Deep Learning Series

Episode 2 - “La revanche des chats”

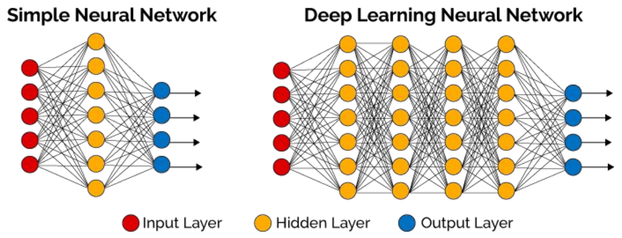
Recap from last episode

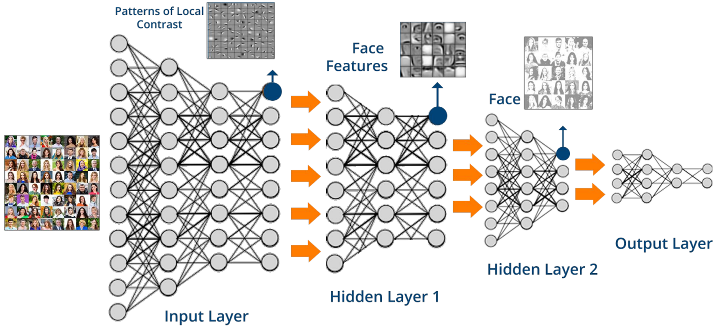
What is Deep Learning ?

Deep Learning is a **subfield of Machine Learning** : A specific way of learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations

Other approaches to **Machine Learning** tend to focus on learning only one or two layers of representations of the data -> Shallow Learning

Deep Learning = Deep Sequence of simple data transformations (layers)



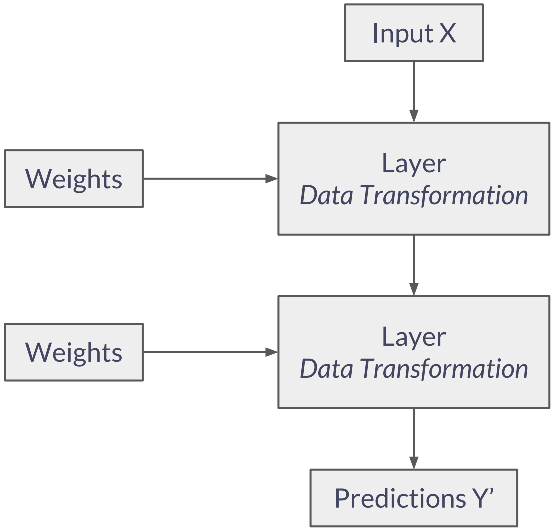


Layered representations of the data are almost always learned via models called ***Neural Networks***

How Deep Learning works ?

The Deep Learning workflow

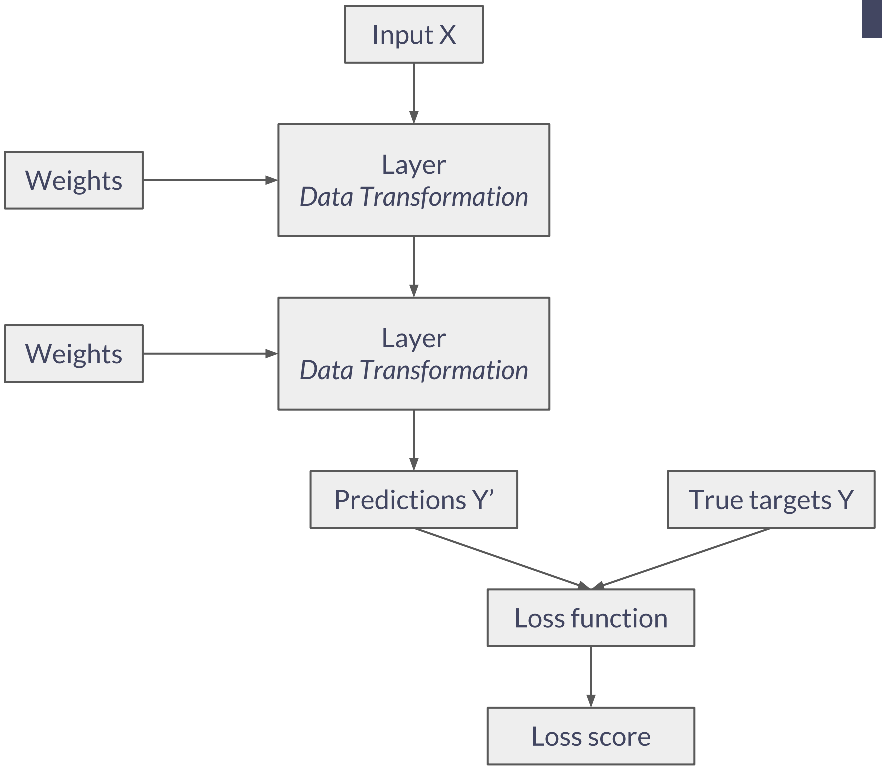
The transformation implemented by a layer is parameterized by its weights.



How Deep Learning works ?

The Deep Learning workflow

The transformation implemented by a layer is parameterized by its weights.



