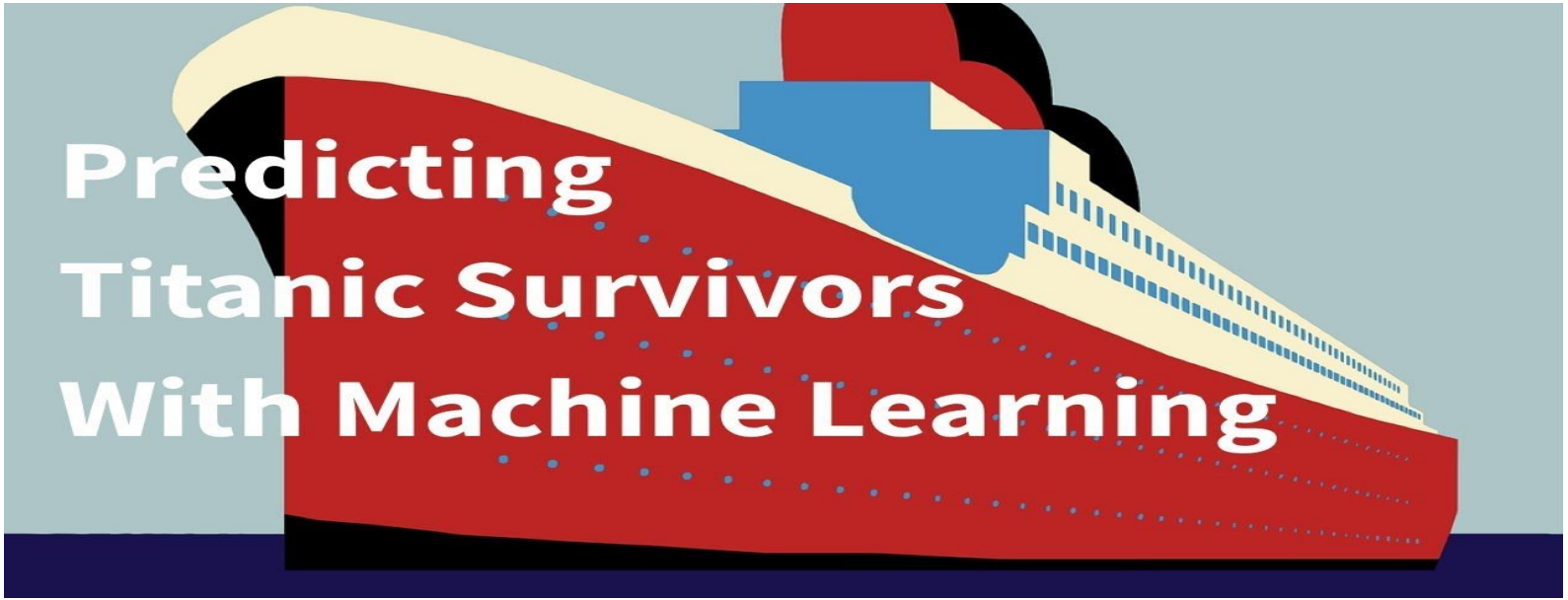
Soutenance du Projet Filé Kaggle

Dogs vs Cats : Classification binaire de chiens et de chats

Naoto Lucas, Jean-Baptiste Gaeng, Ghada Meftah, Julien Danh

I - Travail préliminaire : Projet Kaggle Titanic

1 - Présentation du sujet :



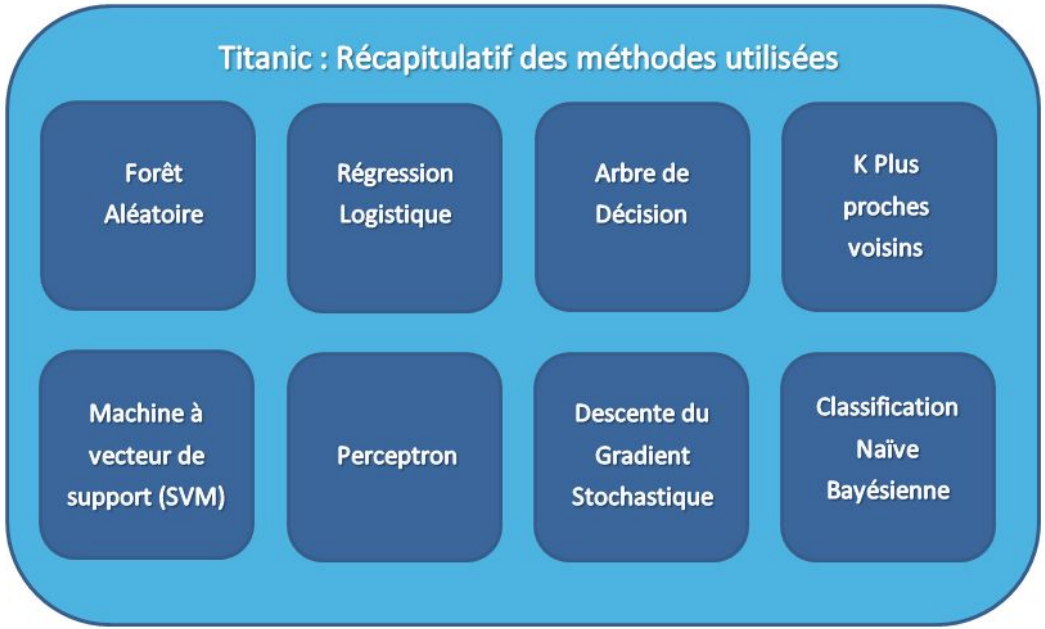
1 - Présentation du sujet :

