Rapport du projet filé Kaggle

Naoto LUC S, Jean-Baptiste GAENG, Ghada MEFTAH, Julien DANH

May 28, 2018

Table des matières

1 Introduction : le problème de classification en Machine Learning

En Machine Learning, un problème de classification consiste à donner des étiquettes à un ensemble de données. On dispose d'une part d'un ensemble de données connues que l'on a déjà classé (photos, plantes, individus...) : c'est l'ensemble d'entraînement (training set) qui permet d'entraîner un modèle. On souhaite ensuite classer de nouveaux éléments dont les étiquettes sont inconnues : c'est l'ensemble test (test set).

2 Première approche du Machine Learning : le problème du Titanic

2.1 Présentation du problème

Le site Kaggle propose un jeu de données comme projet d'initiation autour du Titanic. Le but de ce projet consiste à créer un modèle de prédiction de la survie des passagers du Titanic suite à son naufrage. C'est également une première application concrète du Machine Learning sur un jeu de données réel. Le jeu de données contient initialement les données des 1309 passagers du Titanic, réparties sur 12 variables.

2.2 Objectifs

2.2.1 But envisagé

Dans ce projet, nous devons analyser les types de personnes susceptibles de survivre au naufrage du Titanic. En particulier, nous allons appliquer les outils de Machine Learning pour prédire quels passagers ont survécu à la tragédie.

2.2.2 Moyens mis en oeuvre pour y parvenir

Nous avons suivi le tutoriel fourni par un kernel qu'on peut retrouver à l'adresse suivante :https://www.kaggle.com/startupsci/titanic-data-science-solutions/notebook.

2.3 Approches/Méthodes

2.3.1 Description des données

Les données rassemblent le nom et certaines caractéristiques des 1309 passagers du Titanic ('PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'), divisées en deux ensembles :

- Les données d'entraînement (train.csv), comprenant 891 passagers, avec pour chacun d'entre eux un flag (0 ou 1) indiquant s'il a survécu au naufrage.
- Les données de test (test.csv), comprenant les 418 passagers restants, pour lesquels l'objectif est de prédire s'ils ont survécu ou non.

2.3.2 Récupération des informations

La librairie Pandas nous permet facilement de charger et de manipuler les données sous forme structurée (Pandas DataFrames), et faire référence aux colonnes de données (les features) par leur nom. Nous avons choisi de fusionner les deux jeux de données pour pouvoir y effectuer certaines opérations simultanément.

2.3.3 Analyse des features

A ce stade, l'analyse des features nous permet de distinguer les variables catégoriques des numériques, et aussi de détecter les variables ayant des données manquantes commme : Cabin, Embarked et Age. En effet, les algorithmes de machine learning (ML) ne peuvent pas travailler sur des données vides (ou null).

Cette étape s'achève par la recherche de corrélations entre les variables, et on va s'intéresser principalement à la colonne Survived puisqu'il s'agit de notre variable à expliquer.

2.3.4 Préparation des données

Nous avons utilisé un algorithme d'apprentissage supervisé, il a donc fallu séparer les labels des données.
En se basant sur l'analyse faite précédemment, la préparation des données consiste d'abord à retirer les colonnes Name, Ticket, Cabin, Embarked, et PassengerId qui n'expliquent pas la variable Survived, puis cette variable car on en a déjà récupéré les étiquettes(labels). Ensuite, on a encodé la variable Sex en binaire, et on a rempli les valeurs d'âge manquantes avec l'âge médian. Toutes ces opérations ont été faites avec des fonctions simples de la librairie Scikit-Learn.
Les données étant conformes à ce qu'attendent les algorithmes de (ML),on peut entrainer le modèle et ainsi prédire les résultats qu'on souhaite obtenir.

2.3.5 Choix et entrainement d'un modèle

Comme il s'agit d'un problème de classification et de régression, on limite le choix des méthodes à utiliser à des algorithmes supervisés de la bibilothèque Sckit-learn selon le protocole suivant:

```
from sklearn.* import Method
meth = Method()
meth.fit(X_train, Y_train)
Y_pred = meth.predict(X_test)
acc_meth = round(meth.score(X_train, Y_train) * 100, 2)
acc_meth
```

Les algorithmes qu'on a testés sont :

- Logistic Regression `LogisticRegression()`
- KNN or k-Nearest Neighbors `KNeighborsClassifier(n_neighbors = 3)`
- Support Vector Machines(SVM) `SVC()`
- Naive Bayes classifier `GaussianNB()`
- Decision Tree `DecisionTreeClassifier()`
- Random Forest `RandomForestClassifier(n_estimators=100)`
- Perceptron `Perceptron()`
- Artificial neural network `SGDClassifier()`
- RVM or Relevance Vector Machine `LinearSVC()`

2.3.5.1 Classifieur de Bayes naïf

L'algorithme de classification naîve Bayesienne est l'un des algorithmes de plus simples pour les problèmes de classification. Comme son nom l'indique, l'algotithme utilise la formule du théorème de Bayes $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ pour calculer un résultat sur une variable que l'on cherche à prédire en se basant sur plusieurs variables prédictives.

L'application du théorème de Bayes sur plusieurs variables rend le calcul complexe. Pour contourner cela, une approche consiste à faire l'hypothèse que ces variables sont indépendantes les unes des autres. Il s'agit d'une hypothèse forte. Généralement, les variables prédictives sont liées entre elles. Le terme "naïf" vient du fait qu'on suppose cette indépendance des variables.

2.3.5.2 Descente de Gradient Stochastique

L'algorithme de gradient stochastique est un algorithme itératif permettant de minimiser une fonction dite "de coût d'erreur" associée à un modèle de régression particulier.

Si l'on prend l'exemple de la régression linéaire univariée, on cherche à trouver une fonction de prédiction $h(x)$, x étant la variable prédictive. Cette fonction sera une droite qui s'approchera le plus possible des données d'apprentissage, et s'écrira sous la forme : $h(x) = \theta_0 + \theta_1 x$ avec $\theta_0 et \theta_1$ les coefficients de la droite.