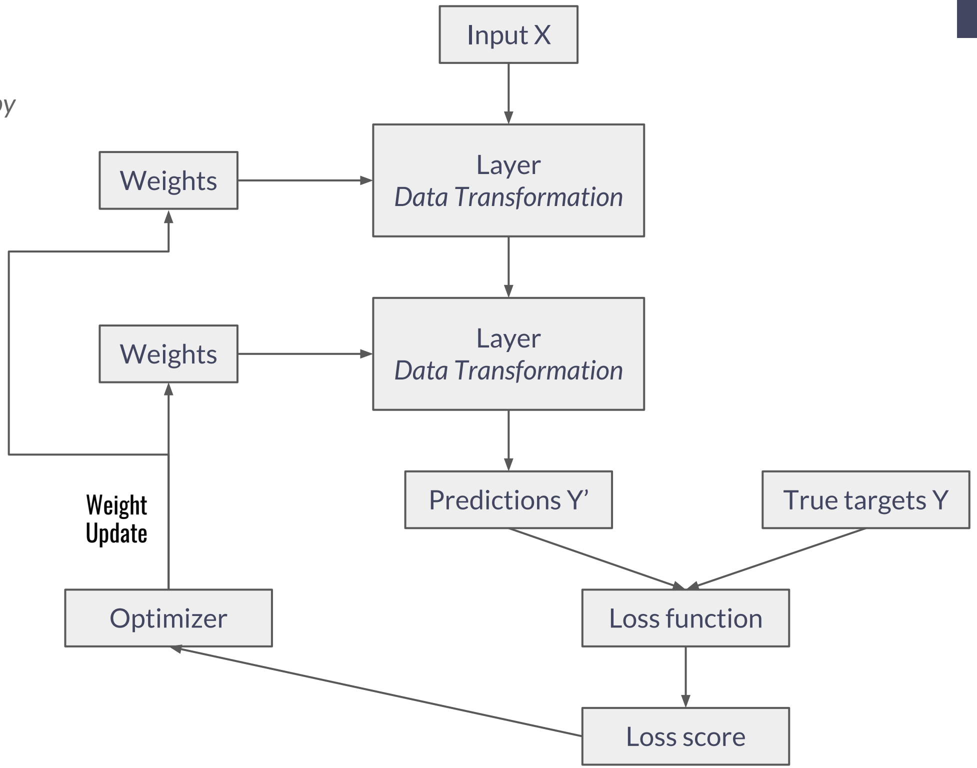
A loss function *measures the quality of the network’s output.*

It computes a distance score between the prediction and the true target

How Deep Learning works ?

The Deep Learning workflow

The transformation implemented by a layer is parameterized by its weights.



Use this score as a feedback signal *to adjust the value of the weights a little in order to lower the score for the current example.*

This job is done by an optimizer, *which implements the* Backpropagation algorithm

A loss function *measures the quality of the network's output.*

It computes a distance score between the prediction and the true target

Dive into a layer

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

In our example, we were building our network by stacking Dense layers on top of each other.
This layer can be interpreted as a function, which takes as input a 2D Tensor and returns another 2D Tensor (*a new representation for the input tensor*).

keras.layers.Dense(512, activation="relu")

⇔

max(x, 0)

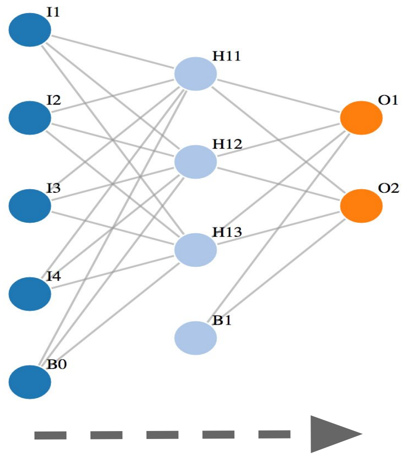
output = relu(dot(W, input) + b)

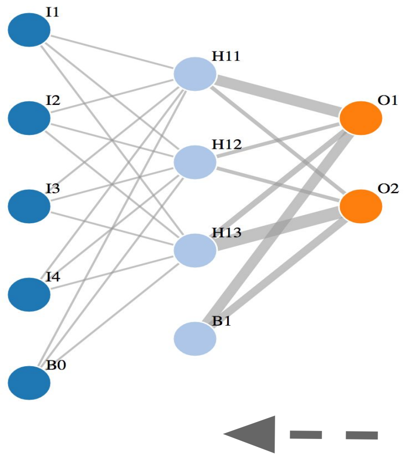
(512,)   (512, 28*28)   (28*28,)   (512,)

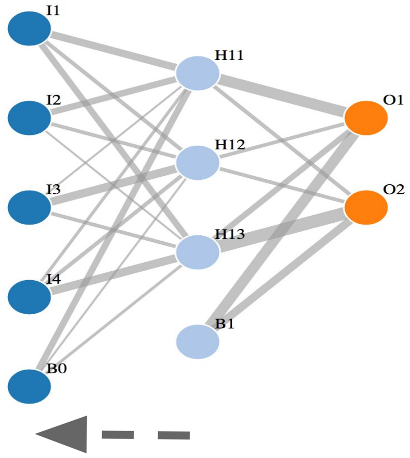
Backpropagation algorithm

In practice, a Neural Network function consists of many tensor operations chained together, each of which has a simple, known derivative.

- Applying the **chain rule** to the computation of the gradient values of the neural network gives rise of the backpropagation algorithm
- Backpropagation starts with the final loss value and works backwards from the top layers to the bottom layers.