- **Données :**
- 2201 observations
- 3 variables prédictives :
  - ❏ **Classe** : 0 à 3 ,
  - ❏ **Age** : 0 (enfant) ou 1 (adulte)/
  - ❏ **Sexe :** 0 (femme) ou 1 (homme)

➔ **Objectif :**

- prédire la survie (ou le décès) d'un passager du Titanic (Y = 1 si survie, 0 sinon)

2 - Méthodes utilisées :



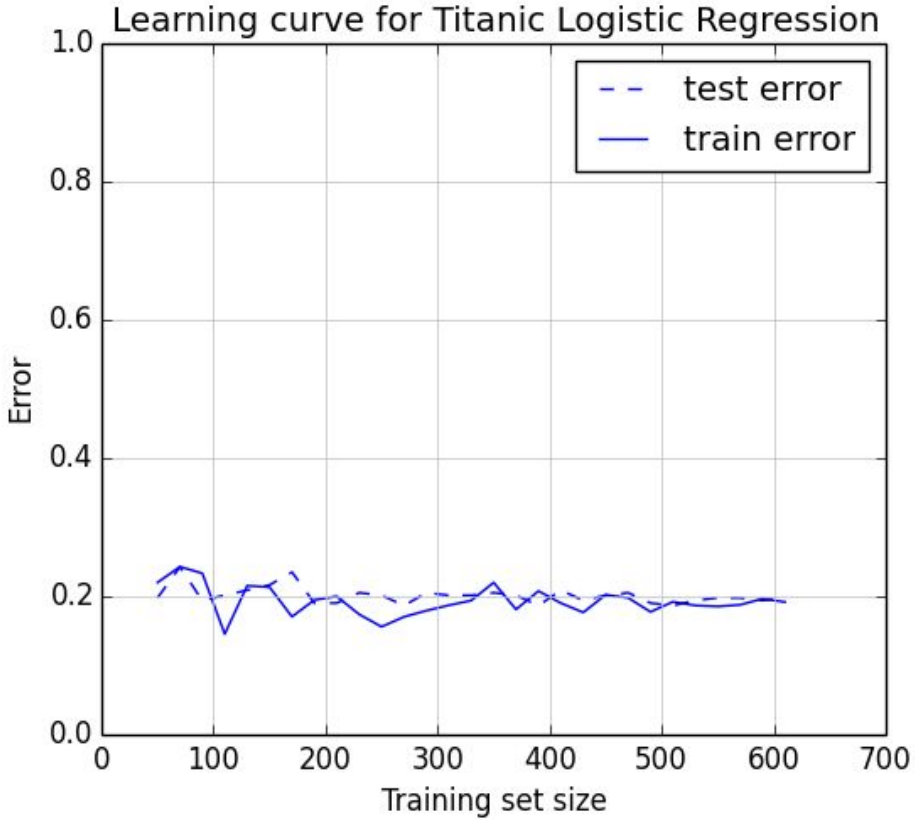
> On ne détaille dans la suite que **Forêt Aléatoire** et **Régression Logistique** (autres algos : cf. rapport)

2.1 - Régression logistique :