On peut montrer que pour un tel modèle, les valeurs de θ_0 et de θ_1 peuvent être calculés en minimisant la fonction de coût d'erreur $J(\theta_0, \theta_1)$ définie comme suit :

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=0}^{m} (\theta_0 + \theta_1 x_i - y_i)^2$$

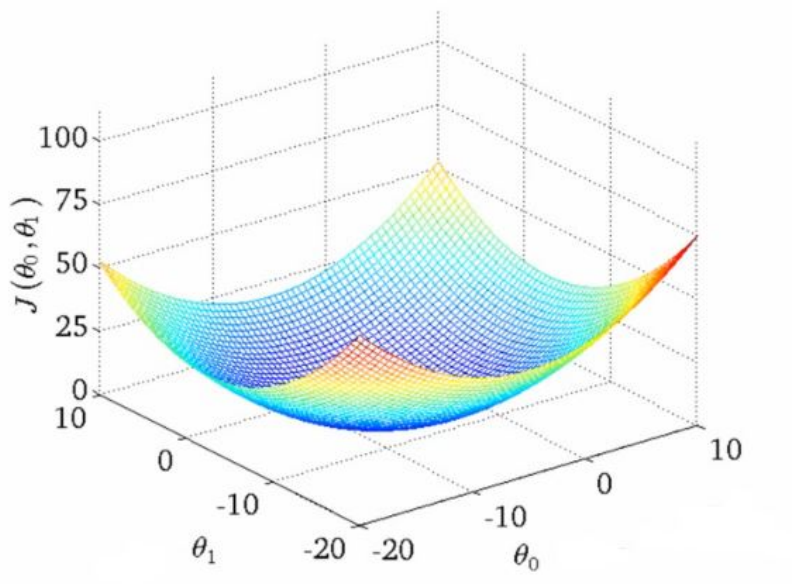

Figure 1: Graphe de la fonction de coût d'erreur J(θ_0, θ_1) pour le modèle de régression linéaire univariée

Une manière de calculer le minimum de la fonction de coût J(θ_0,θ_1) est d'utiliser l'algorithme de descente du gradient (Gradient descent). C'est un algorithme itératif qui va changer, à chaque itération, les valeurs de θ_0 et θ_1 jusqu'à trouver le meilleur couple possible.

Pseudo-code de l'algorithme :

1. Initialiser aléatoirement les valeurs de : θ_0 et θ_1
2. répéter jusqu'à convergence au minimum global de la fonction de coût
   - $pour j \in N \wedge \forall j \in \{0, 1\}$

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

3. retourner θ_0 et θ_1
4. Fin algorithme

Intuitivement, l'algorithme consiste à chaque pas de l'itération à regarder les points proches du point courant afin de trouver la meilleure pente pour avancer vers le bas.Une fois que la pente est trouvée, on avance d'un pas de grandeur α.

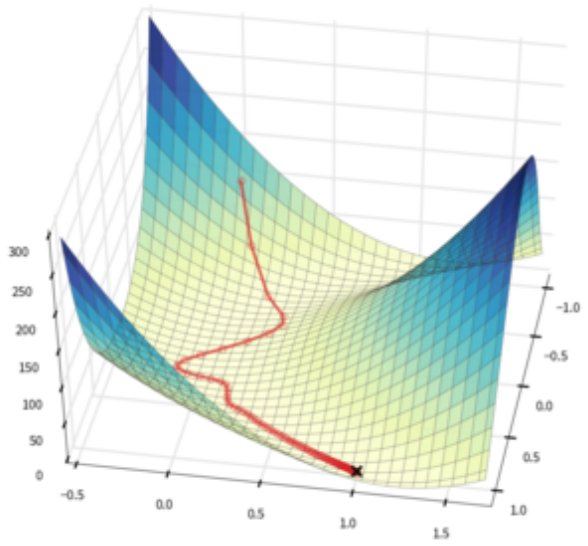

Figure 2: Schéma illustrant les itérations de l'algorithme de descente de gradient sur une fonction de coût quelconque

Le terme α s'appelle le Learning Rate : il fixe la "grandeur" du pas de chaque itération de la descente de Gradient.
De manière générale, si le dataset est très important, une itération de l'algorithme peut prendre beaucoup de temps (car on doit calculer le gradient en passant la totalité des observations). C'est pourquoi une méthode consiste à n'utiliser qu'un certain groupe d'observations (exemples) tiré aléatoirement à chaque itération de l'algorithme pour calculer le gradient : on parle de lot. La descente du gradient stochastique consiste à prendre un lot égal à 1.

2.3.5.3 Perceptron

D'après Wikipédia, le perceptron est un algorithme de classifieurs binaires. Il s'agit d'un neurone formel muni d'une règle d'apprentissage qui permet de déterminer automatiquement les poids synaptiques de manière à séparer l'ensemble en deux classes.

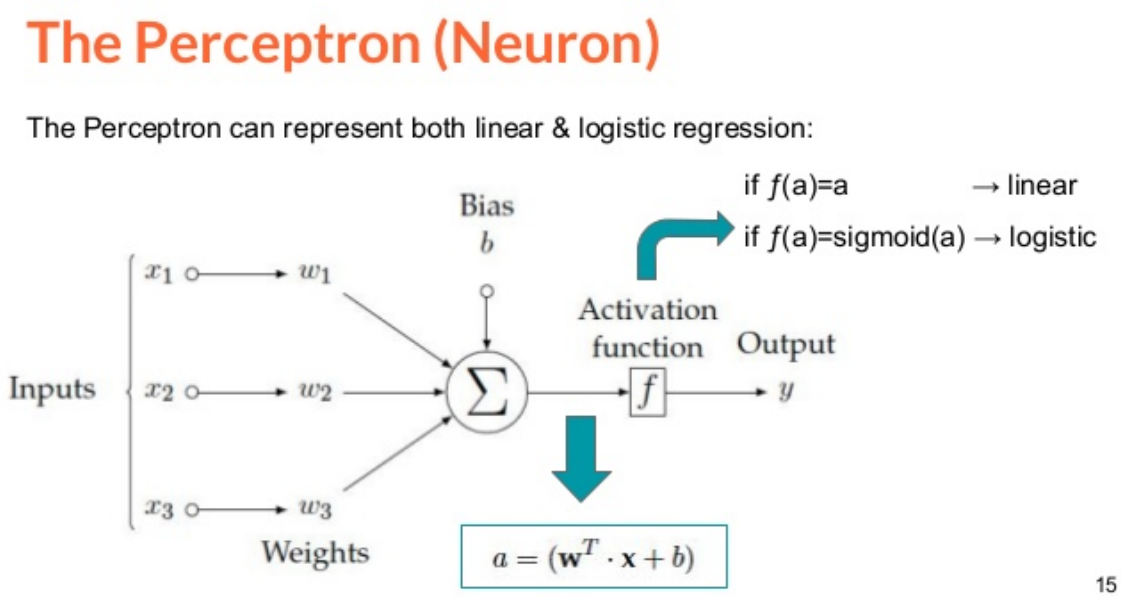

Figure 3: Principe du perceptron

2.3.5.4 Régression Logistique

La régression logistique est un modèle de régression binomiale. Elle prend en entrée des variables prédictives et mesure la probabilité de la valeur de sortie en utilisant la fonction sigmoïd.

