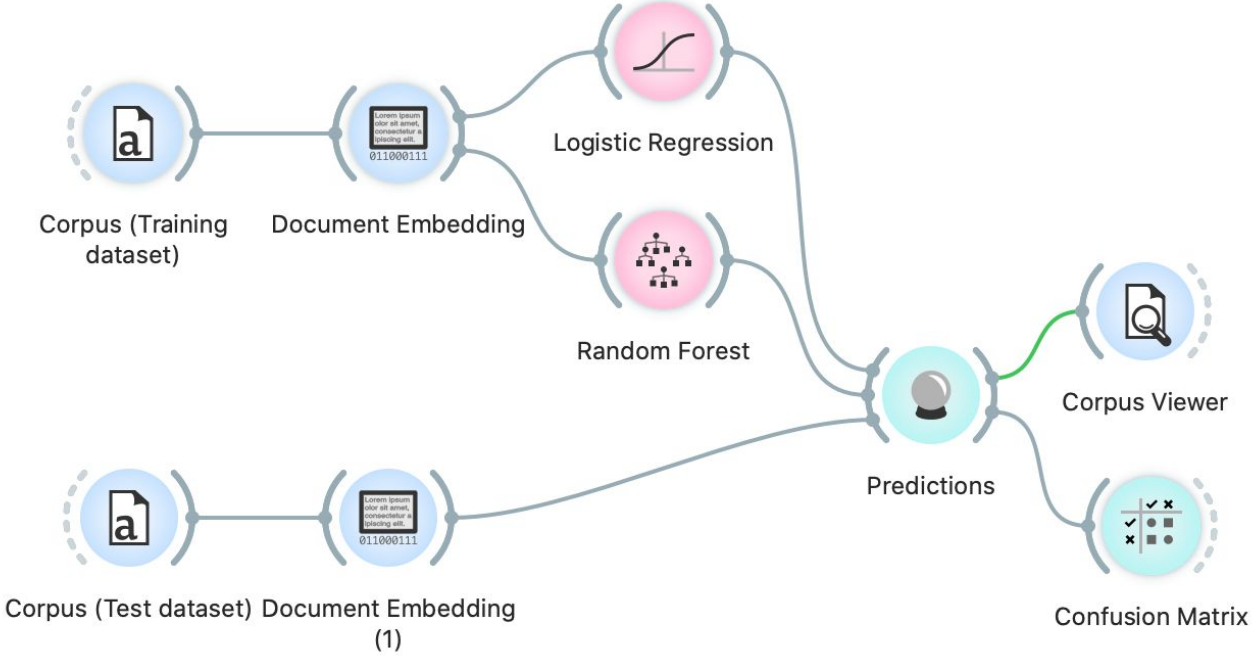
Embedding

Multi-lingual Embedding



Embedding

Document Embedding



Embedding

GNOMON DIGITAL

MTEB

Massive Text Embedding Benchmark

8 Tasks

58 Datasets

Clustering

- ArxivP2P
- ArxivS2S
- BiorxivP2P
- BiorxivS2S
- MedrxivP2P
- MedrxivS2S
- Reddit
- RedditP2P
- StackExchange
- StackExchangeP2P
- TwentyNewsgroup

Bitext Mining

- BUCC
- Tatoeba

Retrieval

Beir

- ArguAna
- ClimateFEVER
- DBPedia
- CQADupstackRetrieval
- FEVER
- FiQA2018
- HotpotQA
- MSMARCO
- NFCorpus
- NQ
- Quora
- SCIDOCS
- SciFact
- Touche2020
- TRECCOVID

STS

- BIOSESS
- SICK-R
- STS11
- STS12
- STS13
- STS14
- STS15
- STS16
- STS17
- STS22
- STSB

Summarization

- SummEval

Classification

- AmazonCounterfactual
- AmazonPolarity
- AmazonReviews
- Banking77
- Emotion
- Imdb
- MassiveIntent
- MassiveScenario
- MTOPDomain
- MTOPIntent
- ToxicConversations
- TweetSentimentExtraction

