

Faculté des sciences et techniques MARRAKECH

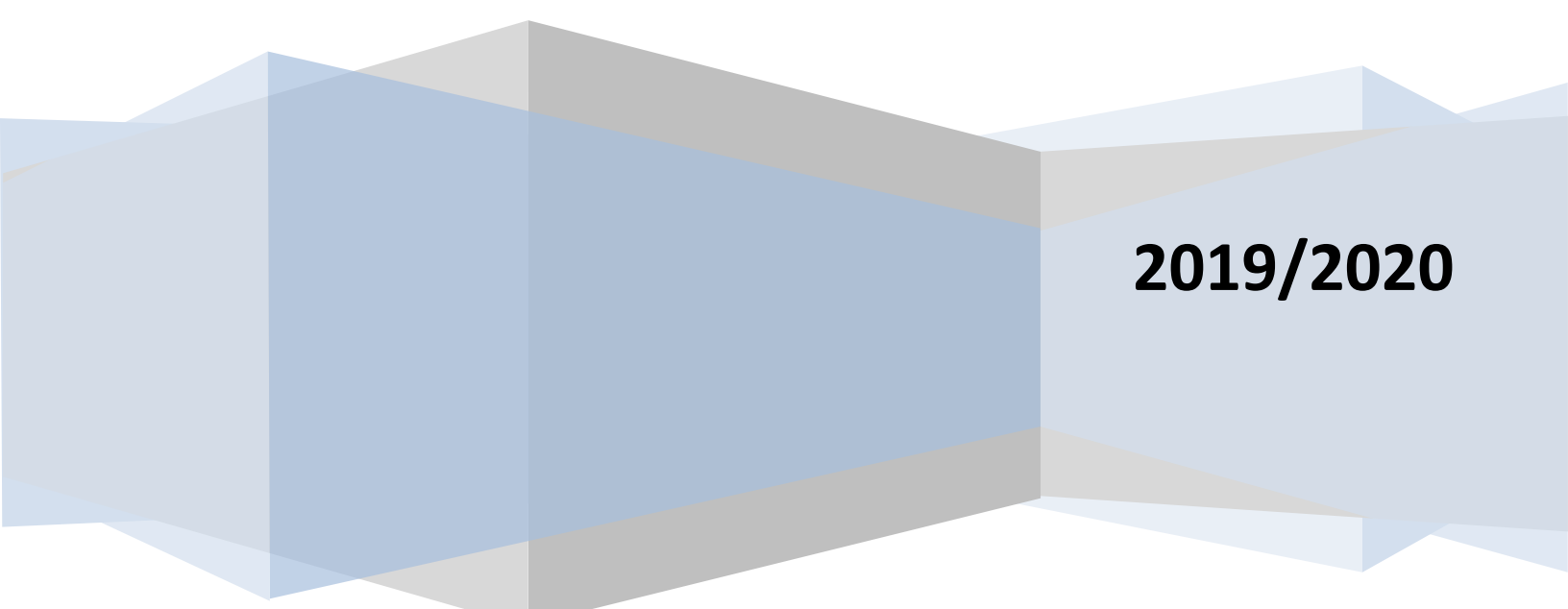
# **Mini-Projet : Classification des images**

**Intelligence Artificielle**

**Master SDAD**

**ELMRHARI Yassine**

**Enseignante : Mme K. Bouzaachane**



**2019/2020**

# Sommaire

## Table des matières

<b>Objectifs .....</b>	<b>4</b>
<b>Introduction .....</b>	<b>5</b>
<b>Présentation du dataset CIFAR-10 .....</b>	<b>6</b>
<b>Phase 1 : Apprentissage avec ML .....</b>	<b>7</b>
I. Présentation du Machine Learning: .....	7
II. Présentation de l'architecture de la phase 1: .....	7
III. Etapes d'exécution de la phase 1: .....	8
IV. Présentation des descripteurs utilisés: .....	8
1. Moments Hu (Descripteur de forme) : .....	8
2. Descripteurs Haralick (Descripteur de texture) : .....	10
3. Histogramme de couleur (Descripteur des couleurs) : .....	11
V. Présentation des algorithmes du Machine Learning utilisées : .....	12
1. KNN : .....	12
2. SVM : .....	13
3. Random Forest : .....	14
4. Linear Discriminant Analysis : .....	15
5. StackingClassifier: .....	17
VI. Résultats: .....	18
<b>Phase 2 : Apprentissage avec DL .....</b>	<b>19</b>
I. Présentation du Deep Learning: .....	19
II. Présentation de l'architecture de la phase 2: .....	19
1. Architecture générale: .....	19
2. Architecture du modèle: .....	20
3. Paramétrage du modèle: .....	21
4. Résumer du modèle: .....	25
III. Etapes d'exécution de la phase 2 : .....	25

IV. Résultat: .....	26
<b>Implémentation des phases .....</b>	<b>27</b>
I. Obtention des images test: .....	27
II. Prédiction de l'étiquette de chaque image suivant les deux modèles: .....	28
1. Importation de l'image: .....	28
2. Prédiction des classes: .....	29
3. Résultats: .....	30
<b>Conclusion .....</b>	<b>31</b>
<b>Références.....</b>	<b>32</b>