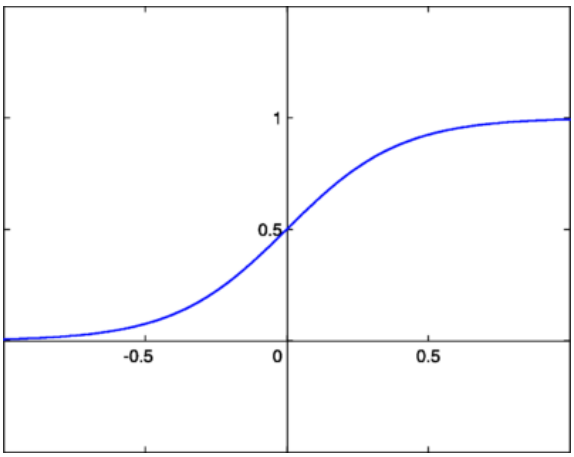

Figure 4: Graphe de la fonction sigmoïd

On peut utiliser cette méthode pour effectuer une classification multi-classes ou avec la méthode un-contre-tous (One-Versus-All). Pour plus de détails, on consultera la page web : https://mrmint.fr/logistic-regression-machine-learning-introduction-simple

2.3.5.5 Machine à Vecteur de Support (SVM)

La méthode de Machine à Vecteurs de Support ou Séparateur à Large Marge (SVM) sont un ensemble de techniques destinées à résoudre notamment des problèmes de régression. Les SVM sont une généralisation des classifieurs linéaires. Intuitivement, la méthode consiste à trouver un hyperplan séparateur pour un ensemble de données de départ.
Dans la figure suivante, nous avons deux classes. La régression Logistique pourra séparer ces deux classes en définissant le trait en rouge. le SVM va opter à séparer les deux classes par le trait vert.

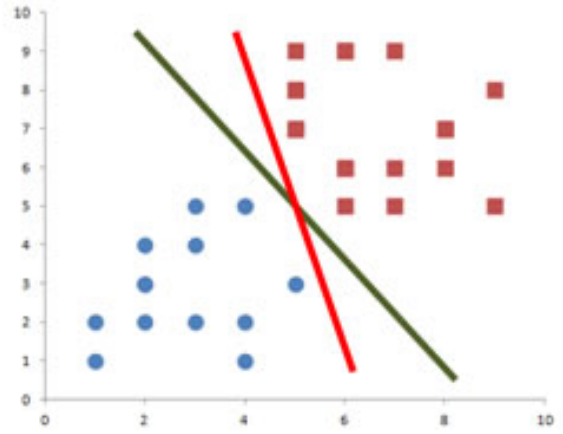

Figure 5: Séparation de deux classes par Régression Logistique (rouge) et par SVM (vert)

Sans entrer dans les détails, et pour des considérations mathématiques, le SVM choisira la séparation la plus nette possible (où les marges sont les plus importantes) entre les deux classes (comme le trait vert).

2.3.5.6 K plus proches voisins (KNN)

La méthode des K plus proches voisins ou K Neareast Neighbors (KNN) consiste à classer une nouvelle entrée par un vote majoritaire de ses voisins dans le dataset de départ.
La nouvelle entrée est assignée à la classe la plus commune parmi ses k voisins les plus proches (k est un nombre entier positif, typiquement petit). Si k = 1, alors l'objet est simplement assigné à la classe de ce voisin le plus proche.

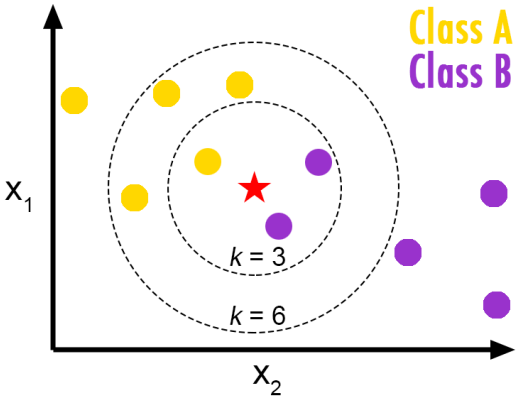

Figure 6: Méthode des k plus proches voisins. Si k = 3, la classe de étoile (nouvelle entrée) est assignée à B. Si k = 6, la classe de étoile est assignée à A

La distance considérée (valeur de k) est un paramètre à définir, de même que le stockage du dataset qui doit être efficace pour permettre une telle méthode, car on doit effectuer des recherche dans la totalité du dataset pour chaque nouvelle entrée. Des arbres de recherche sont souvent utilisés.

2.3.5.7 Arbres de décisions

Selon Wikipédia, l'apprentissage par arbre de décision désigne une méthode basée sur l'utilisation d'un arbre de décision comme modèle prédictif.
Dans ces structures d'arbre, les feuilles représentent les valeurs de la variable-cible et les embranchements correspondent à des combinaisons de variables d'entrée qui mènent à ces valeurs.

2.3.5.8 Forêt Aléatoire

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.

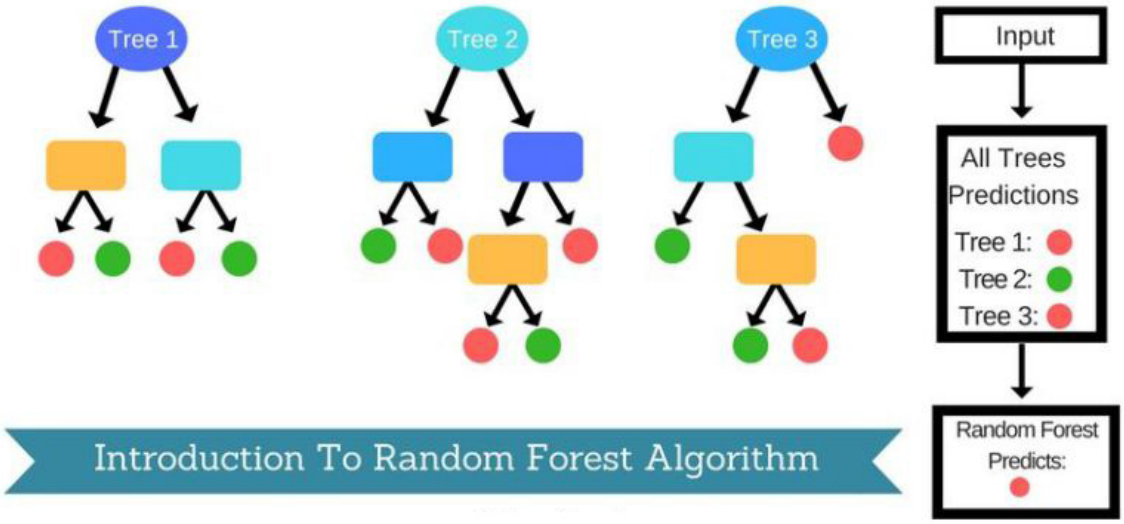

Figure 7: Principe de l'algorithme de la Forêt Aléatoire

2.4 Résultats obtenus

Pour comparer le pertinence des algorithmes, on a choisi la métrique *accuracy* $*$ 100 qui mesure la qualité de la prédiction d'une régression linéaire. L'évaluation de tous les modèles selon leur précision, nous a amené à choisir entre la méthode de Random Forest et celle de la Decision Tree. Afin d'éviter le risque du surapprentissage, on a opté plutôt pour la première méthode.

	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

Figure 8: Tableau récapitulatif des scores des différentes méthodes expérimentées

En utilisant donc l'algorithme de Random Forest, le score obtenu sur les données du test est 0.76555. Il est inférieur au score sur le training set, mais il reste raisonable et nous pouvons dire que l'algorithme arrive à peu près à prédire la survie des passagers, sachant que la chance entre aussi en jeu pour ce problème, même si une étude plus approfondie aurait permis un meilleur score.