Pair Classification

- SprintDuplicateQuestions
- TwitterSemEval2015
- TwitterURLCorpus

Reranking

- AskUbuntuDupQuestions
- MindSmallReranking
- SciDocsRR
- StackOverFlowDupQuestions

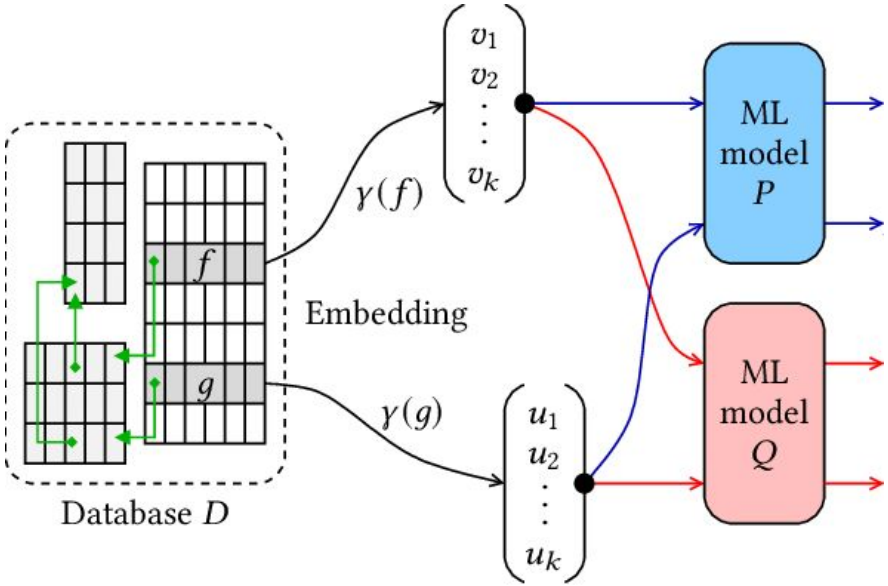
Embedding

GNOMON DIGITAL®

Hugging Face

Merci à Hugging face

Embedding

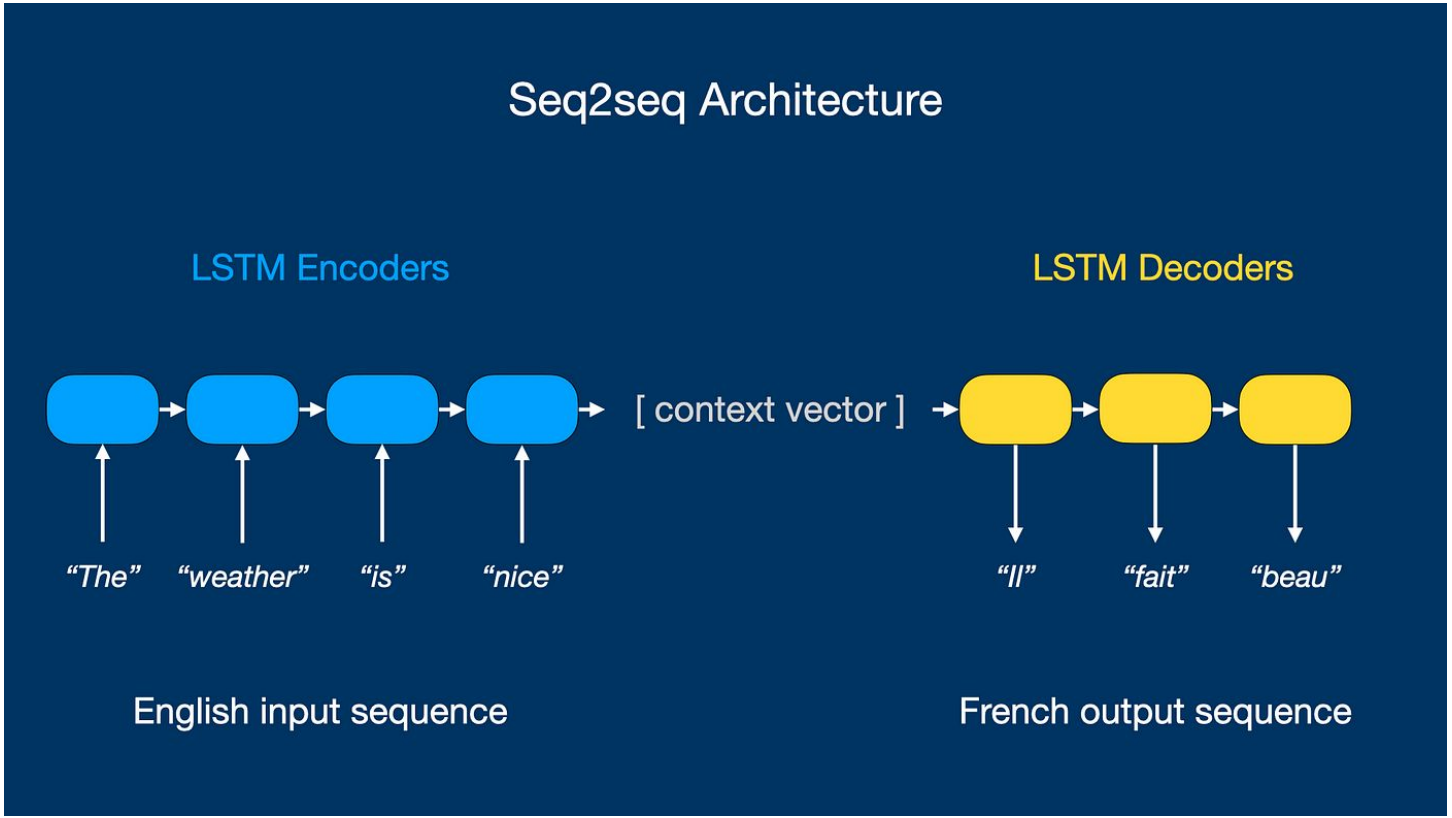


Hugging Face

Merci à Hugging face

https://huggingface.co/blog/getting-started-with-embeddings

S2S


Seq2seq Architecture
LSTM Encoders
LSTM Decoders
[context vector]
"The"
"weather"
"is"
"nice"
"Il"
"fait"
"beau"
English input sequence
French output sequence

S2S

GNOMON DIGITAL

Tokens	Characters
113	381


```
{"role": "assistant", "content": null, "function_call": {"name": "currentWeather", "arguments": "{ \"latitude\": \"37.773972\", \"longitude\": \"-122.431297\"}"}},
{"role": "function", "name": "currentWeather", "content": "{\"temperature\": "73", \"unit\": \"fahrenheit\", \"description\": \"Mostly sunny. High near 73, with temperatures falling to around 68 in the afternoon.\"}"}
```