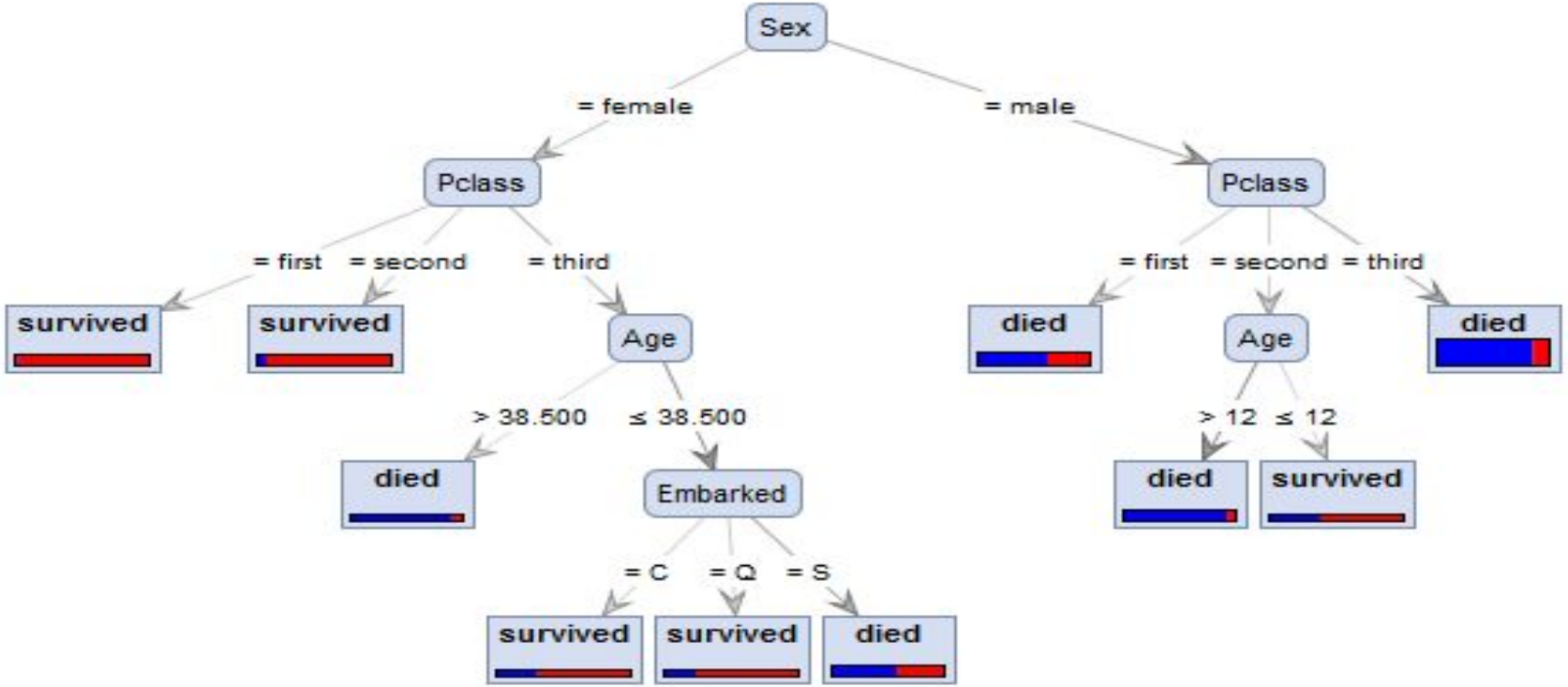
- Prélever aléatoirement ⅓ du jeu d'apprentissage comme données de "test" .

- Prépare le modèle avec des sous-ensembles aléatoires et de taille croissante des données restantes (de 50 à 916)

- Pour chacun de ces modèles on calcule l'*Erreur* (*Erreur = 1.0 - Accuracy*) sur le jeu d'apprentissage et les données de "test" prélevées initialement.

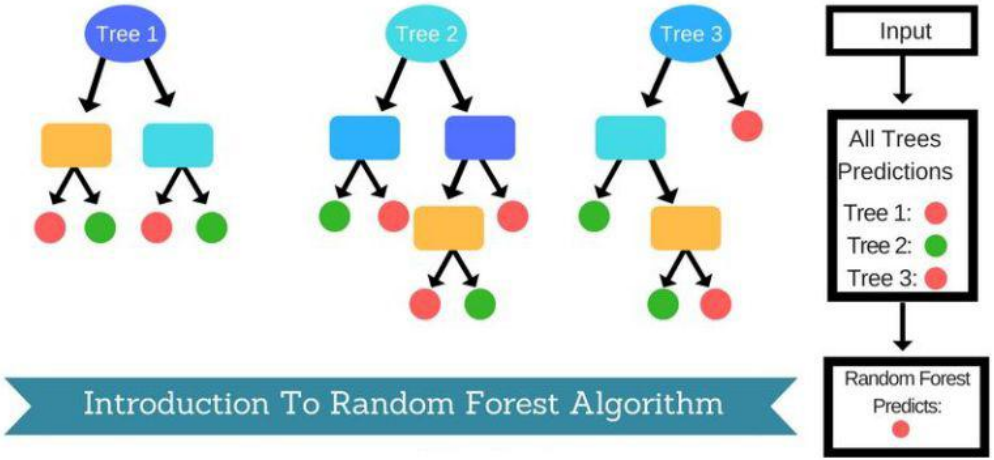


2.2 - Arbre de décision:



2.3 - Random Forest

La méthode de classification par forêt aléatoire consiste à construire une forêt d'arbres décisionnels, et de prédire le résultat final en se basant sur le résultat prédit par chaque arbre de la forêt.