## Table des figures

<b>Figure 1 – Echantillons d'images de CIFAR-10.....</b>	<b>6</b>
<b>Figure 2 – Calcul des moments Hu pour différentes silhouettes d'images.....</b>	<b>10</b>
<b>Figure 3 – Extraction des textures suivant Haralick.....</b>	<b>11</b>
<b>Figure 4 – Exemple de pixels d'image.....</b>	<b>12</b>
<b>Figure 5 – Extraction des histogrammes de couleurs pour une image.....</b>	<b>12</b>
<b>Figure 6 – Le concept du Stacking .....</b>	<b>18</b>
<b>Figure 7 – Diagramme de précision des algorithmes de Machine Learning .....</b>	<b>19</b>
<b>Figure 8 – Courbes de précision et de loss du modèle de Deep Learning .....</b>	<b>27</b>

# Objectifs

- Construire un système intelligent formé avec un ensemble de données massif d'images de CIFAR-10.
- Réaliser un apprentissage suivant des algorithmes de Machine Learning.
- Réaliser un autre apprentissage suivant les réseaux neurones du Deep Learning.
- Développer une interface en python pour interagir avec les codes sources.
- Faire une comparaison entre les performances du Machine Learning et du Deep Learning.

# Introduction

La classification d'images est un problème fondamental en vision par ordinateur, qui a de nombreuses applications concrètes. Le but est de construire un système capable d'assigner correctement une catégorie à n'importe quelle image en entrée. Un tel système exploite des algorithmes de Machine Learning issus de l'apprentissage supervisé.

Dans cet article, on examinera un des problèmes de classification d'images, à savoir la reconnaissance des classes du dataset **CIFAR-10**. Comme on sait que le Machine Learning consiste à apprendre des données passées, on a besoin d'un énorme ensemble d'images pour effectuer la reconnaissance en temps réel.

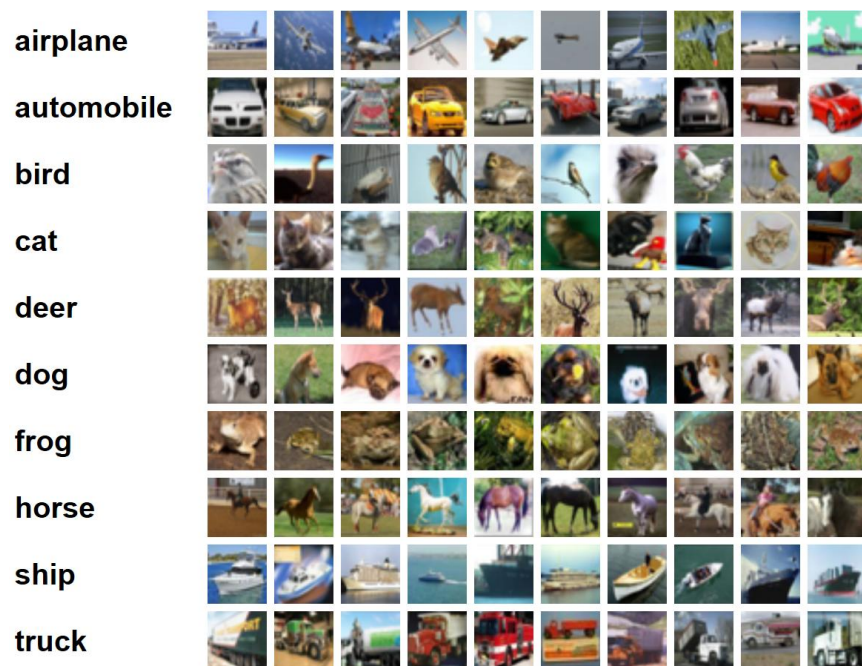
On apprendra comment effectuer une tâche de classification d'image en utilisant des algorithmes de vision par ordinateur et de Machine Learning ainsi que de Deep Learning à l'aide de Python.

# Présentation du dataset CIFAR-10

**CIFAR** est un acronyme qui signifie « Canadian Institute For Advanced Research » l'ensemble de données CIFAR-10 a été développé avec l'ensemble de données CIFAR-100 par des chercheurs de l'Institut CIFAR.

L'ensemble de données CIFAR-10 comprend 60000 images couleurs 32x32 en 10 classes, avec 6000 images par classe. Il y a 50000 images d'apprentissage et 10000 images de test.

Voici les classes de l'ensemble de données, ainsi que 10 images aléatoires de chacune:



**Figure 1** – Echantillons d'images de CIFAR-10

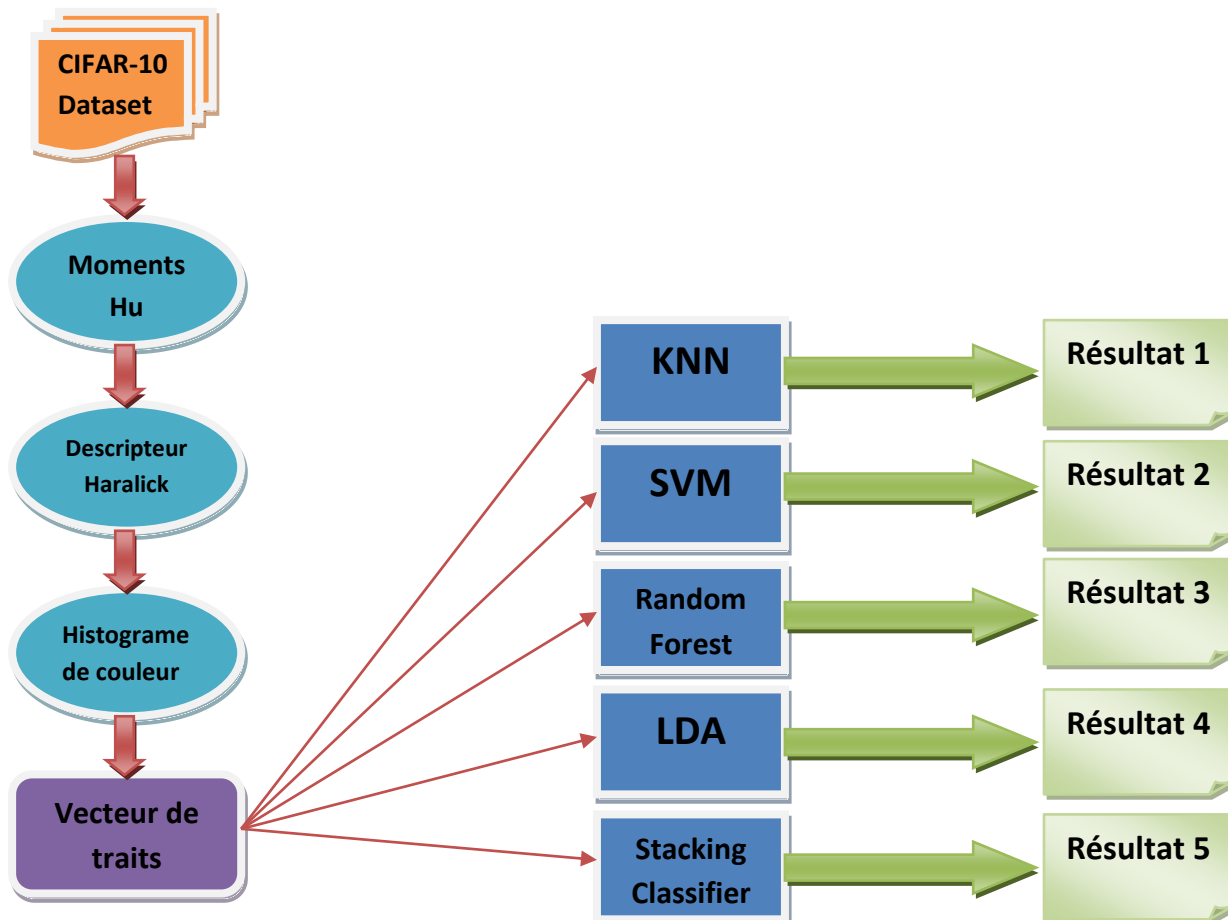
Ce sont de très petites images, beaucoup plus petites qu'une photographie typique destiné à la recherche en vision par ordinateur.

# Phase 1 : Apprentissage avec ML

## I. Présentation du Machine Learning :

Le Machine Learning est une technologie d'intelligence artificielle permettant aux ordinateurs d'apprendre sans avoir être programmés explicitement à cet effet. Pour apprendre et se développer, les ordinateurs ont toutefois besoin de données à analyser et sur lesquelles apprendre. De fait, il est nécessaire de bien choisir ses données pour avoir des résultats satisfaisants.

## II. Présentation de l'architecture de la phase 1:



### III. Etapes d'exécution de la phase 1 :

- ✓ Importation des images CIFAR-10.
- ✓ Calcul des descripteurs de forme/texture/couleur pour chaque image pour construire un vecteur de trait pour chaque image.
- ✓ Utilisation des vecteurs de traits pour implémenter les algorithmes de Machine Learning de classification.
- ✓ Evaluation du résultat de chaque algorithme.
- ✓ Sauvegarde du modèle dans un fichier .sav (pickle)

### IV. Présentation des descripteurs utilisés :

#### 1. Moments Hu (Descripteur de forme) :

Les moments Hu sont normalement extraits de la silhouette ou du contour d'un objet dans une image. En décrivant la silhouette ou le contour d'un objet, nous pouvons extraire un vecteur d'entité de forme (c'est-à-dire une liste de nombres) pour représenter la forme de l'objet. Nous pouvons ensuite comparer deux vecteurs de caractéristiques à l'aide d'une métrique de similitude ou d'une fonction de distance pour déterminer la similitude des formes.