2.5 Bilan sur ce premier projet

Ce premier tutoriel nous a permis d'appréhender les premiers algorithmes du ML, et de comprendre la phase de préparation de données qui occupe une bonne part de l'analyse. Le but étant de se familiariser avec ces algorithmes pour pouvoir les utiliser dans le jeu de données choisi : Cats vs Dogs.

3 Compétition Kaggle : classification de chiens et de chats

3.1 Présentation du problème

Nous avons choisi une compétition Kaggle reposant sur un problème de classification binaire avec des images de chiens et de chats. Le sujet de la compétition s'appelle "Dogs vs. Cats Redux: Kernels Edition" et peut être consulté à l'adresse suivante : https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition
Les données fournies pour le projet sont les suivants :

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

3.2 Objectifs

3.2.1 Buts envisagés

Le critère d'évalutation pour ce projet est la LogLoss. C'est une mesure de la perte d'information engendrée par la prédiction. Plus elle est faible, mieux c'est.

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

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.

Figure 9: Critère d'avaluation du projet : la LogLoss

Notre objectif est donc de minimiser cette fonction en testant différents algorithme de Machine Learning.

3.2.2 Moyens mis en oeuvre pour y parvenir

Pour avoir des références de LogLoss, nous avons tout d'abord testé des algorithmes simples de classifications, en nous basant sur nos connaissances acquises lors du projet Titanic. Pour les méthodes plus avancées telles que les réseaux de neurones (non entraînés ou pré-entraînés), nous nous sommes aidés de Kernels disponibles sur le site de la compétition Kaggle. Les différents Kernels utilisés sont cités dans les sections suivantes adéquates.

3.3 Approches/Méthodes

3.3.1 Un mot sur le data processing

Afin d'être plus efficace concernant la complexité des algorithmes, nous allons compresser les images en les redimenssionnant. Dans la suite, nous les redimensionnerons généralement en taille 64x64. La librairie utilisée pour traiter les images est OpenCV.

3.3.2 Résolution du problème : méthodes naïves

Nous avons voulu dans un premier temps commencer par résoudre le problème à l'aide de méthodes naïves, afin d'avoir des références de Logloss sur lesquelles se situer ensuite. Ainsi, nous proposons dans la suite une liste des méthodes que nous avons pu expérimenter, parmi lesquelles :

- Classificateur toujours incertain
- Régression logistique
- Forêt Aléatoire
- Gradient Boosting

3.3.2.1 Classifieur toujours incertain

La méthode de classification la plus naïve consiste à ne pas classer les images, c'est à dire à proposer en guise de predictions pour chaque image du jeu d'images tests des probabilités égales à 0,5.
En remplacant les $\widehat{y_i}$ dans la formule de la Logloss donnée précédemment, on voit facilement que nous devrions obtenir un Logloss théorique égal à $\ln 2$.
Remarque : Pour avoir une référence de LogLoss non probabiliste, nous avons soumis un fichier de prédiction prédisant à coup sûr pour chaque image un chien. Ceci correspond à des probabilités toujours égales à 1.

3.3.2.2 Régression logistique et Forêt Aléatoire

Pour les explications concernant le principe de ces deux méthodes, se réferrer à la partie sur le projet Titanic.

3.3.2.3 Gradient Boosting

3.3.2.3.1 Principe de la méthode

L'algorithme de Gradient Boosting est une méthode particulière pour tester et entrainer les données. Il est alors ensuite possible de choisir n'importe quel type d'algorithme pour prédire les données, dont les arbres de décisions.
Si les arbres de décisions est la méthode choisie pour prédire les données, alors l'algorithme ressemble à celui de la forêt aléatoire, à la différence près que dans l'algorithme de la forêt aléatoire, on entraine et on test les données en utilisant le "bagging ou bootstrapping" alors que la méthode de Gradient Tree Boosting utilise le "boosting".

3.3.2.3.2 Différences entre le bagging et le boosting

Dans la méthode du bagging, on construit m ensemble de données $D_1 ... D_m$ en tirant aléatoirement avec remise des données issus de l'ensemble Train. On utilise ensuite ces m ensembles de données pour entrainer un modèle différent.
On donne ensuite la même entrée X à tous les modèles ainsi entraînés, le modèle final Y étant obtenus en faisant la moyenne des sorties.

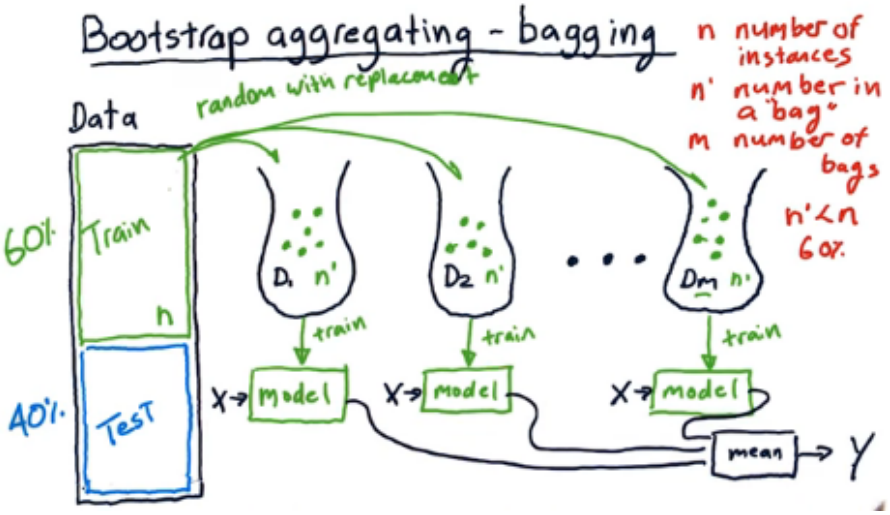

Figure 10: Principe de la méthode de Bagging pour entrainer et tester des données

Dans la méthode du boosting, on commence par construire le premier ensemble D_1 de données en en tirant aléatoirement dans l'ensemble Train. On utilise ensuite directement ces données pour entrainer un premier modèle. On utilise ensuite l'ensemble des données de Train pour tester ce modèle, ce qui va mettre en évidence que certaines données de l'ensemble Train sont mal prédites.