3 - Résultats obtenus

	Model	Score
3	Random Forest	86.76
8	Decision Tree	86.76
1	KNN	84.74
0	Support Vector Machines	83.84
2	Logistic Regression	80.36
7	Linear SVC	79.12
5	Perceptron	78.00
6	Stochastic Gradient Decent	77.67
4	Naive Bayes	72.28

II - Projet Kaggle Dogs vs Cats

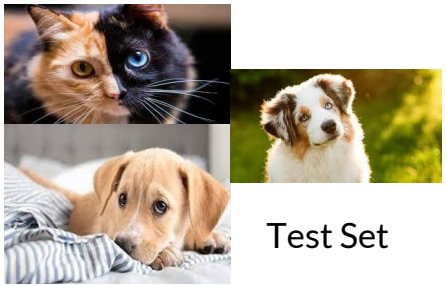
1 - Présentation du sujet

> Problème de **classification binaire avec des images de chiens et de chats**. Sujet de la compétition "Dogs vs. Cats Redux: Kernels Edition" : https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition

> Données fournies :

- Un dossier train : 25,000 images de chiens et chats. Chaque nom de fichier de ce dossier contient un label (cat ou dog)
- Un dossier test : 12,500 images. Pour chaque photo, nous devons prédire la probabilité qu'elle corresponde à un chien (1 = chien, 0 = chat).

Train Set



Test Set

Modèle Prédictif

1 = chien

0 = chat

2 - Objectifs

> Minimiser la fonction LogLoss

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

where

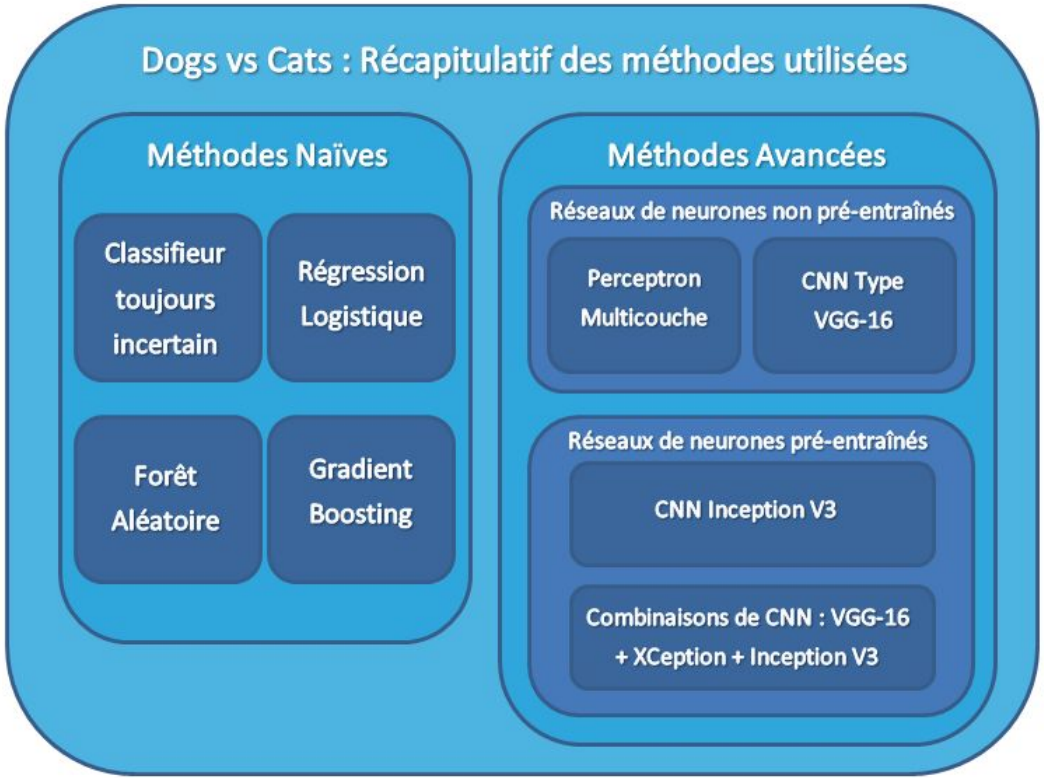
- n is the number of images in the test set
- $\hat{y}_i$ is the predicted probability of the image being a dog
- y_i is 1 if the image is a dog, 0 if cat
- $log()$ is the natural (base e) logarithm

A smaller log loss is better.

> Pour ce faire, utilisation d'**algos simples** comme pour Titanic **puis plus complexes**

> **Les Kernels Kaggle sont une bonne base pour les méthodes complexes**

3 - Méthodes Utilisées



3.1 - Méthodes naïves

But : avoir des références de LogLoss

3.1.1 - Classifieur toujours incertain

> On ne classe pas les images, i.e on **propose en guise de prédictions pour chaque image du jeu d'images tests des probabilités égales à 0,5**.