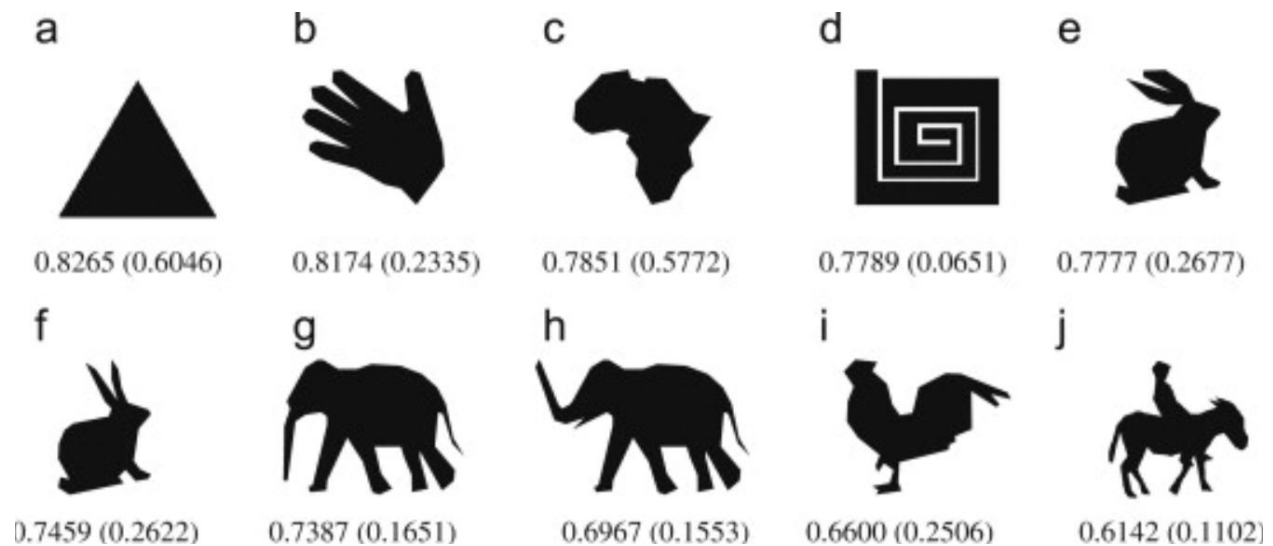
Ils sont un ensemble de 7 nombres calculés en utilisant des moments centraux invariants aux transformations d'image. Les 6 premiers moments se sont révélés invariants à la translation, à l'échelle, à la rotation et à la réflexion. Alors que le signe du 7ème moment change pour la réflexion de l'image.

$H_1 = \eta_{20} + \eta_{02}$ $H_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$ $H_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$ $H_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$ $H_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2]$ $+ (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$ $H_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$ $+ 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$ $H_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2]$ $+ (3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$	normalized central moments
--	----------------------------

$$\eta_{ij} = \frac{\mu_{i,j}}{\mu_{00}^{(i+j)/2+1}}$$

$$\mu_{ij} = \sum_x \sum_y (x - \bar{x})^i (y - \bar{y})^j I(x, y)$$

La figure ci-dessous montre le calcul des moments Hu pour différentes silhouettes d'images :

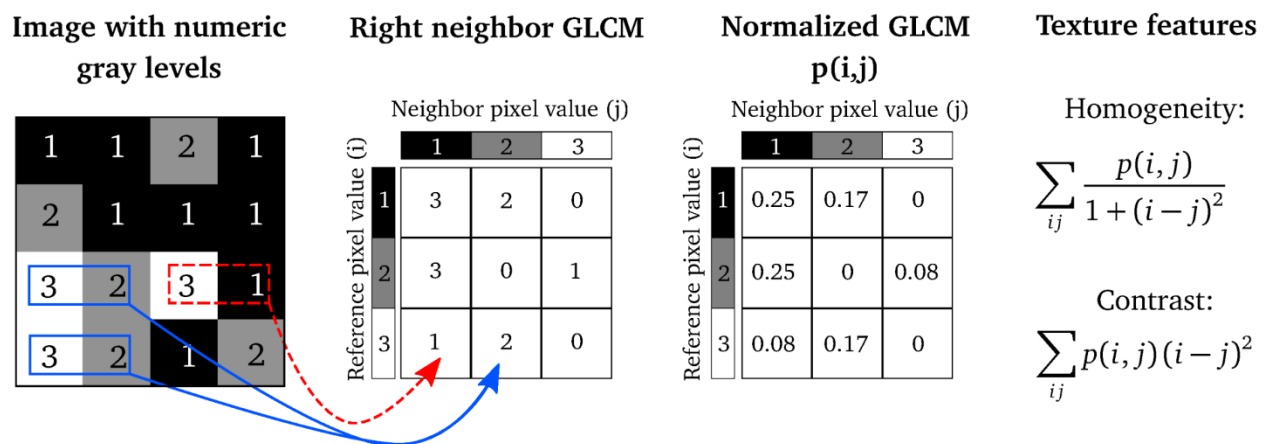


**Figure 2** – Calcul des moments Hu pour différentes silhouettes d'images

## 2. Descripteurs Haralick (Descripteur de texture) :

Dans une image  $4 \times 4$ , trois niveaux de gris sont représentés par des valeurs numériques de 1 à 3. Le GLCM est construit en considérant la relation de chaque voxel avec son voisinage. Dans cet exemple, par souci de simplicité, nous ne regardons que le voisin de droite. Le GLCM agit comme un compteur pour chaque combinaison de paires de niveaux de gris dans l'image. Pour chaque voxel, sa valeur et la valeur du voxel voisin sont comptées dans un élément GLCM spécifique. La valeur du voxel de référence détermine la colonne du GLCM et la valeur voisine détermine la ligne. Dans ce retour sur investissement, il y a deux cas où un voxel de référence de 3 «co-se produit» avec un voxel voisin de 2 (indiqué en solide, bleu), et il y a une instance d'un voxel de référence de 3 avec un voxel voisin de 1 (indiqué en pointillé, rouge). Le

GLCM normalisé représente la probabilité estimée que chaque combinaison se produise dans l'image. Les caractéristiques de texture Haralick sont des fonctions du GLCM normalisé, où différents aspects de la distribution des niveaux de gris dans le ROI sont représentés. Par exemple, les éléments diagonaux du GLCM représentent des paires de voxels avec des niveaux de gris égaux. La fonction de texture «contraste» donne aux éléments ayant des valeurs de niveaux de gris similaires un poids faible mais aux éléments ayant des niveaux de gris différents un poids élevé.