Choosing the right last-layer activation and loss function

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

Problem type	Last-layer activation	Loss function
Binary classification	sigmoid	binary_crossentropy
Multiclass classification	softmax	categorical_crossentropy
Regression to arbitrary values	None	mse
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy

Machine Learning best practices

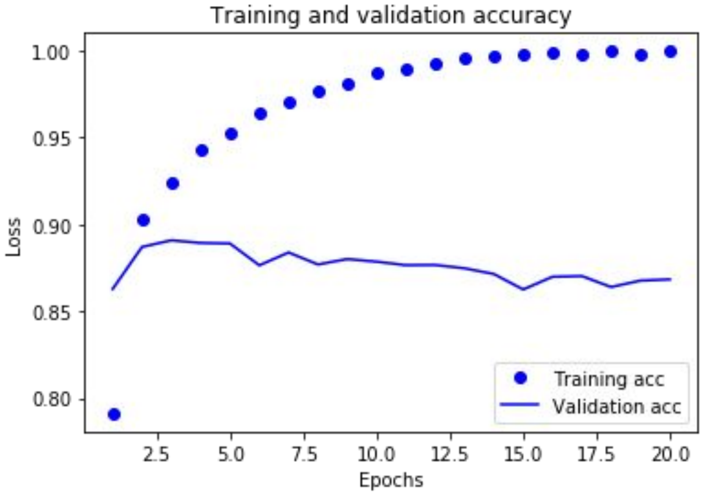
It’s not all about the model !

Evaluating Machine Learning models

How to know it works ?

The importance of validation and test sets

Recap from the exercices from last episode : All models began to overfit after a few epochs : Their performance on never-before-seen data started worsening compared to the performances on the training set.



The model begins to overfit after a few epochs

- In Machine Learning, the goal is to achieve models that generalize.
- Splitting the dataset into a training, validation and test set is a crucial step to measure the generalization power of the model.

Training, validation and test sets

Evaluation a model always boils down to splitting the available data into 3 sets :

- **Training set** : The data to train your model on
- **Validation set** : The data to evaluate your model on
- **Test set** : The data to evaluate your model on during the final step before production

Why not have 2 sets instead of 3 ?

- Developing a model always involves tuning its hyperparameters (#of layers, size of the layers, # of epochs, etc.)
- Choose the best hyperparameters based on the metric on your validation set
- Do a final evaluation of your model on the test set to see if you did not start to overfit the validation set (*information leaks*)

Train / validation / test split

Other forms of evaluation

Main problem : validation and test sets are not representative enough if **too little data is available**.

Solution : K-Fold validation

- Split your data into K partitions of equal size
- For each partition, train on K-1 folds, and evaluate on the remaining one
- Your final score is the average of the K scores obtained

3-Fold validation

Data preprocessing, Feature Engineering and Feature learning

How to prepare the data before feeding the model ?

Data preprocessing the Neural Networks

Vectorization

All inputs and targets in a neural network must be **tensors of floating-point data**.

Ex : Text data -> Representation as list of integers and one-hot-encoding

Data preprocessing the Neural Networks

Vectorization

All inputs and targets in a neural network must be **tensors of floating-point data**.

Ex : Text data -> Representation as list of integers and one-hot-encoding

Value Normalization

It isn't safe to feed into a neural network data that takes relatively large values, or data that is heterogeneous. It can trigger large gradient updates that will harm convergence. **Your data must take small values and be homogeneous.**

Ex : Pixel data encoded in the 0-255 range -> Normalize to floating-points in the 0-1 range

Best practice : Normalize each feature independently to have a mean of 0 and a standard deviation of 1.

Data preprocessing the Neural Networks

Vectorization

All inputs and targets in a neural network must be **tensors of floating-point data**.
Ex : Text data -> Representation as list of integers and one-hot-encoding

Value Normalization

It isn't safe to feed into a neural network data that takes relatively large values, or data that is heterogeneous. It can trigger large gradient updates that will harm convergence. **Your data must take small values and be homogeneous.**
Ex : Pixel data encoded in the 0-255 range -> Normalize to floating-points in the 0-1 range

Best practice : Normalize each feature independently to have a mean of 0 and a standard deviation of 1.

Handling missing values

Neural Networks can't handle missing values. They need to be **filled with a statistically coherent value,** or **drop** the whole data point.

Data preprocessing the Neural Networks

Feature Engineering & Feature Learning

Using your own domain knowledge about the data to help the model do better predictions by applying hardcoded transformations to the data before it goes into the model. It's about making the problem easier by **expressing it in a simpler way**.

Data preprocessing the Neural Networks

Feature Engineering & Feature Learning

Using your own domain knowledge about the data to help the model do better predictions by applying hardcoded transformations to the data before it goes into the model. It's about making the problem easier by **expressing it in a simpler way**.

- **Before Deep Learning, Feature Engineering used to be critical** : classical shallow algorithms didn't have hypothesis spaces rich enough to learn useful features by themselves
- **Modern Deep Learning removes the need for most Feature Engineering**, because Neural Networks are capable of automatically extracting useful features from raw data.

But this doesn't mean you don't have to worry about Feature Engineering with Deep Learning algorithms !

- Good features still allow you to **solve problems using fewer resources** (model complexity)
- Good features let you **solve a problem with far less data**. The ability of Neural Networks to learn features on their own relies on having lots of data available.