Proba(Chien) = 0.5

Proba(Chien) = 0.5

Proba(Chien) = 0.5

Score de Logloss obtenu : 0.69

> Remarque : référence de Logloss non probabiliste : soumission d'un **fichier de prédiction prédisant à coup sûr pour chaque image un chien**. Ceci correspond à des probabilités toujours égales à 1.

Score de Logloss obtenu : 17. 27

3.1 - Méthodes naïves (2)

3.1.2 - Gradient Boosting

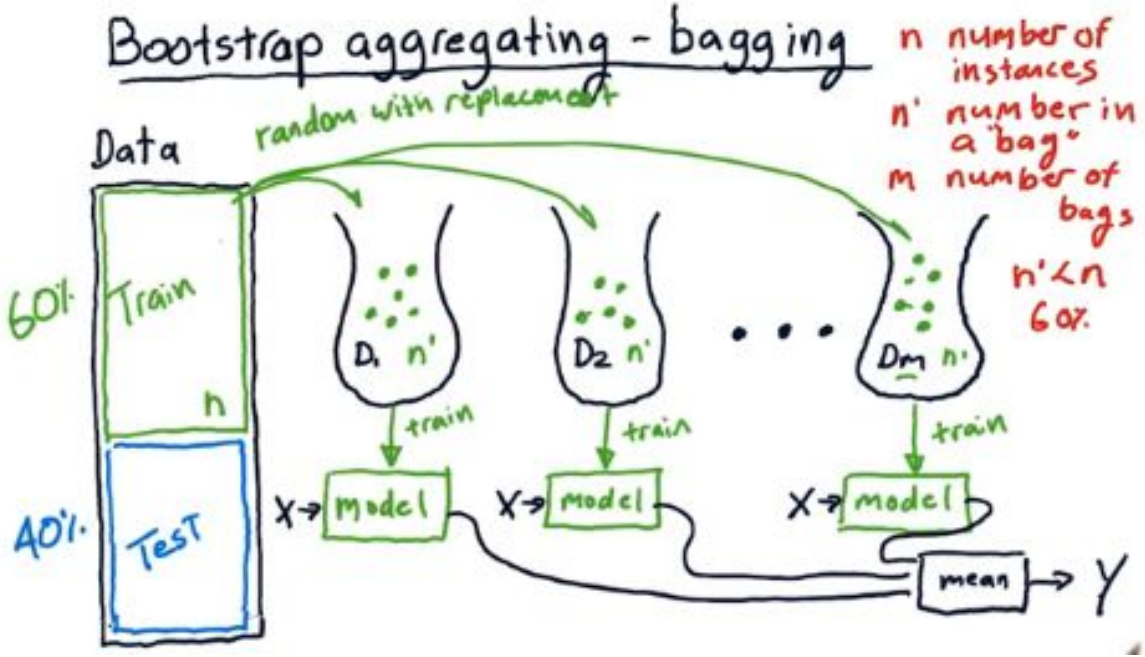
> Principe de la méthode :

> Gradient Boosting : méthode particulière pour tester et entraîner les données. On peut choisir ensuite n'importe quel type d'algorithme pour prédire les données, dont les arbres de décisions (Gradient Tree Boosting)

> Gradient Tree Boosting ≈ Random Forest. Seule différence : la façon d'entraîner et de tester les données : "bagging" pour Random Forest et "boosting" pour Gradient Tree Boosting

Différences entre “bagging” et “boosting”

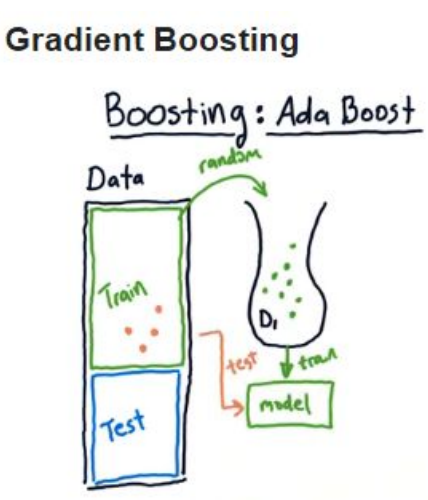
> Bagging :



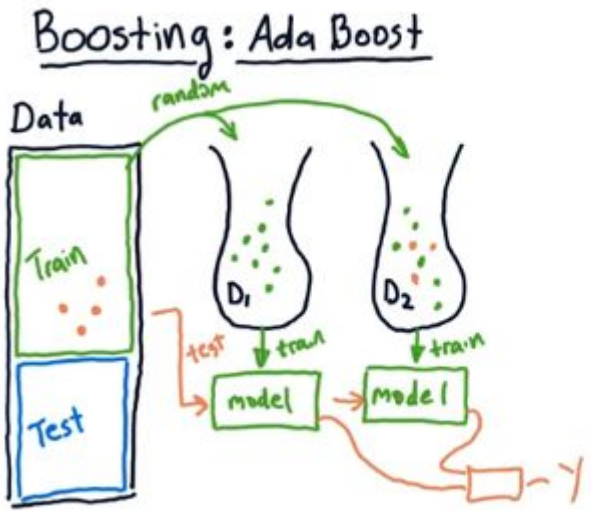
Différences entre “bagging” et “boosting” (2)