**Figure 3** – Extraction des textures suivant Haralick

### 3. Histogramme de couleur (Descripteur des couleurs) :

Un histogramme représente la distribution des couleurs dans une image. Il peut être visualisé sous forme de graphique (ou tracé) qui donne une intuition de haut niveau de la distribution d'intensité (valeur en pixels). Nous allons supposer un espace colorimétrique RGB dans cet exemple, donc ces valeurs de pixels seront comprises entre 0 et 255.

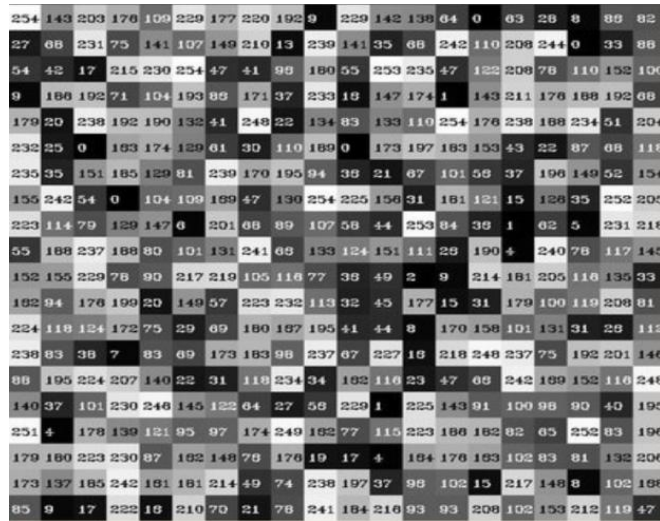


Figure 4 – Exemple de pixels d'image

La figure ci-dessous représente l'extraction des histogrammes de couleurs pour les composants R et G et B d'une image.

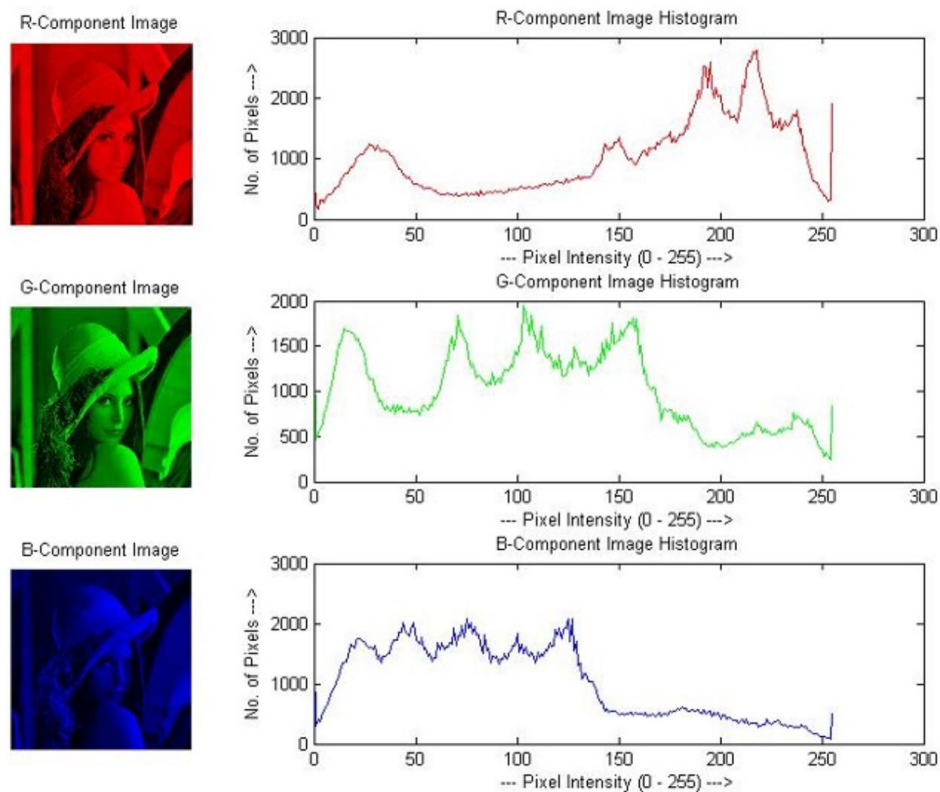


Figure 5 – Extraction des histogrammes de couleurs pour une image

## V. Présentation des algorithmes du Machine Learning utilisées :

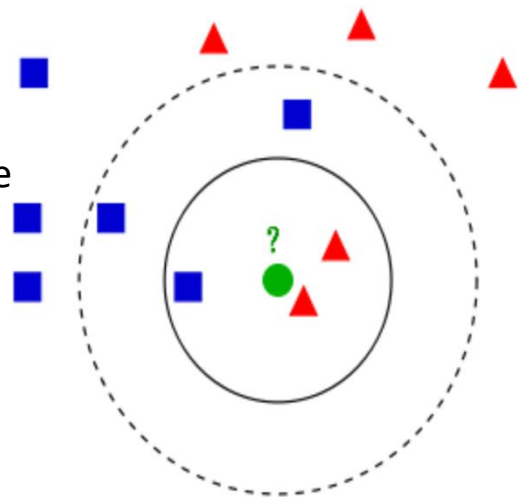
### 1. KNN :

L'algorithme K-NN (K-nearest neighbors) est une méthode d'apprentissage supervisé. Il peut être utilisé aussi bien pour la régression que pour la classification. Son fonctionnement peut être assimilé à l'analogie suivante "dis moi qui sont tes voisins, je te dirais qui tu es...".