Overfitting and underfitting

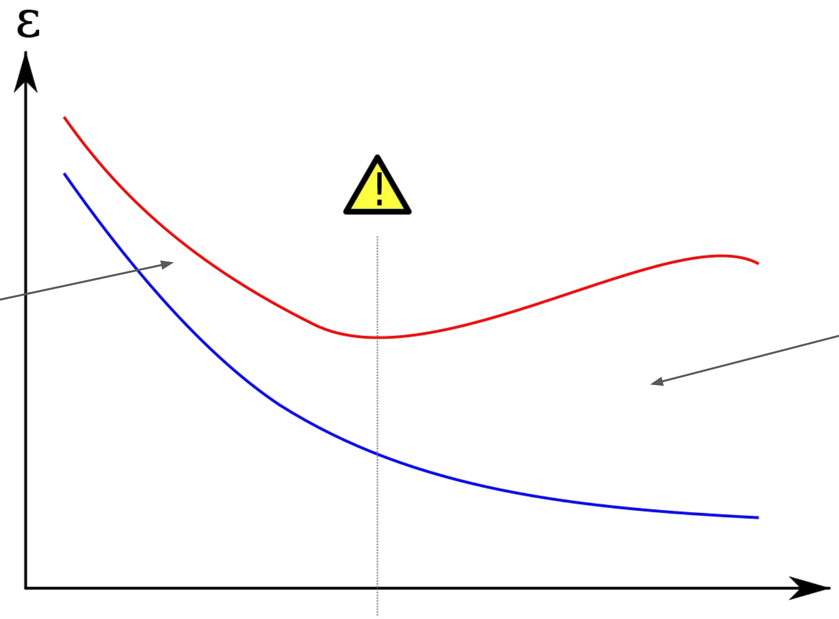
How to find the right balance ?

Uderfitting vs Overfitting

Underfitting

*At the beginning, **the lower the loss in training data, the lower the loss on validation data**.*

There is still progress to be made : The model hasn't yet modeled all relevant patterns in the training data.



Overfitting

After some iterations, generalization stops improving.

*The model is starting to **learn patterns that are specific to the training data and irrelevant to new data**.*

image source : wikipedia

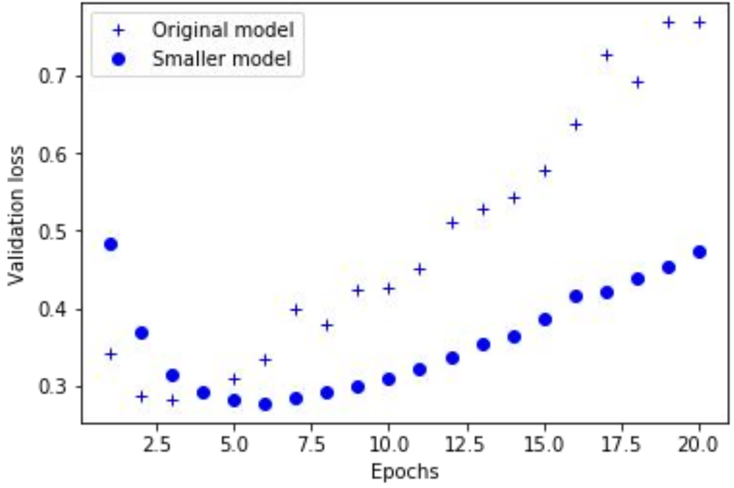
Fighting overfitting : Reducing the network's size

Simplest way to reduce overfitting : Reducing the number of learnable parameters (=*degrees of freedom*) in the model

- A model with more parameters has more memorization capacity
- With fewer learnable parameters, the model will have to learn **compressed representations** to minimize its loss
- Find the right amount of parameters by tuning the model's configuration

```
original_model = models.Sequential()
original_model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
original_model.add(layers.Dense(16, activation='relu'))
original_model.add(layers.Dense(1, activation='sigmoid'))
```

```
smaller_model = models.Sequential()
smaller_model.add(layers.Dense(4, activation='relu', input_shape=(10000,)))
smaller_model.add(layers.Dense(4, activation='relu'))
smaller_model.add(layers.Dense(1, activation='sigmoid'))
```



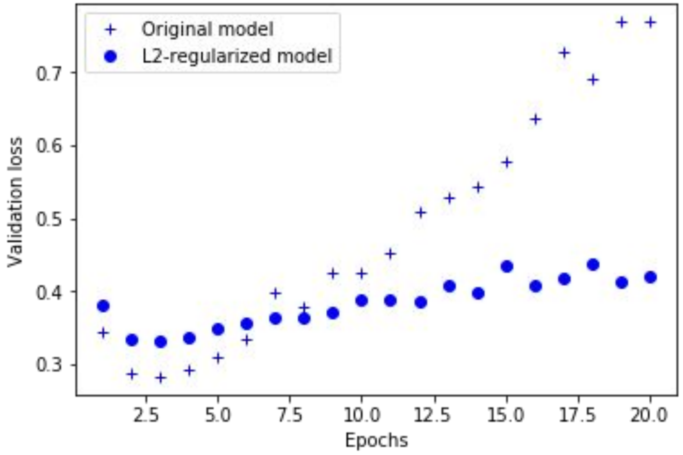
Fighting overfitting : Adding weight regularization

Principle of *Occam's razor* : Given two explanations for something, the explanation most likely to be correct is the simplest one.

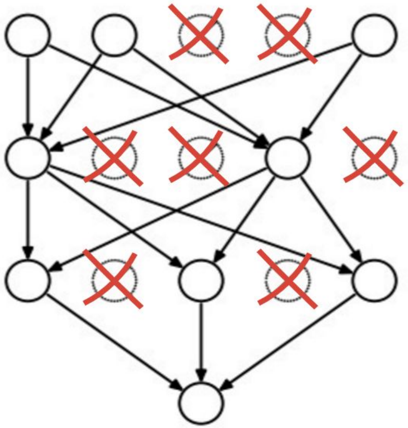
- ➢ Simpler models are less likely to overfit that complex ones
- ➢ Simpler model => Forcing its weights to take only small values (more regular distribution of weights)
- ➢ This is done by adding to the lost function a cost associated with having large weights (L1 or L2 regularization)

```
original_model = models.Sequential()
original_model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
original_model.add(layers.Dense(16, activation='relu'))
original_model.add(layers.Dense(1, activation='sigmoid'))
```

```
l2_model = models.Sequential()
l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                          activation='relu', input_shape=(10000,)))
l2_model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                          activation='relu'))
l2_model.add(layers.Dense(1, activation='sigmoid'))
```