> Boosting :

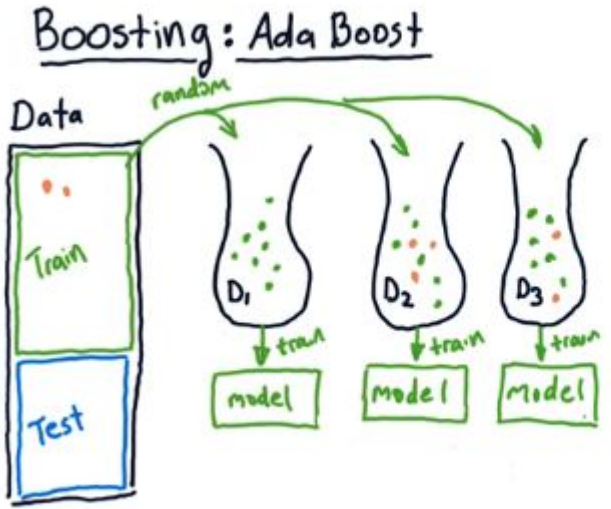
Step 1



Step 2



Step 3



3.2 - Résultats des méthodes naïves

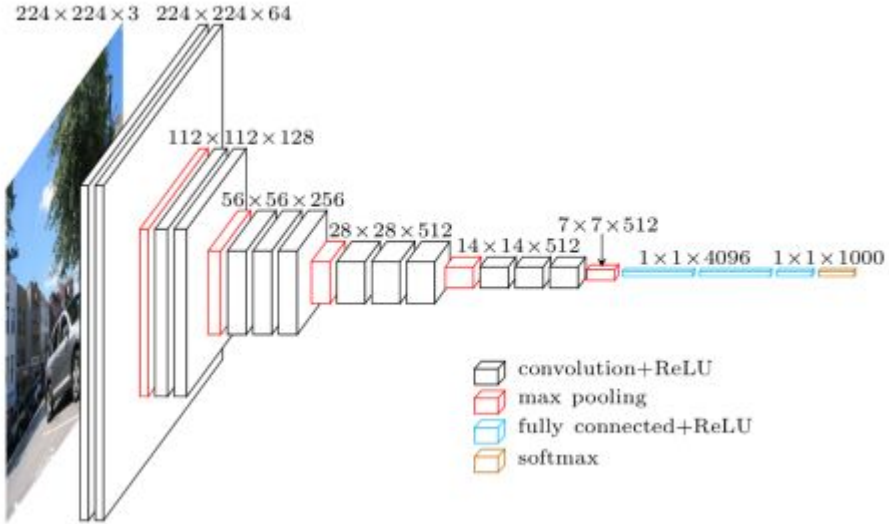
Méthode	**Accuracy sur les données d'apprentissage**	**Logloss**
Classification toujours incertaine	0.50	0.69
Régression Logistique probabiliste	0.68	0.66
Forêt Aléatoire	0.98	17.57
Gradient Boosting	0.70	17.61

3.3 Méthodes avancées

3.3.1 Réseau de neurones convolutifs de type VGG-16

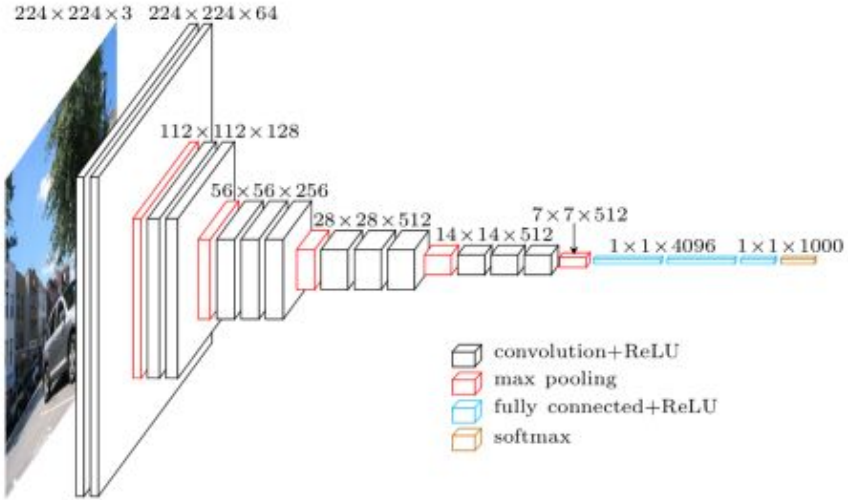
> Kernel Kaggle : https://www.kaggle.com/jeffd23/catdognet-keras-convnet-starter