Pour effectuer une prédiction, l'algorithme K-NN va se baser sur le jeu de données en entier. En effet, pour une observation, qui ne fait pas parti du jeu de données, qu'on souhaite prédire, l'algorithme va chercher les K instances du jeu de données les plus proches de notre observation. Ensuite pour ces K voisins, l'algorithme se basera sur leurs variables de sortie (output variable) y pour calculer la valeur de la variable y de l'observation qu'on souhaite prédire.

Quelle est la classe du point vert ?

Celle des triangles rouges (délimitée par le cercle continu) ou celle des carrés bleus (cercle tracé en pointillés) ? Si le nombre de plus proches voisins k est fixé à 3, la classe du point vert est celle des triangles

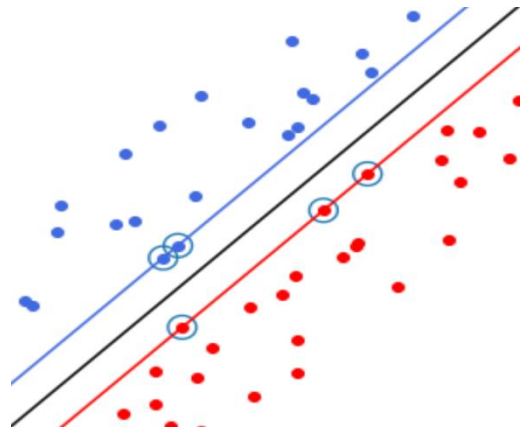


rouges, car ces derniers sont au nombre de 2 contre un seul carré bleu. Si  $k$  vaut 5, la classe du point vert est celle des carrés bleus, au nombre de 3 contre 2 triangles rouges.

## 2. SVM :

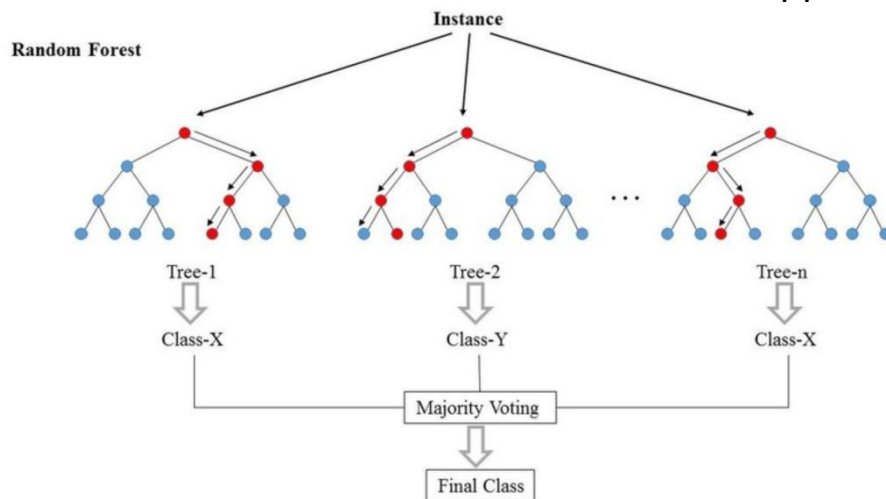
SVM (Support Vector Machine ou Machine à vecteurs de support) sont une famille d'algorithmes d'apprentissage automatique qui permettent de résoudre des problèmes tant de classification que de régression ou de détection d'anomalie.

Les SVMs ont été développés dans les années 1990. Comme le montre la figure ci-dessous, leur principe est simple : il ont pour but de séparer les données en classes à l'aide d'une frontière aussi « simple » que possible, de telle façon que la distance entre les différents groupes de données et la frontière qui les sépare soit maximale. Cette distance est aussi appelée « marge » et les SVMs sont ainsi qualifiés de « séparateurs à vaste marge », les « vecteurs de support » étant les données les plus proches de la frontière.



### 3. Random Forest :

L'algorithme des « forêts aléatoires » (ou Random Forest parfois aussi traduit par forêt d'arbres décisionnels) est un algorithme de classification qui réduit la variance des prévisions d'un arbre de décision seul, améliorant ainsi leurs performances. Pour cela, il combine de nombreux arbres de décisions dans une approche de type bagging.



L'algorithme des « forêts aléatoires » a été proposé par Leo Breiman et Adèle Cutler en 2001. Dans sa formule la plus classique, il effectue un apprentissage en parallèle sur de multiples arbres de décision construits aléatoirement et entraînés sur des sous-ensembles de données différents. Le nombre idéal d'arbres, qui peut aller jusqu'à plusieurs centaines voire plus, est un paramètre important : il est très variable et dépend du problème. Concrètement, chaque arbre de la forêt aléatoire est entraîné sur un sous ensemble aléatoire de données selon le principe du bagging, avec un sous ensemble aléatoire de features (caractéristiques variables des données) selon le principe des « projections aléatoires ». Les prédictions sont ensuite moyennées lorsque les données sont quantitatives ou utilisés pour un vote pour des données qualitatives, dans le cas des arbres de classification.