Fighting overfitting : Adding dropout

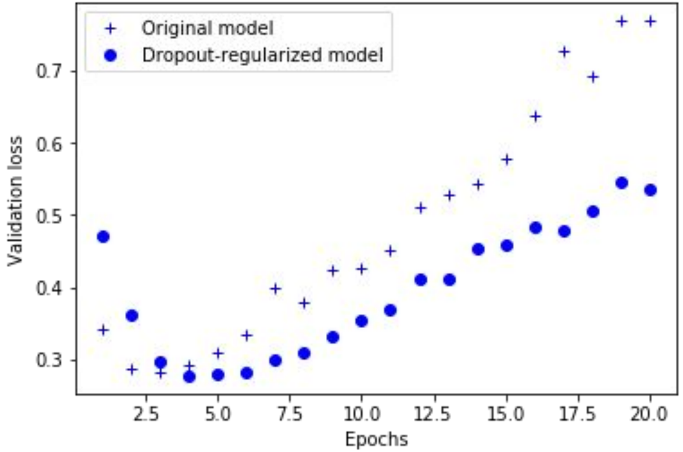


Dropout : Randomly dropping out (setting to 0) a number of output features of a given layer during training. The dropout rate is usually between 0.2 and 0.5. Use all outputs at test time.

➢ Core idea : **Introducing noise** in the output values can **break up "*bad luck circumstance patterns*"** in the training data. Information need to be encoded at several places to be relevant.

```
original_model = models.Sequential()
original_model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
original_model.add(layers.Dense(16, activation='relu'))
original_model.add(layers.Dense(1, activation='sigmoid'))
```

```
dpt_model = models.Sequential()
dpt_model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
dpt_model.add(layers.Dropout(0.5))
dpt_model.add(layers.Dense(16, activation='relu'))
dpt_model.add(layers.Dropout(0.5))
dpt_model.add(layers.Dense(1, activation='sigmoid'))
```



Universal workflow of Machine Learning

What steps need to be systematically done ?

Universal workflow

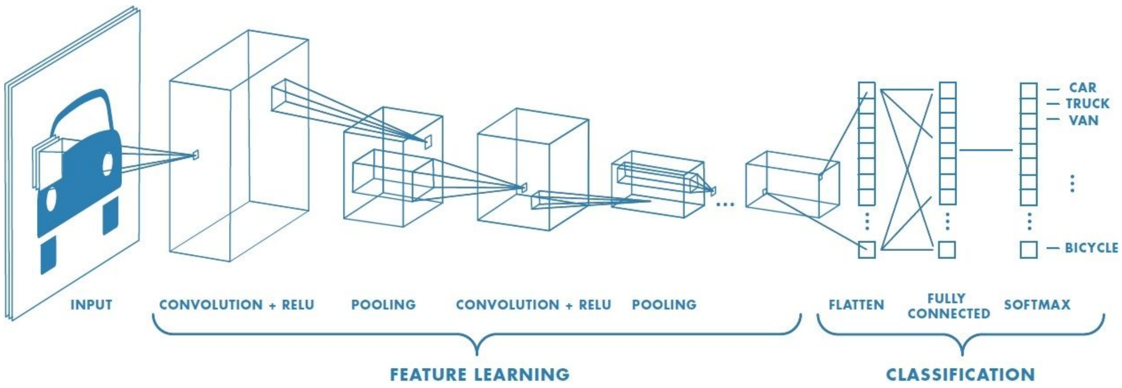
1. Define the problem & assemble a dataset
2. Choose a measure of success
3. Decide an evaluation protocol
4. Prepare your data
5. Develop a model better than a baseline
6. Develop a model that overfits
7. Regularize & tune hyperparameters

Convolutional Neural Networks

The best choice for image data

What we will see ?

- Understanding Convolutional Neural Networks
- Using Data Augmentation
- Using pretrained networks
- Visualize what convnets learn



Source : https://www.mathworks.com/discovery/convolutional-neural-network.html

First dive into the code

Then, we'll understand :)

Defining the convolutional base

Densely-connected network

```
network = models.Sequential()
network.add(layers.Dense(512, activation='relu',
                         input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

Defining the convolutional base

- A basic convnet is a stack of Conv2D and MaxPooling2D layers
- Takes as input tensors of shape *(image_height, image_width, image_channels)*
  - Reminder for Densely connected networks : input tensors of shape (num_pixels,)

Densely-connected network

```
network = models.Sequential()
network.add(layers.Dense(512, activation='relu',
                         input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

Defining the convolutional base

- A basic convnet is a stack of Conv2D and MaxPooling2D layers
- Takes as input tensors of shape *(image_height, image_width, image_channels)*
  - Reminder for Densely connected networks : input tensors of shape (num_pixels,)