https://tokens-lpj6s2duga-ew.a.run.app/

Self Attention

Chat Generative **P**re-trained **T**ransformer -> Transformer

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

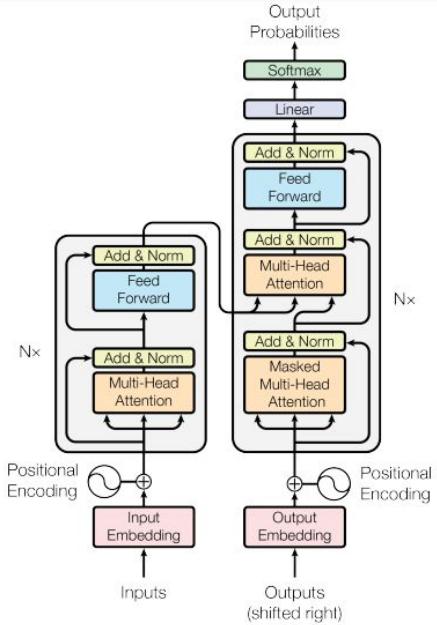
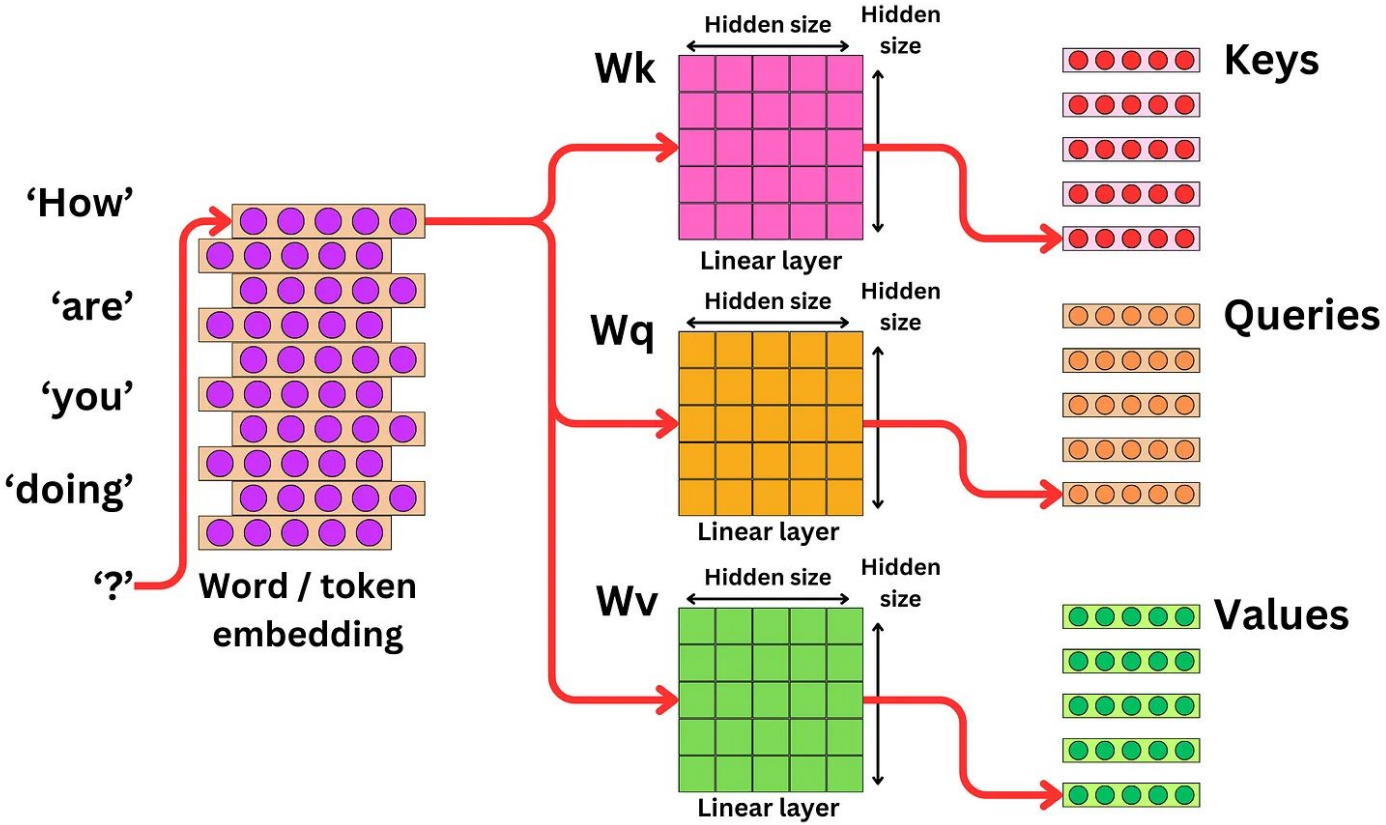
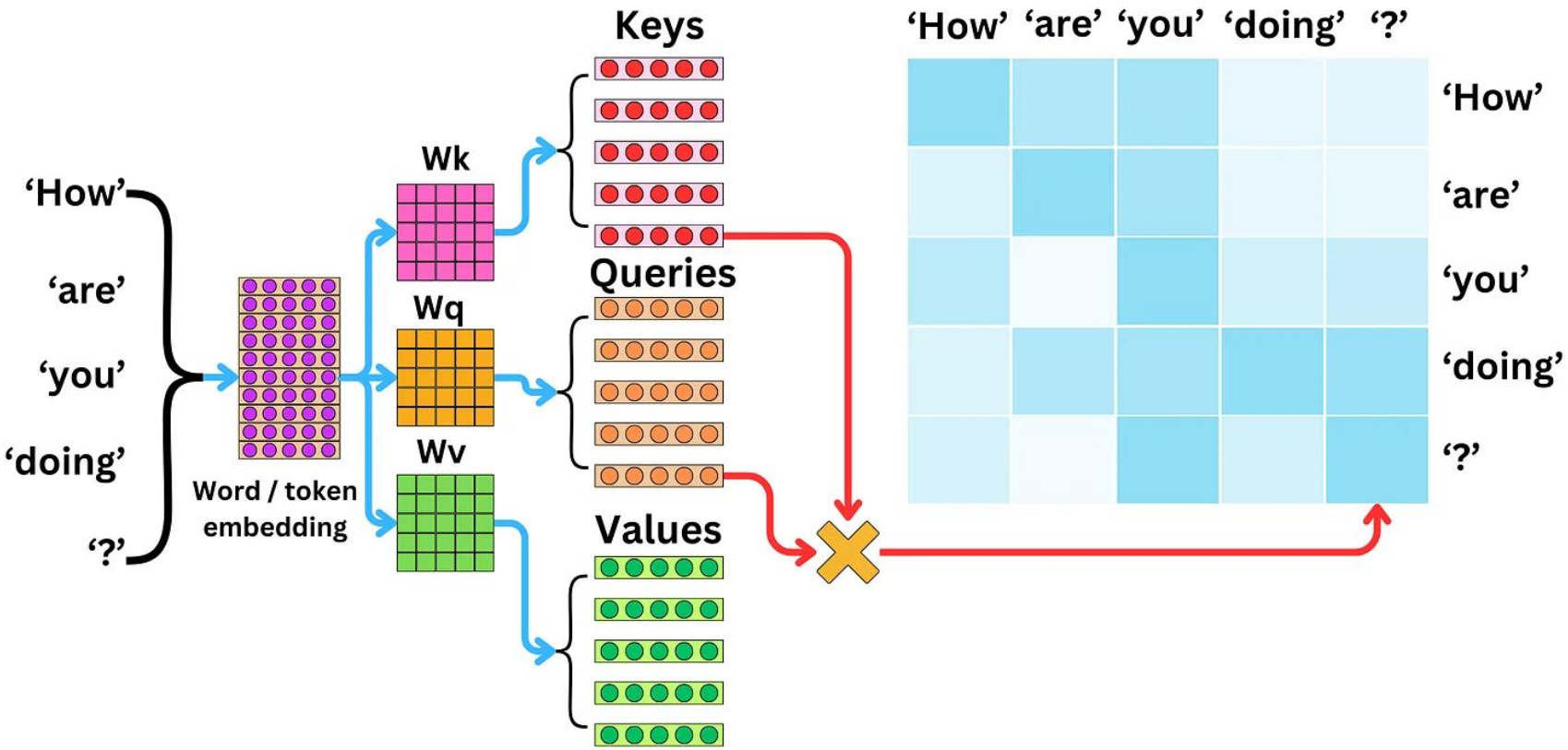