#### 4. Linear Discriminant Analysis :

L'analyse discriminante linéaire ou l'analyse discriminante normale ou l'analyse fonctionnelle discriminante est une technique de réduction de dimensionnalité qui est couramment utilisée pour les problèmes de classification supervisée. Elle est utilisée pour modéliser les différences dans les groupes, c'est-à-dire séparer deux classes ou plus. Elle est utilisée pour projeter les entités d'un espace de dimension supérieure dans un espace de dimension inférieure.

Par exemple, nous avons deux classes et nous devons les séparer efficacement. Les classes peuvent avoir plusieurs fonctionnalités. L'utilisation d'une seule fonctionnalité pour les classer peut entraîner un certain chevauchement (overlapping), comme illustré dans la figure ci-dessous. Ainsi, nous continuerons d'augmenter le nombre de fonctionnalités pour une classification appropriée.



En général les étapes pour performer une analyse discriminante sont les suivants :

1. Calculez les vecteurs moyens d-dimensionnels pour les différentes classes de l'ensemble de données.
2. Calculez les matrices de dispersion (matrice de dispersion intra-classe et intra-classe).
3. les vecteurs propres ( $e_1, e_2, \dots, e_d$ ) et les valeurs propres correspondantes ( $\lambda_1, \lambda_2, \dots, \lambda_d$ ) pour les matrices de diffusion.



4. Triez les vecteurs propres en diminuant les valeurs propres et choisissez  $k$  vecteurs propres avec les plus grandes valeurs propres pour former une matrice dimensionnelle  $d \times k$   $WW$  (où chaque colonne représente un vecteur propre).
5. Utilisez cette matrice de vecteur propre  $d \times k$  pour transformer les échantillons dans le nouveau sous-espace. Cela peut être résumé par la multiplication matricielle:  
 $YY = XX \times WW$  (où  $XX$  est une matrice  $n \times d$  dimensionnelle représentant les  $n$  échantillons, et  $yy$  sont les échantillons transformés  $n \times k$  dimensions dans le nouveau sous-espace).

## 5. StackingClassifier :

La forme de stacking la plus simple peut être décrite comme une technique d'apprentissage d'ensemble où les prédictions de classificateurs multiples (appelés classificateurs de niveau 0) sont utilisées comme nouvelles fonctionnalités pour former un méta-classificateur. Le méta-classificateur peut être n'importe quel classificateur (appelé classificateur de niveau 1). La figure ci-dessous montre comment trois classificateurs différents sont formés. Leurs prédictions sont stackés et sont utilisées comme des fonctionnalités pour former le méta-classificateur qui fait la prédiction finale.

## Concept Diagram of Stacking

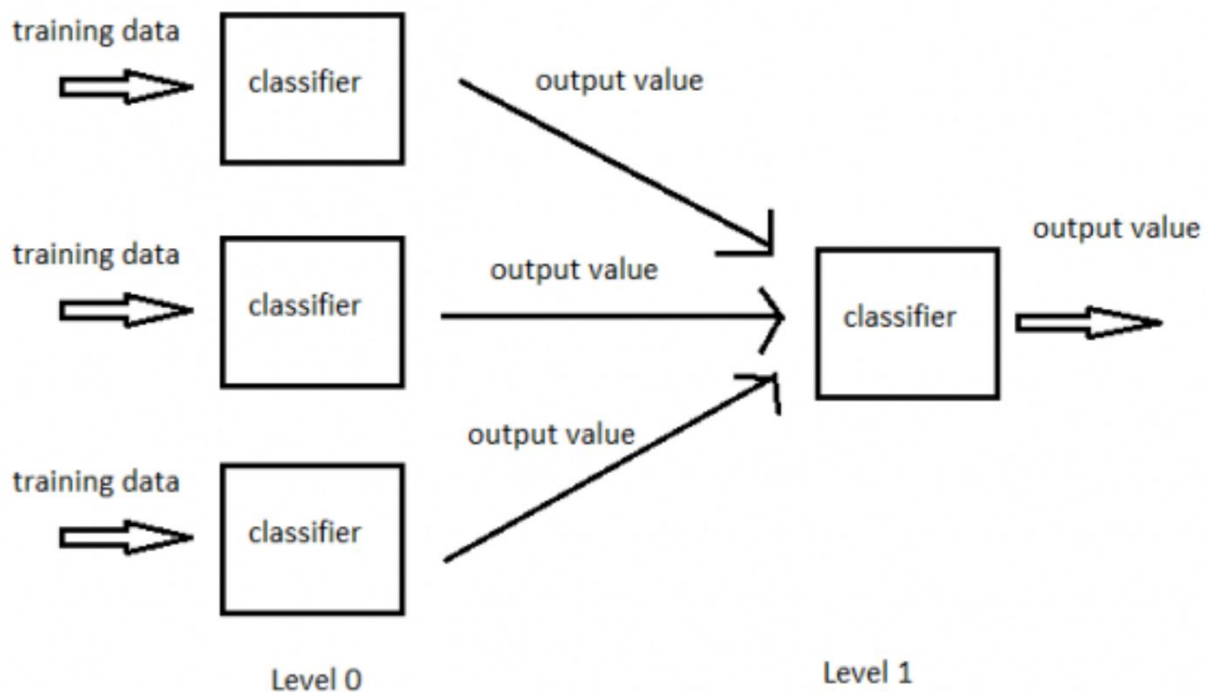
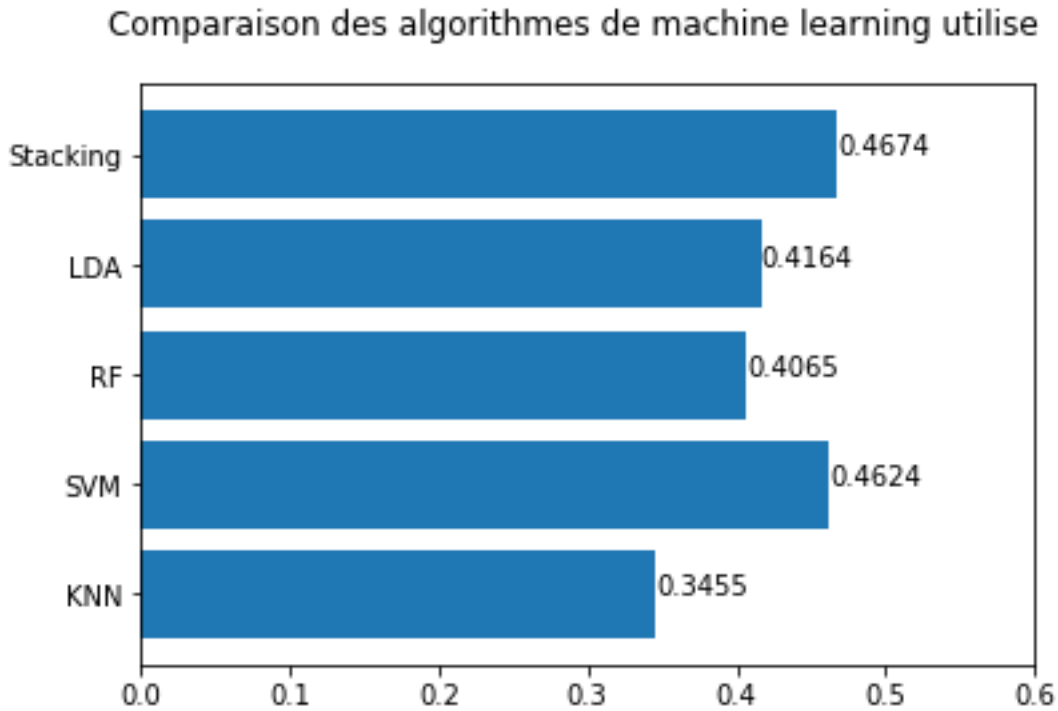