Densely-connected network

```
network = models.Sequential()
network.add(layers.Dense(512, activation='relu',
                         input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

Convolutional Neural Network

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

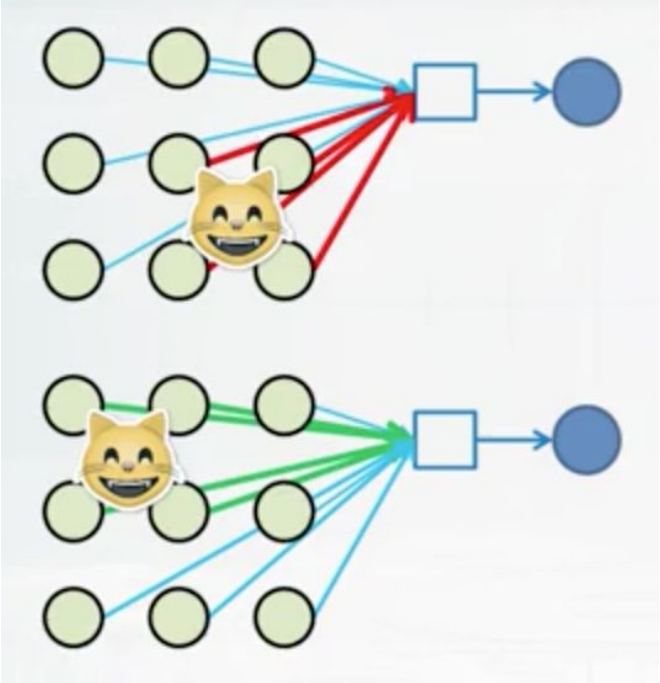
+

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

Understanding convnets

I can't wait any longer !

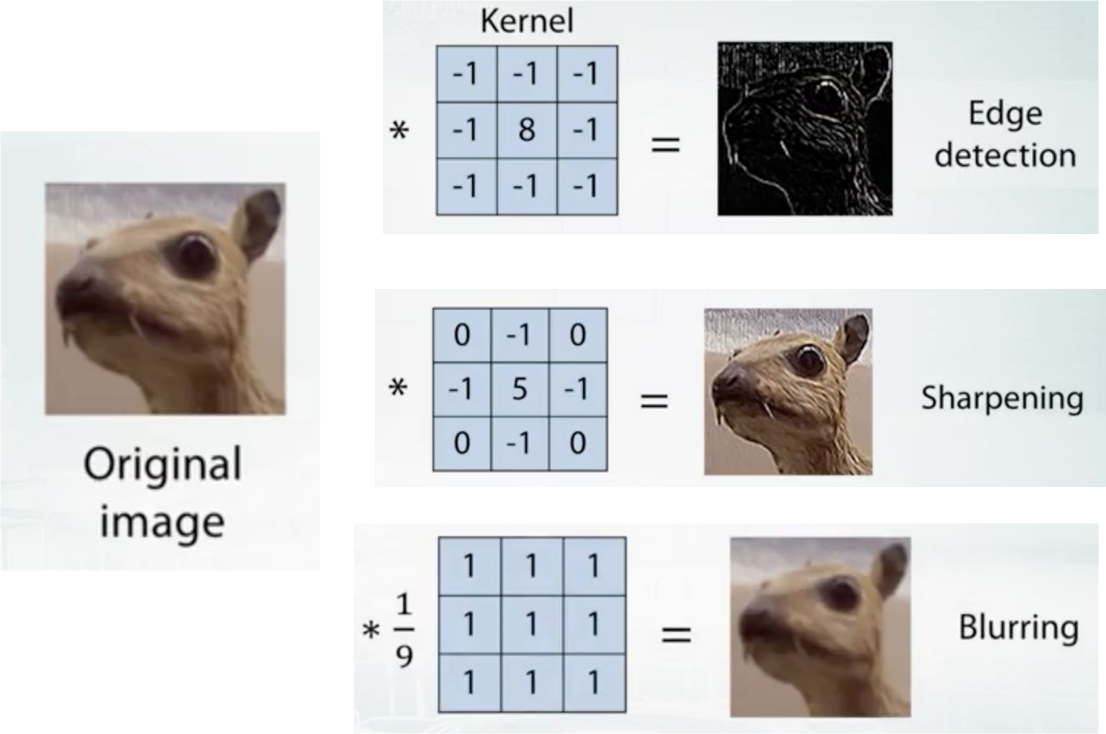
Why do we need convnets ?



- A densely-connected neural network has to learn the same "cat features" in different areas
  - What if cats appear in a different place in the test set ?

Source : https://www.coursera.org/learn/intro-to-deep-learning

Defining the convolutional operation



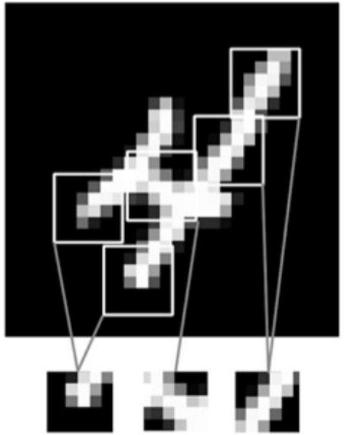
Global vs Local patterns

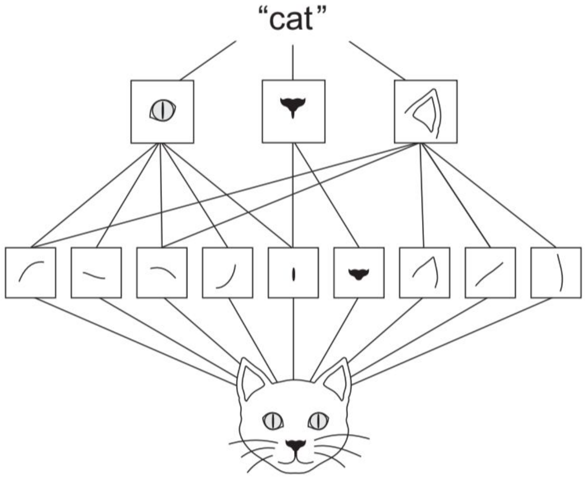
- A Dense layer learns global patterns (involving all pixels)
- A Convolutional layer learns local patterns

Source : https://www.coursera.org/learn/intro-to-deep-learning

Key characteristics of convolutional neural networks

- The patterns they learn are translation invariant: it can recognize a pattern anywhere
- They can learn spatial hierarchies of patterns
    - A first convolutional layer will learn small local patterns such as edges
    - A second convolutional layer will learn larger patterns made of the features of the first layers, and so on.
    - Allows convnets to learn increasingly complex and abstract visual concepts





Key characteristics of convolutional neural networks