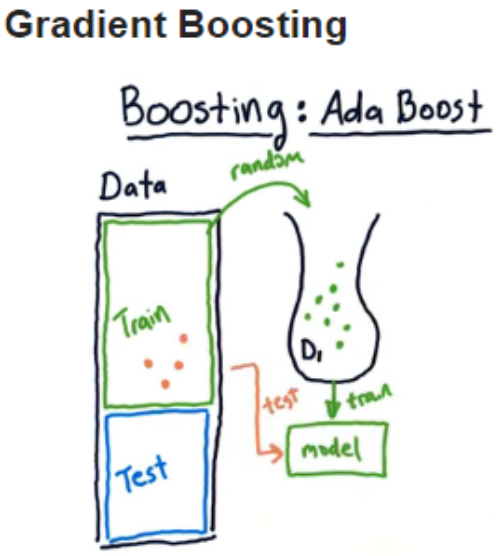

Figure 11: Principe de la méthode de Boosting pour entrainer et tester des données

Pour construire le deuxième ensemble de données D_2, on va retirer aléatoirement des données de l'ensemble Train, mais cette fois-ci on a pondéré les données mal prédites du modèle précédent de sorte que ces données aient plus de chance d'être tirées cette fois-ci. De la même manière que précédemment, on utilise ensuite les données de D_2 pour entrâiner un second modèle. Puis en donnant la même entrée aux modèles issus de D_1 et D_2, on obtient un modèle final Y intermédiaire, qui permet de mettre en évidence de nouvelles données mal prédites de l'ensemble Train. Ces nouvelles données auront plus de chance d'être tirées pour la construction de D_3, et ainsi de suite.

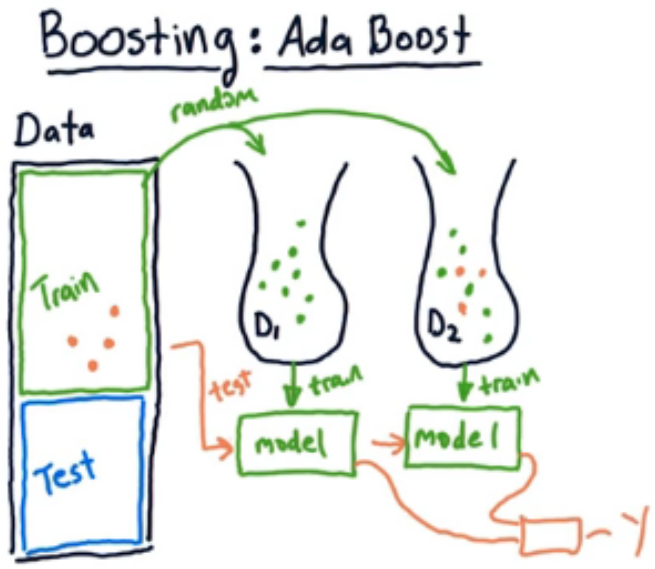

Figure 12: Principe de la méthode de Boosting pour entrainer et tester des données (suite)

On répète ce processus pour les m ensemble de données pour avoir le modèle Y final.

3.3.3 Résolution du problème : méthodes avancées

Comme nous pouvons le voir dans les paragraphes précédents, les méthodes naïves ne donnent pas de très bons résultats (pour une interprétation plus détaillée, cf. plus loin). Nous allons donc maintenant exposer des méthodes plus avancées pour résoudre ce problème, basées sur les réseaux de neurones.

3.3.3.1 Réseau de neurones non pré-entraînés

3.3.3.1.1 Réseau de neurones type Perceptron Multicouche

Le perceptron multicouche (multilayer perceptron MLP) est un type de réseau de neurones organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement ; il s'agit donc d'un réseau à propagation directe (feedforward). Chaque couche est constituée d'un nombre variable de neurones, les neurones de la dernière couche (dite « de sortie ») étant les sorties du système global.

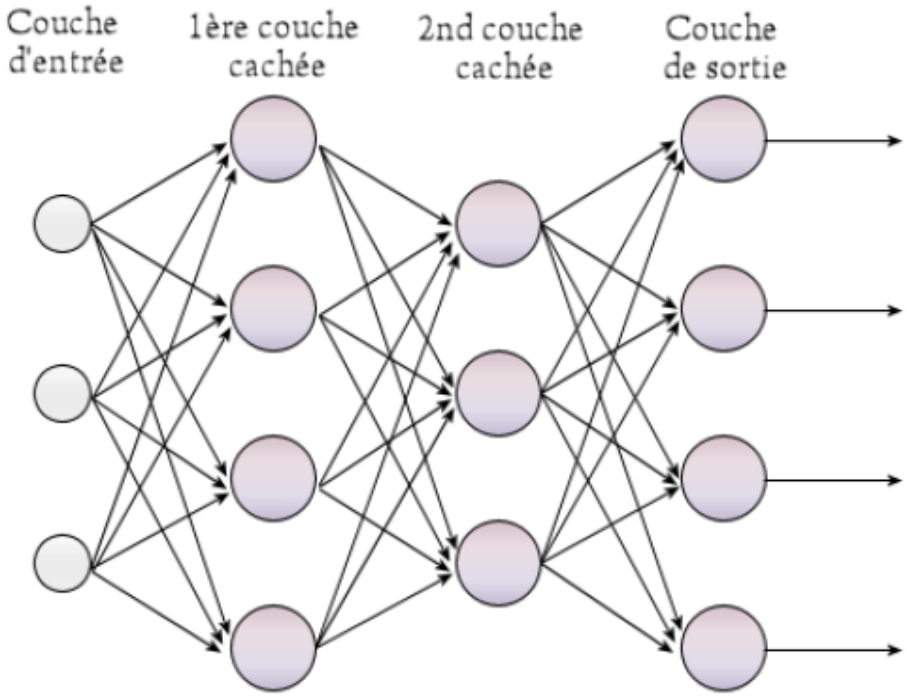

Figure 13: Schéma d'un Perceptron Multicouche

3.3.3.1.2 Réseau de neurones convolutifs (CNN)

D'après Wikipédia, en machine learning, un réseau de neurones convolutifs (en anglais CNN ou ConvNet pour Convolutional Neural Networks) est un type de réseau de neurones artificiels acycliques (feed-forward), dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Leur fonctionnement est inspiré par les processus biologiques, ils consistent en un empilage multicouche de perceptrons, dont le but est de prétraiter de petites quantités d'informations.
Pour mieux comprendre le fonctionnement des réseaux de neurones convolutifs, nous nous sommes basés sur un article de Brandon Rohrer (Senior Data Scientist à Facebook), intitulé : "How Convolutional Neural Networks Work". On peut trouver une traduction française de cet article ici : https://medium.com/@CharlesCrouspeyre/comment-les-r%C3%A9seaux-de-neurones-%C3%A0-convolution-fonctionnent-b288519dbcf8

3.3.3.1.3 Reseau de neurones convolutifs de type VGG-16

Pour cette méthode, nous nous sommes appuyés sur un Kernel de la compétition Kaggle consultable à l'adresse suivante : https://www.kaggle.com/jeffd23/catdognet-keras-convnet-starter
Le VGG-16 est un réseau de neurones convolutifs de 16 couches, crée par l'équipe Visual Geometry Group lors de la compétion Kaggle ImageNet de 2014 à laquelle elle termine première. Le schéma du VGG-16 est le suivant pour une image en couleurs de dimension 224x224:

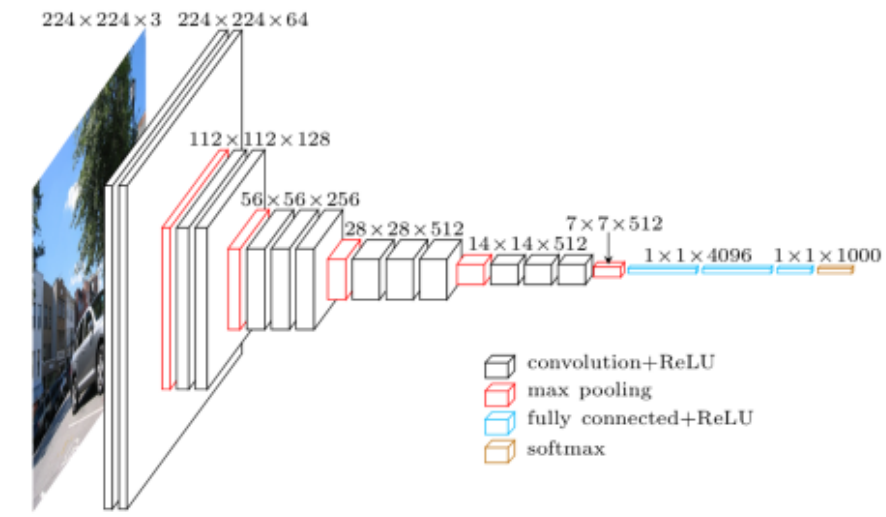

Figure 14: Schéma du VGG-16

- Le Pooling est une méthode permettant de prendre une large image et d'en réduire la taille tout en préservant les informations les plus importantes qu'elle contient. D'un point de vue mathématique, il suffit de faire glisser une petite fenêtre pas à pas sur toutes les parties de l'image et de prendre la valeur maximum de cette fenêtre à chaque pas.
- Le ReLU ou Unité linéaire rectifiée est une méthode consistant à 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 (mathématiquement parlant) en empêchant les valeurs apprises de rester coincées autour de 0 ou d'exploser vers l'infini.
- Après plusieurs couches de convolution et de max-pooling, le raisonnement de haut niveau dans le réseau de neurones se fait via des couches entièrement connectées (fully connected layers). Les neurones dans une couche entièrement connectée ont des connexions avec toutes les activations de la couche précédente. Leurs activations peuvent donc être calculées avec une multiplication matricielle.
- Le Softmax est une fonction d'activation logistique utilisée pour la classification multiclasse. La fonction Softmax est appliquée dans la dernière couche du réseau pour prendre la probabilité maximale des classes et pour prédire.

3.3.3.2 Un autre réseau de neurones convolutifs à 16 couches

Nous nous sommes appuyés sur le Kernel suivant :
https://www.kaggle.com/sentdex/full-classification-example-with-convnet. Le Kernel propose une autre configuration pour le réseau de neurones : il comporte néanmoins 16 couches comme le réseau de neurones précédent.

3.3.3.3 Réseau de neurones pré-entraînés et Transfer Learning

En pratique, très peu de personnes ré-entraînent un CNN de zéro car ce processus nécessite souvent du temps mais aussi une très grande base de données et machine performante. Du coup, il est courant de préférer l'utilisation de CNN déjà pré-entraîné sur de très grande base de données ressemblant aux siennes (exemple : base d'image ImageNET : 1,2 millions d'images, pour plus de 1000 catégories). On dit alors que l'on utilise ce CNN comme feature extractor. C'est le principe du transfer learning.

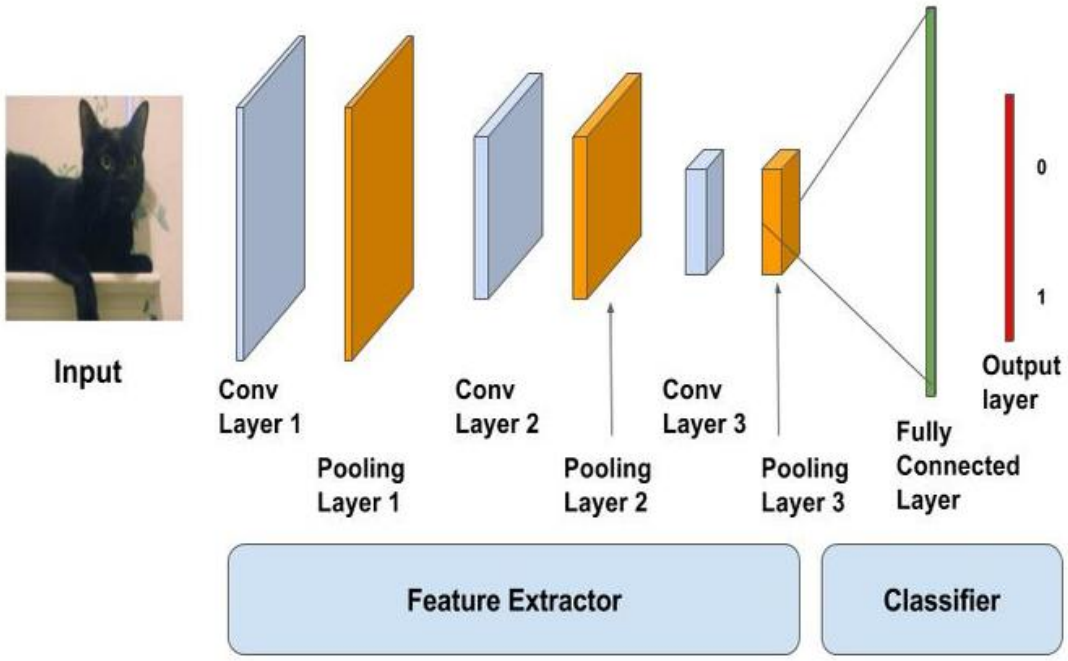

Figure 15: Principe du Transfer Learning

3.3.3.3.1 Différentes utilisations possibles d'un CNN pré-entraîné

Il est possible d'utiliser un CNN pré-entraîné de deux manières : en tant que feature extractor fixe ou en procédant à du fine-tuning.

- En tant que feature-extractor fixe : cela consiste à enlever les dernières couches du CNN (fully-connected layer), on se sert alors du reste du CNN comme feature extractor. En effet, les premières couches d'un CNN entraîné sur des images permettent de détecter des patterns très génériques (exemple : détections de contours), alors que les dernières couches sont plus spécifiques aux détails des classes contenues dans le dataset d'origine
- En procédant à du fine tuning : Cela consiste à rendre le CNN plus spécifique à son problème en autorisant la rétropropagation et la modification des poids des neurones à partir des dernières couches du CNN (on laisse les toutes premières couches fixes pour éviter le phénomène d'over-fitting)