Figure 1: The Transformer - model architecture.

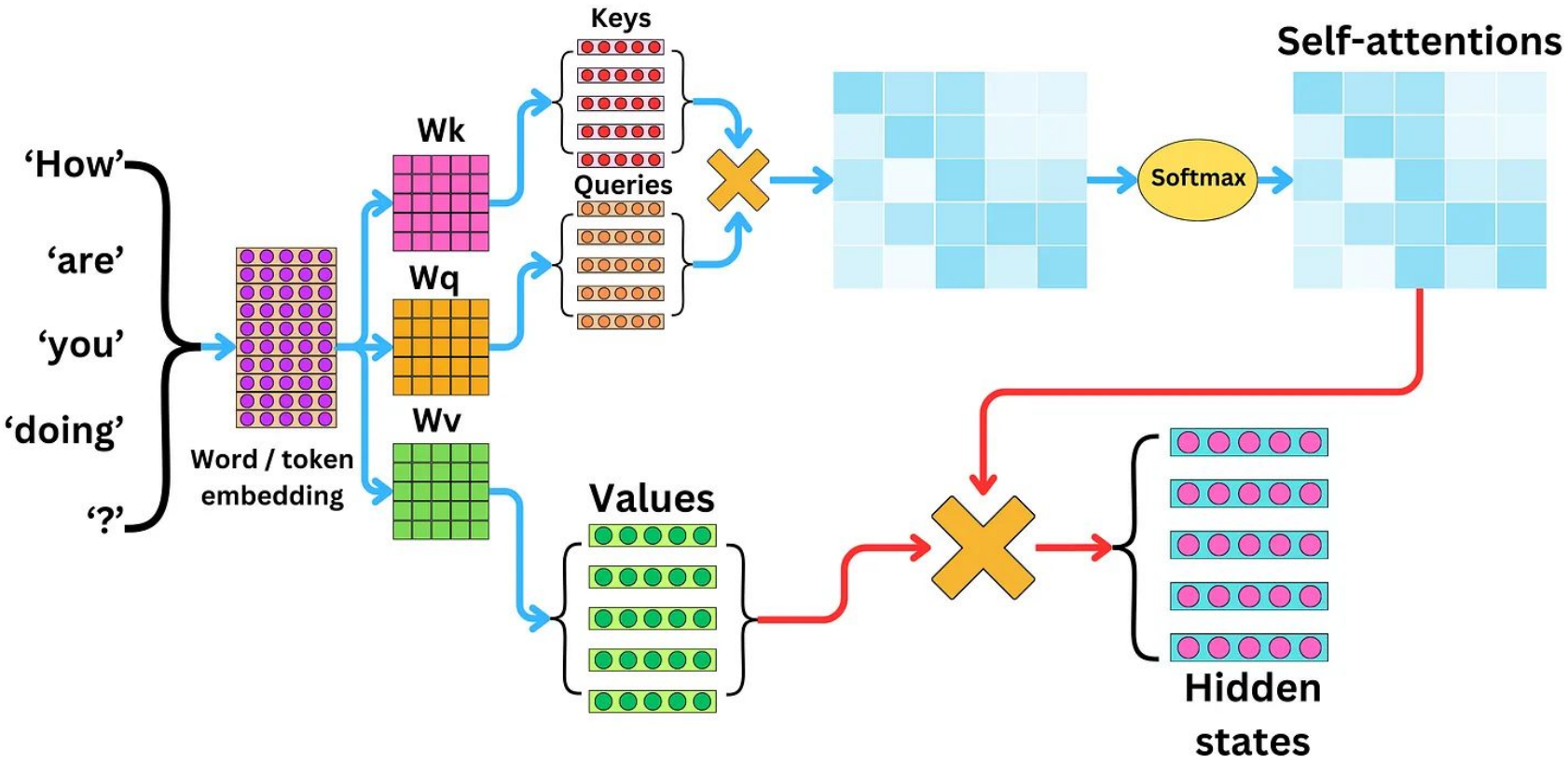
Self Attention



Self Attention

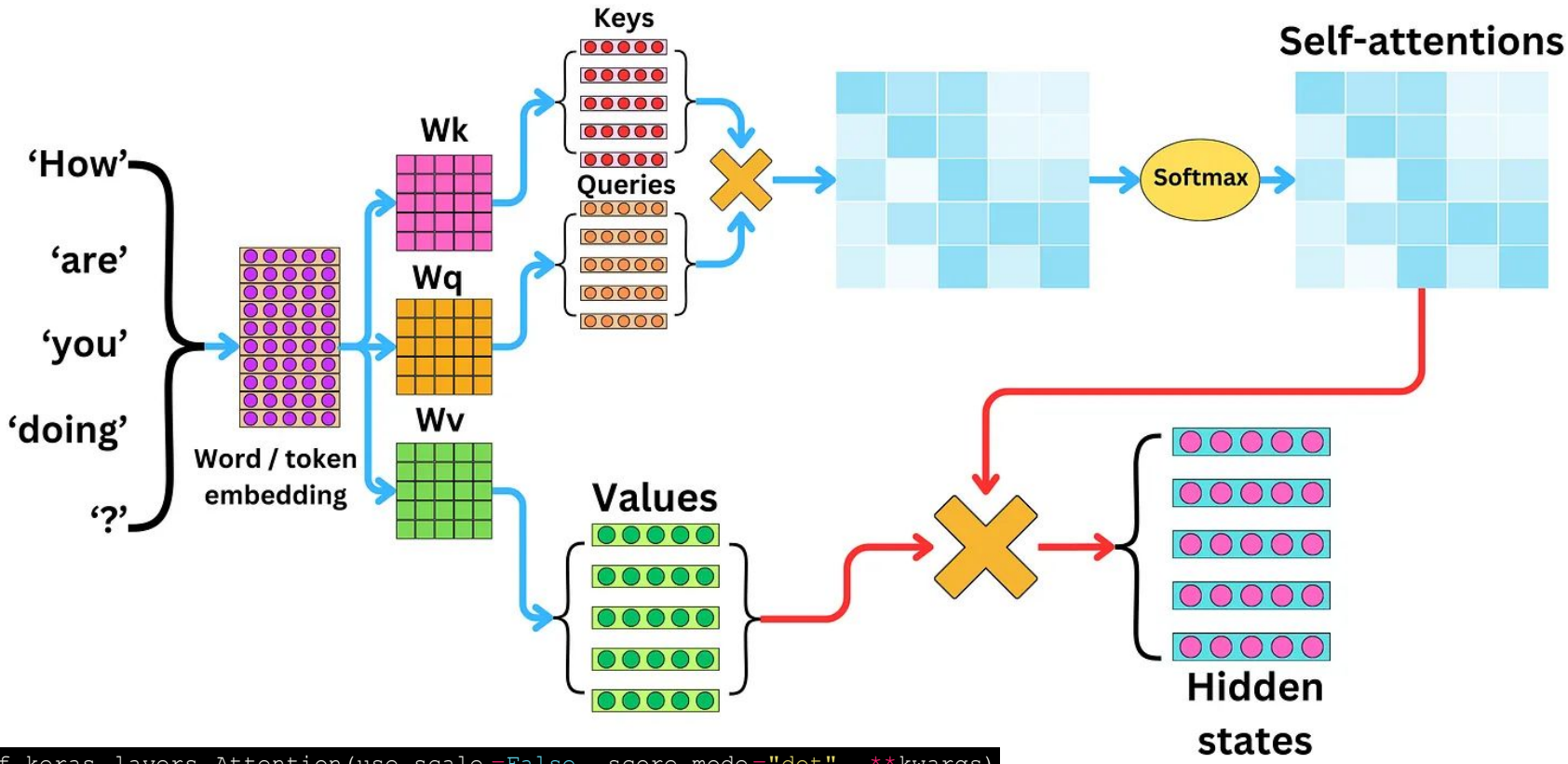


Self Attention



https://keras.io/2.17/api/layers/attention_layers/attention/

Self Attention




```
tf.keras.layers.Attention(use_scale=False, score_mode="dot", **kwargs)