```
>>> model.summary()

Layer (type)                   Output Shape          Param #
=================================================================
conv2d_1 (Conv2D)              (None, 26, 26, 32)    320
_________________________________________________________________
maxpooling2d_1 (MaxPooling2D)  (None, 13, 13, 32)    0
_________________________________________________________________
conv2d_2 (Conv2D)              (None, 11, 11, 64)    18496
_________________________________________________________________
maxpooling2d_2 (MaxPooling2D)  (None, 5, 5, 64)      0
_________________________________________________________________
conv2d_3 (Conv2D)              (None, 3, 3, 64)      36928
_________________________________________________________________
flatten_1 (Flatten)            (None, 576)           0
_________________________________________________________________
dense_1 (Dense)                (None, 64)            36928
_________________________________________________________________
dense_2 (Dense)                (None, 10)            650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

First conv layer computes 32 filters over its input, resulting on 32 response maps of size (26, 26)

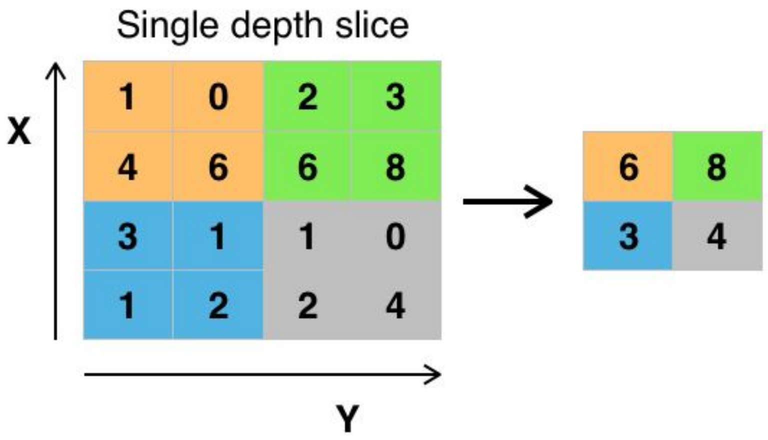
Second conv layer computes 64 filters over its input, resulting on 64 response maps of size (11, 11)

Width and height dimensions tend to shrink as we go deeper, while the number of channels tends to increase.

Defining the Max Pooling operation

Role of max pooling

- ➢ Aggressively downsample feature maps
- ➢ Consists of extracting windows from the input feature maps and outputting the max value of each channel



Source : https://commons.wikimedia.org/wiki/File:Max_pooling.png

Why Max Pooling is important ?

Convolutional base without max-pooling operations

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 22, 22, 64)        36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
```

What's wrong if there is no max-pooling layer ?

- ➢ You don't learn spatial hierarchy of features. The high level patterns learned will still be very small with regard to the initial input
- ➢ The final feature map will have **too many parameters** to feed to the Dense layer
    - Dense layer of size 512 -> 36928 * 512 = 15.8 million parameters !

Cats & Dogs Recognition

Training a convnet from scratch on a small dataset

Data Augmentation

- Data Augmentation takes the approach of generating more training data from existing training samples via a number of random transformations that yield believable-looking images
- Goal : At training time, the model will never see the same image twice (but they are still very correlated).
- This helps to generalize better.

Source : Keras Blog (https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html)

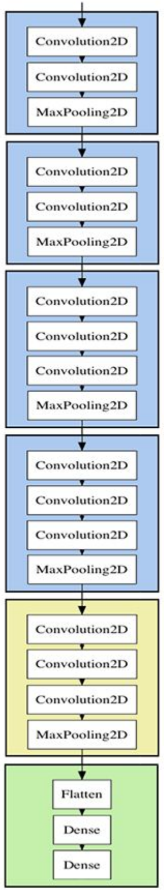
Cats & Dogs Recognition

Using a pretrained convnet for Feature Extraction

Pretrained Network - What is it ?

- Pretrained network: Saved network that was **previously trained on a large dataset**
- The learned features are highly repurposable
- Highly effective approach to Deep Learning on small image datasets

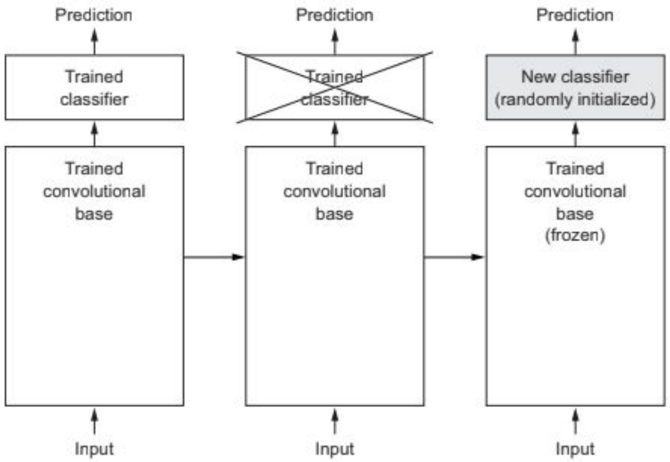
- A lot of pretrained networks exist for image classification
  - They are very often trained on ImageNet, a database of millions of images for 1000 distinct classes
  - Common architectures : VGG16, ResNet, Inception, …
- Some of them come prepackaged with Keras in the keras.applications module.



VGG16 architecture

Pretrained Network for Feature Extraction

- Feature Extraction : Using the representations learned by a previous network to extract interesting features from new samples
- These features are then run through a new classifier, which is trained from scratch.
- We take the convolutional base of the pretrained network and train a new classifier on top of the output.
- The level of generality of the representations depends on the depth of the layer in the model.



Only reuse the convolutional base !

- Representations learned by the convolutional base are likely to be **more generic** and therefore **more reusable**
- Representations found in densely-connected layers do not contain information about **where objects are located**

Pretrained Network for Feature Extraction