> Définition VGG -16 : réseau de neurones convolutifs de 16 couches, créé par Visual Geometry Group : termine premier à compétion Kaggle ImageNet (2014)



Architecture du VGG-16 pour une image de taille 224x224

Les différentes couches d'un CNN



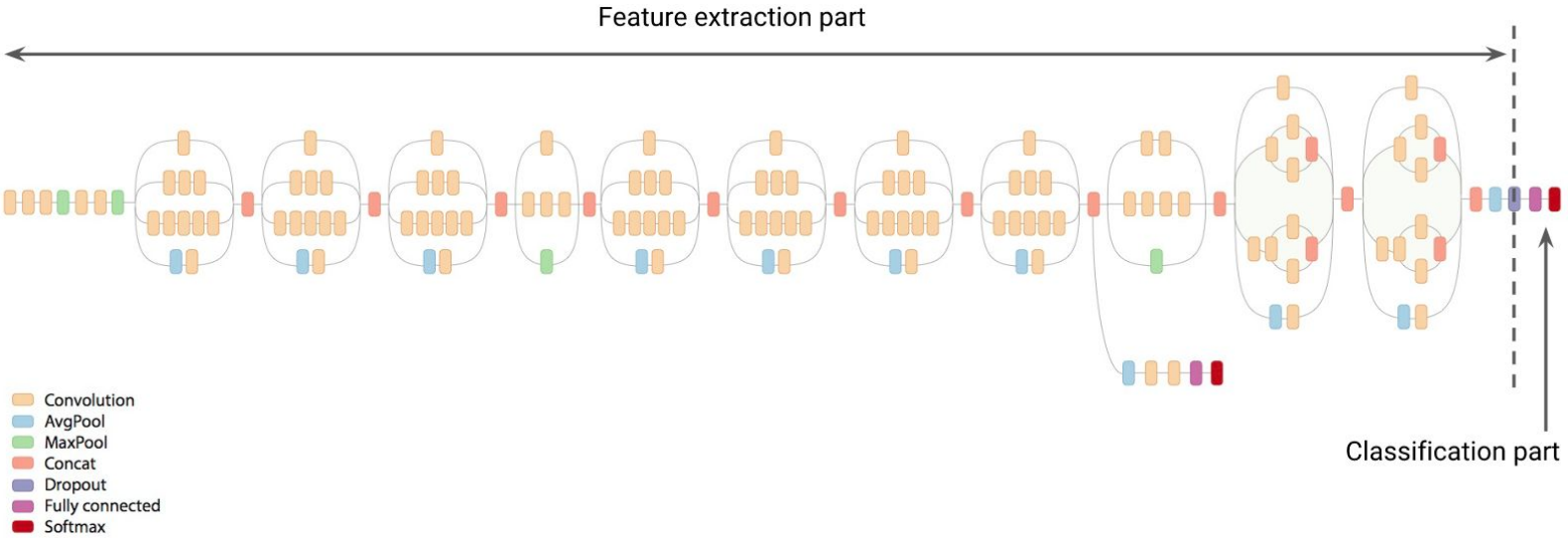
- **Pooling** : permettant de prendre une large image et d'en réduire la taille tout en préservant les informations les plus importantes qu'elle contient
- **Unité linéaire rectifiée (ReLU)** : remplacer par un 0 chaque fois qu'il y a une valeur négative dans un pixel. Cette opération permet au CNN de rester en bonne santé
- **Fully Connected Layer** : Raisonnement de haut-niveau. Connexions avec toutes les activations de la couche précédente.
- **Softmax** : fonction d'activation logistique. Sers à prendre la proba maximale des classes sur dernière couche

3.3.1 Résultats du CNN de type VGG-16

Méthode	Accuracy sur l'ensemble de validation	Logloss
CNN de type VGG-16	0.71	0.50