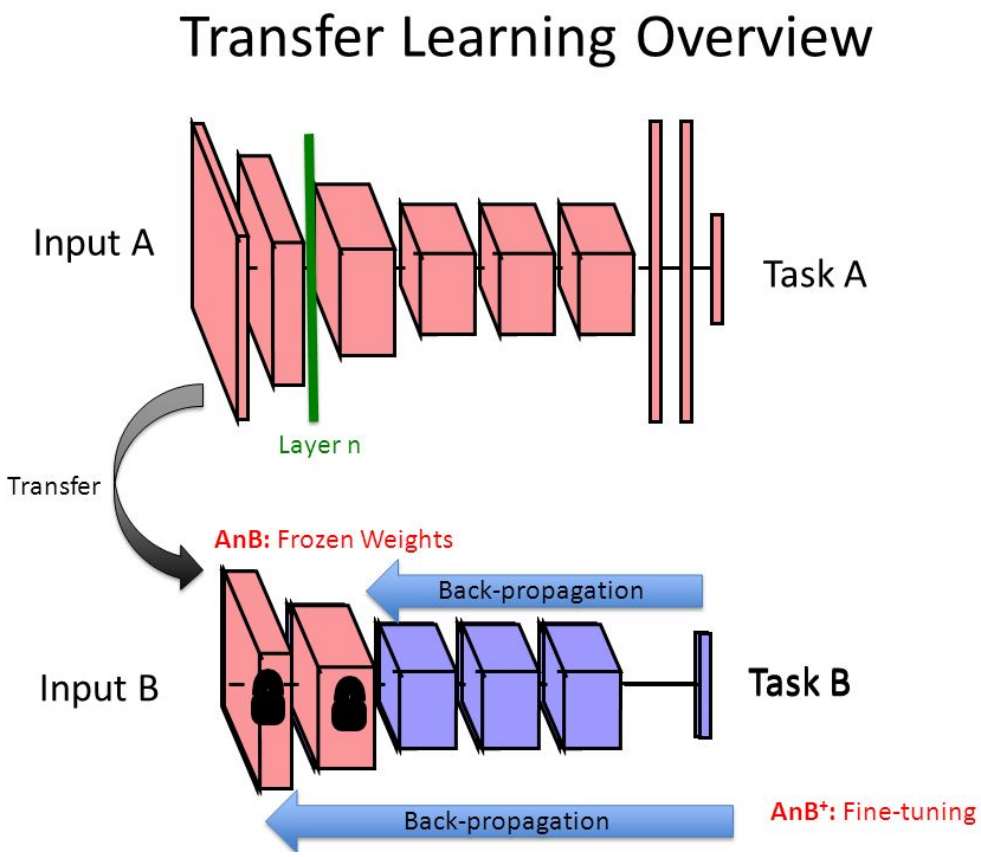

Figure 16: Principe du Fine-Tuning

3.3.3.3.2 Critère d'utilisation du fine-tuning

L'utilisation ou non du fine-tuning dépend de la taille du nouveau dataset (petit ou grand) et de sa ressemblance avec le dataset original qui a servi à pré-entraîner le CNN.
Exemple : Si le nouveau dataset est petit et similaire à l'original (c'est le cas de Dogs vs Cats) alors pas de fine-tuning pour éviter l'overfitting. De plus, si les nouvelles données sont similaires aux originales alors il est justifié de penser que les dernières couches du CNN extraient des features probablement pertinentes pour le nouveau dataset. On utilise donc dans ce cas là le CNN comme feature extractor fixe.

3.3.3.3.3 CNN pré-entraîné Inception V3

Nous avons utilisé les poids des neurones d'un CNN pré-entraîné appelé Inception V3, en nous basant sur le kernel suivant :
https://www.kaggle.com/tfrmarques/cats-vs-dogs-keras-pretrained-inception-v3
Ce réseau de neurones est utilisé dans notre cas car c'est l'un des réseaux de neurones les plus performants pour la reconnaissance d'image, classé parmi les meilleurs de la compétiton Kaggle ImageNet.

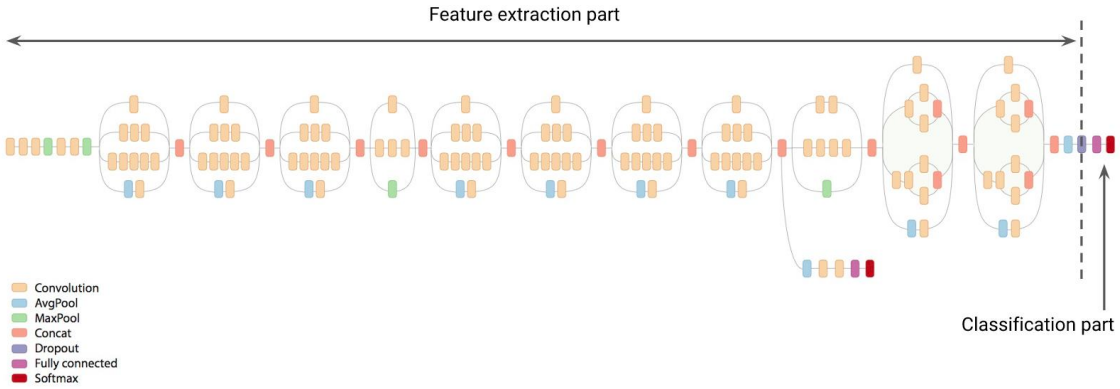

Figure 17: Architecture du CNN Inception V3

3.3.3.4 Combinaison de CNN pré-entraînés

Nous avons voulu combiner les résultats obtenus pour les différents réseaux de neurones pré-entraînés que nous avons testés à savoir : Inception V3, XCeption et VGG-16.
L'idée est simple, on effectue une combinaison linéaire des probabilités données par chacun des CNN, en choisissant les pondérations de manière expérimentale et heuristique. Il s'agit de tester et d'observer au fur et à mesure ce qui marche le mieux.
Nous avons pu soumettre à Kaggle une première combinaison de CNN pré-entraîné en faisant la moyenne des probabilités données par Inception V3 et XCeption (coefficient de pondératon : $(\frac{1}{2}\ ;\ \frac{1}{2})$) ainsi qu'une deuxième combinaison en faisant la moyenne des probabilités données par Inception V3, XCeption et VGG-16 (coefficient de pondératon : $(\frac{1}{3}\ ;\ \frac{1}{3}\ ;\ \frac{1}{3})$).

3.4 Résultats obtenus

3.4.1 Résultats obtenus pour les différentes méthodes

Nous avons rassemblé nos différents résultats sur la figure suivante

• Méthodes naïves •	Accuracy	Score en LogLoss
Classification toujours incertaine	0.50	0.69
Régression Logistique	0.68	0.66
Forêt Aléatoire	0.98	17.57
Gradient Boosting	0.70	17.61
• Méthodes Avancées •		
Perceptron Multicouche	-	0.72
CNN type VGG-16	0.71	0.50
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
Combinaison CNN pré-entraînés XCeption + Inception-V3 (½ ; ½)	-	0.21
Combinaison CNN pré-entraînés XCeption + Inception-V3 + VGG-16 (1/3 ; 1/3 ; 1/3)	-	0.14

Figure 18: Tableau récapitulatif des résultats obtenus

Comme nous l'avons expliqué dans la partie sur les méthodes utilisées, nous avons d'abord voulu avoir des références de LogLoss en implémentant des méthodes très naïves comme le classifieur

toujours incertain ou bien en soumettant un fichier de prédiction prédisant à coup sûr pour chaque image un chien (ceci correspond à des probabilités toujours égales à 1).
Pour ce qui est de la classification toujours incertaine, après soumission à Kaggle, on obtient un score de Logloss effectivement égal à $\ln 2$ soit environ 0,69. En ce qui concerne le fichier de prédiction prédisant à coup sûr pour chaque image un chien, le score en LogLoss obtenu est de 17,27.
Ces deux scores obtenus nous ont servi de première référence pour savoir si une méthode de classification est efficace ou non pour ce problème.