```

Self Attention

Transformer

https://arxiv.org/abs/1706.03762

BERT

https://arxiv.org/abs/1810.04805

Self Attention

Chat Generative **P**re-trained **T**ransformer -> Transformer

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

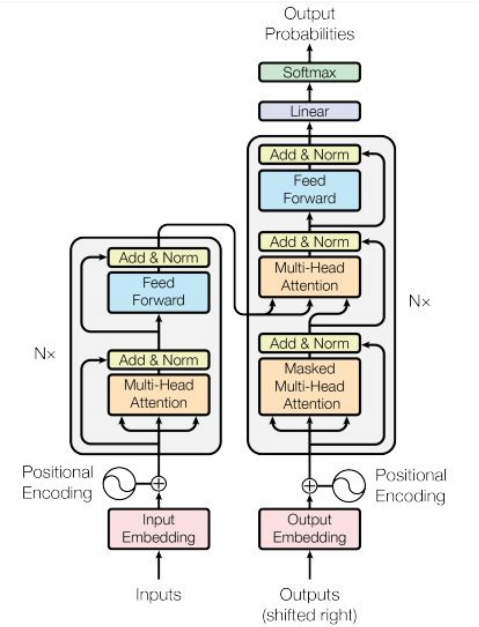


Figure 1: The Transformer - model architecture.