3.3 Méthodes avancées - Transfer Learning

3.3.3 Réseau de neurones pré-entraîné Inception V3

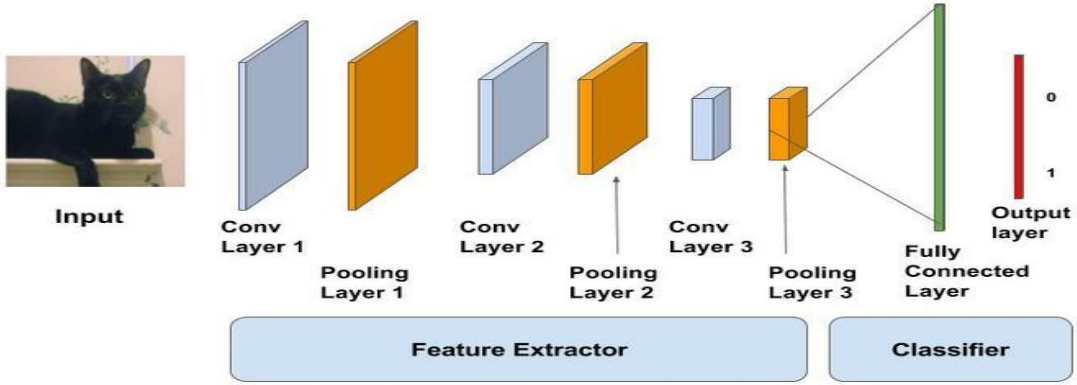


Architecture du Inception V3

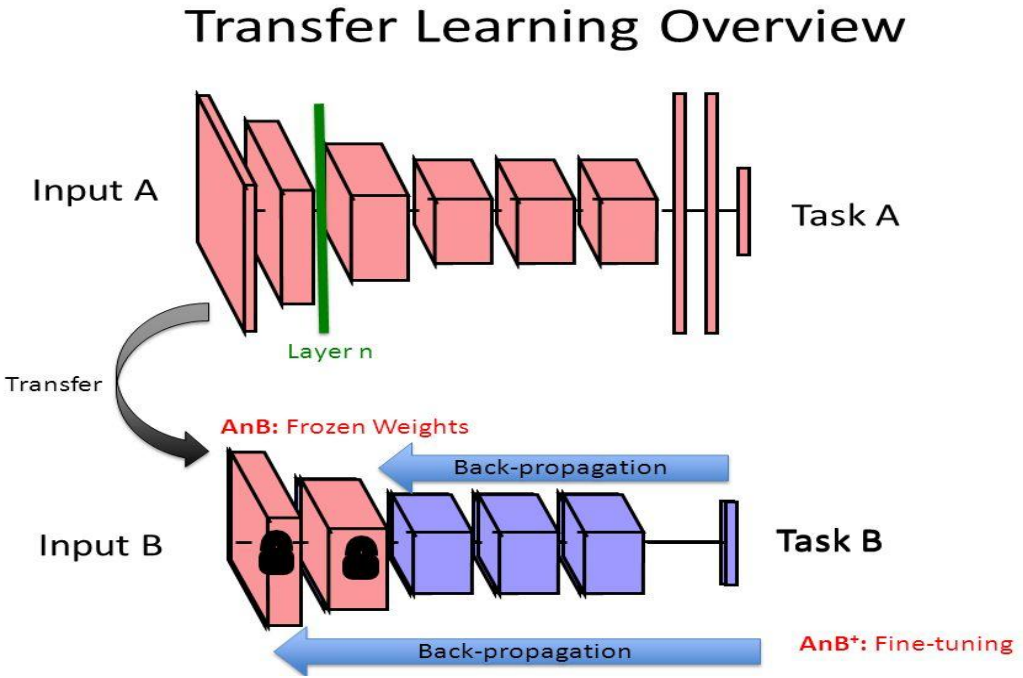
Pourquoi utiliser un CNN pré-entraîné ? Le Transfer Learning

> En pratique, très peu de gens entraînent un CNN de zéro car nécessite t**rès grande base de données + machine performante**

> Du coup, utilisation d'un CNN pré-entraîné sur grande base de données (ex : ImageNET : 1,2 millions d'images, 1000 catégories) et utilisation de ce CNN comme feature extractor => principe du **transfer learning**



Différentes utilisations possibles d'un CNN pré-entraîné



Principe du fine-tuning

1) En tant que **feature-extractor fixe** : on ne garde que les couches profondes qui **détectent des patterns très génériques**

2) **Fine tuning** : Autorisation de la rétropropagation : modifs des poids des neurones => **Rendre le CNN plus spécifique à son problème**

3.3.4 Résultats du CNN pré-entraîné Inception V3

> On a utilisé les CNN pré-entraînés en **feature extractor fixe** car **le nouveau dataset est petit et similaire à l'original**, pour **éviter l'over-fitting**

Méthode	**Accuracy**	**Logloss**
CNN pré-entraîné Inception-V3	0.92	0.33
CNN pré-entraîné XCeption	0.95	0.26
CNN pré-entraîné VGG-16	0.95	0.34

4 - Combinaison de CNN pré-entraînés

> Principe : combiner les sorties des différents CNN pré-entraînés, par exemple en faisant la moyenne.

> Choix des pondérations pour les CNN : méthode expérimentale et heuristique -> on fait des tests et on observe ce qui marche le mieux !

Méthode	**LogLoss**
Combinaison XCeption + Inception V3 (½ ; ½)	0.21
Combinaison XCeption + Inception V3 + VGG-16 (⅓ ; ⅓; ⅓)	0.14

> Le gagnant de la compétition a combiné une vingtaine de CNN pré-entraînés !

Merci de votre attention !