3.4.2 Analyse et critique des résultats

Comme nous pouvons le constater grâce aux paragraphes précédents, les méthodes naïves comme la forêt aléatoire et la régression logistique donnent de très mauvais résultats. Pour l'algorithme de RandomForest, la précision obtenue sur le training set est très bonne, mais ce classifieur ne fonctionne pas sur le test. Il semble y avoir un problème d'overfitting. Ces mauvais résultats viennent du fait que les méthodes autres que la méthode des réseaux de neurones convolutifs demandent un vecteur en entrée. Pour l'éxécution des différents algorithmes, il a donc fallu convertir les images de dimensions 3x64x64 en vecteurs. La cohérence spatiale que porte une image semble être difficilement retrouvable pour ces algorithmes.
La régression logistique est quant à elle un peu plus efficace que le hasard.

En ce qui concerne les méthodes avancées, le perceptron multicouche donne un résultat assez équivalent à la régression logistique, même si le score est meilleur sur les images de l'ensemble train que sur celles de l'ensemble test. Pour ce qui est des réseaux de neurones convolutifs, celui de type VGG-16 donne des résultats en deça du hasard. Ceci est plutôt étonnant et vient sans doute du manque d'entrainement de réseau ou bien de paramètres qu'il reste encore à ajuster. En effet, le code du Kernel a permis d'avoir, pour l'ensemble de la base de données, avec une dimension 64x64 et un backend TensorFlow un score de Logloss de 2,17. Voici les graphes de la Logloss obtenus ainsi que les prédictions pour les 10 premières images tests :

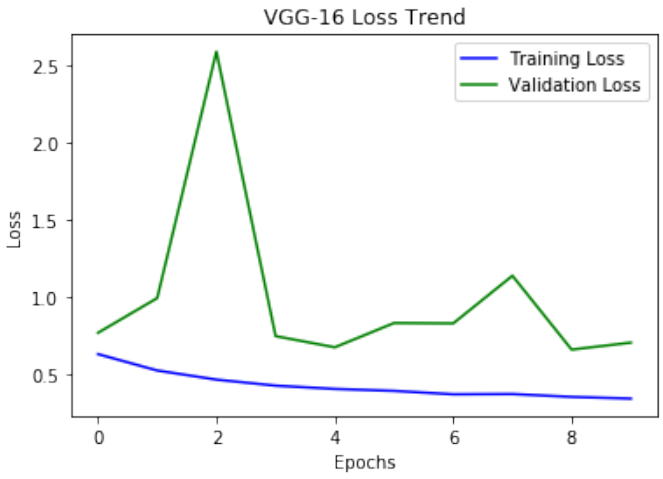

Figure 19: Graphe de la LogLoss - Image 64x64 - Backend TensorFlow

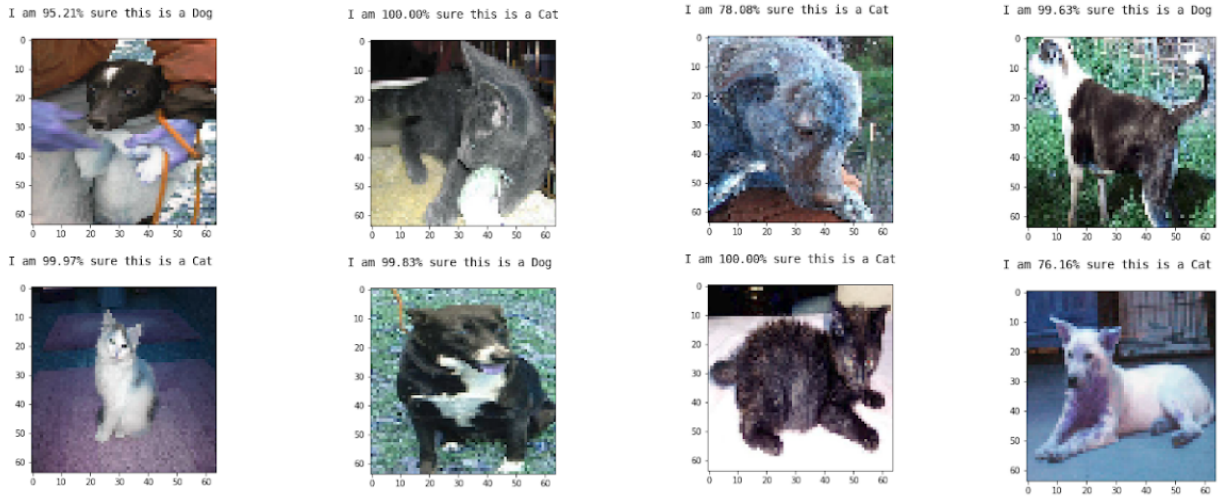

Figure 20: Prédictions obtenues pour les 10 premières images tests

On peut observer que le réseau prédit correctement ici 6 images sur 8 tirés aléatoirement parmi les images tests, ce qui correspond à 75% de bonne prédiction.
Pour l'ensemble de la base de données, avec des images de dimension 256x256, en utilisant un backend Theano, on obtient un score de LogLoss après soumission de 1,96 alors qu'il obtenait de bons résultats sur le training set.

Le deuxième réseau de neurones convolutifs à 16 couches donne quant à lui des résultats bien plus satisfaisants.

Les meilleurs scores de LogLoss sont obtenus en utilisant les réseaux de neurones pré-entraînés de Keras. On s'attendait déjà à un tel résultat car en pratique ces réseaux de neurones sont déjà entraînés sur d'immenses bases d'images et sont donc très performants pour ce type de problème, en témoigne leurs classements très élevés dans la compétition Kaggle de référence ImageNet. Les réseaux Inception V3, Xception et VGG16 ont été choisis en particulier car ils demandaient des entrées similaires. Les résultats obtenus sur ce problème sont très bons et assez semblables.

La combinaison des sorties des meilleures méthodes, c'est-à-dire des réseaux pré-entraînés ont permis d'obtenir un résultat encore meilleur(LogLoss de 0.14). Le score est meilleur que chacune des méthodes, cela montre que les algorithmes ont des points forts différents, et se trompent sur des images différentes. Cela est très intéressant car un meilleur réseau peut être obtenu en combinant les méthodes, et la moyenne de plusieurs de ces méthodes permet de réduire la conséquence d'une erreur sur le score.

3.5 Conclusion

Nous avons étudié différentes méthodes pour résoudre ce problème de classification d'images, et la combinaison de réseaux de neurones pré-entraînés a donné les meilleurs résultats. Nous avons appris les méthodes existantes pour résoudre ce problème, mais surtout tout le traitement des données qui doit être fait en amont pour pouvoir lancer les algorithmes, qui constitue une part très importante de l'expérimentation.