```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
```

Three arguments :

- **weights** : specifies the weight checkpoint from which to initialize the model
- **include_top** : include (or not) the densely connected classifier on top of the network
- **input_shape**: shape of the image tensor fed to the network.
    - If not used, the network will be able to process inputs of any size

Pretrained Network for Feature Extraction

Once we have the convolutional base of the pretrained network, you need to stick a densely connected classifier on top of it. Two ways to proceed :

- Run the convolutional base over the dataset, **record its output on disk** and use this data as input to a standalone densely connected classifier
  - Fast and cheap to run (run the convolutional base only once for each image)
  - Doesn't allow to use data augmentation
- Extend the convolutional base by **adding Dense layers on top**, and run the whole thing end to end on the input data
  - Allows to use data augmentation
  - Far more expensive (only use it on GPU)

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Pretrained Network for Feature Extraction

After extending the convolutional base with dense layers, you must freeze it

- Prevents the weights of the convolutional base from getting updated during training
- Since dense layers are initialized randomly, very large gradient updates will be propagated, destroying the representations learned in the convolutional base

In Keras, freezing a network is done by setting its trainable parameter to False

```
conv_base.trainable = False
```

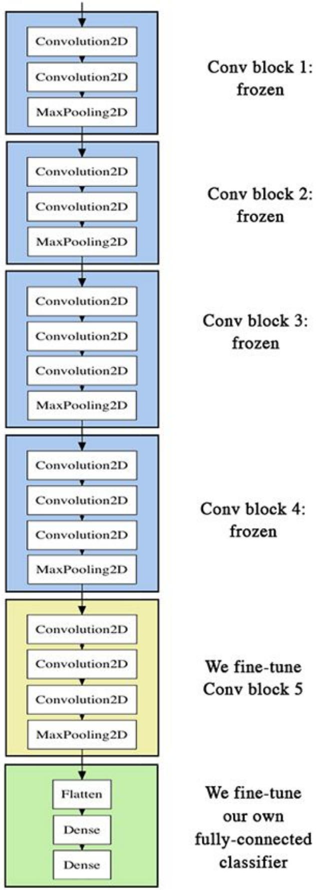
Cats & Dogs Recognition

Using a pretrained convnet with fine-tuning

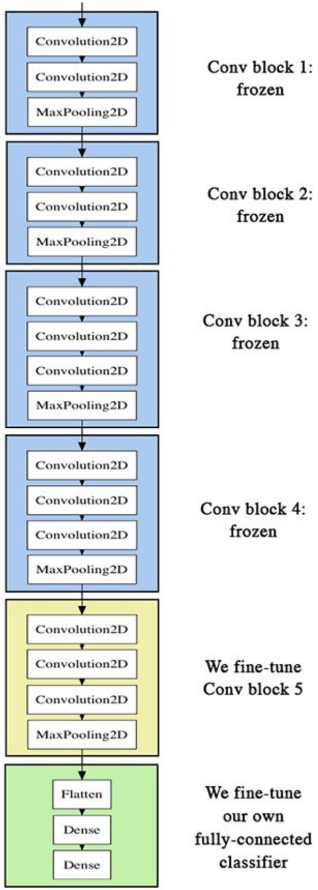
Pretrained Network with fine-tuning

- **Fine-tuning** : Unfreezing a few of the top layers of a frozen model base used for feature extraction, and train them jointly with the densely connected layers.
  - Complementary to feature extraction
  - Slightly adjusts the more abstract representations of the model being reused in order to make them more relevant for the problem at hand

- Way to proceed :
  - Add your custom network on top of a pretrained base network
  - Freeze the base network
  - Train the part you added
  - Unfreeze some layers in the base network
  - Jointly train both these layers and the part you added



Pretrained Network with fine-tuning



Why not fine-tune the entire convolutional base ?

- Earlier layers in the convolutional base encore more generic, reusable features, whereas layers higher up encode more specialized features
  - It is more useful to fine-tune the more specialized features, because they need to be repurposed for our problem
  - There would be fast-decreasing returns in fine-tuning lower layers
- The more parameters you're training, the more you're at risk of overfitting.
  - The convolutional base has 15 million parameters !
  - It is risky to attempt to train it on your small dataset

Take aways

What did we learn ?

Pretrained Network for Feature Extraction

- Convolutional Neural Networks are the best type of machine learning models for computer vision tasks
  - It is possible to train them from scratch even on very small datasets
- On a small dataset, overfitting will be the main issue. Data augmentation is a powerful way to fight it for image data
- It's easy to reuse an existing convnet that was trained on a larger dataset for Feature Extraction
- Fine-tuning adapts to a new problem some of the representations previously learned by an existing model
  - This pushes performances a bit further