Figure 6 – Le concept du Stacking

## VI. Résultats :

Après l'apprentissage, on a obtenu cinq résultats différents, la figure ci-dessous montre la précision des algorithmes de Machine Learning utilisées :



**Figure 7** – Diagramme de précision des algorithmes de Machine Learning

Comme on peut remarquer, en première place viens l’algorithme **SVM**, ensuite **LDA** et **RF** et enfin **KNN**. Ce qui montre l’efficacité de **SVM** pour l’apprentissage des données de CIFAR-10 même si pour un taux de **46.24%** de précision qu’on a essayé d’augmenter en utilisant le Stacking Classifier qui combine tout les algorithmes précédant avec l’implémentation de la régression logistique (C’est une technique prédictive qui vise à construire un modèle permettant de prédire /expliquer les valeurs prises par une variable cible qualitative à partir d’un ensemble de variables explicatives quantitatives ou qualitatives) comme un estimateur final pour conclure a une précision de **46.74%**.

## Phase 2 : Apprentissage avec DL

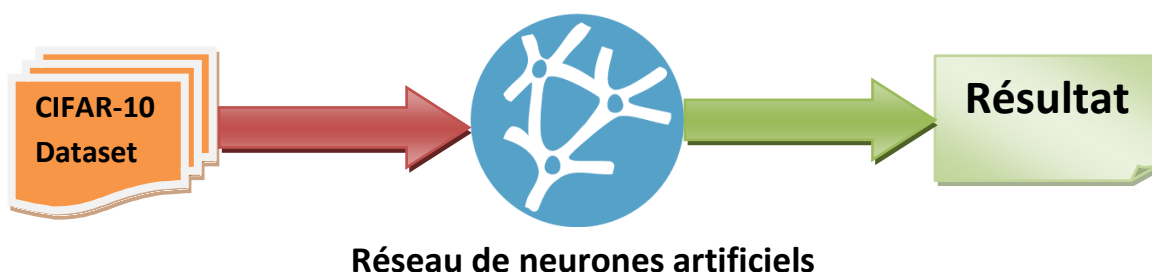
### I. Présentation du Deep Learning :

Le Deep Learning ou apprentissage profond est un type d'intelligence artificielle dérivé du Machine Learning (apprentissage automatique) où la machine est capable d'apprendre par elle-même, contrairement à la programmation où elle se contente d'exécuter à la lettre des règles prédéterminées.

Le deep Learning s'appuie sur un réseau de neurones artificiels s'inspirant du cerveau humain. Ce réseau est composé de dizaines voire de centaines de « couches » de neurones, chacune recevant et interprétant les informations de la couche précédente. Le système apprendra par exemple à reconnaître les lettres avant de s'attaquer aux mots dans un texte, ou détermine s'il y a un visage sur une photo avant de découvrir de quelle personne il s'agit.

### II. Présentation de l'architecture de la phase 2:

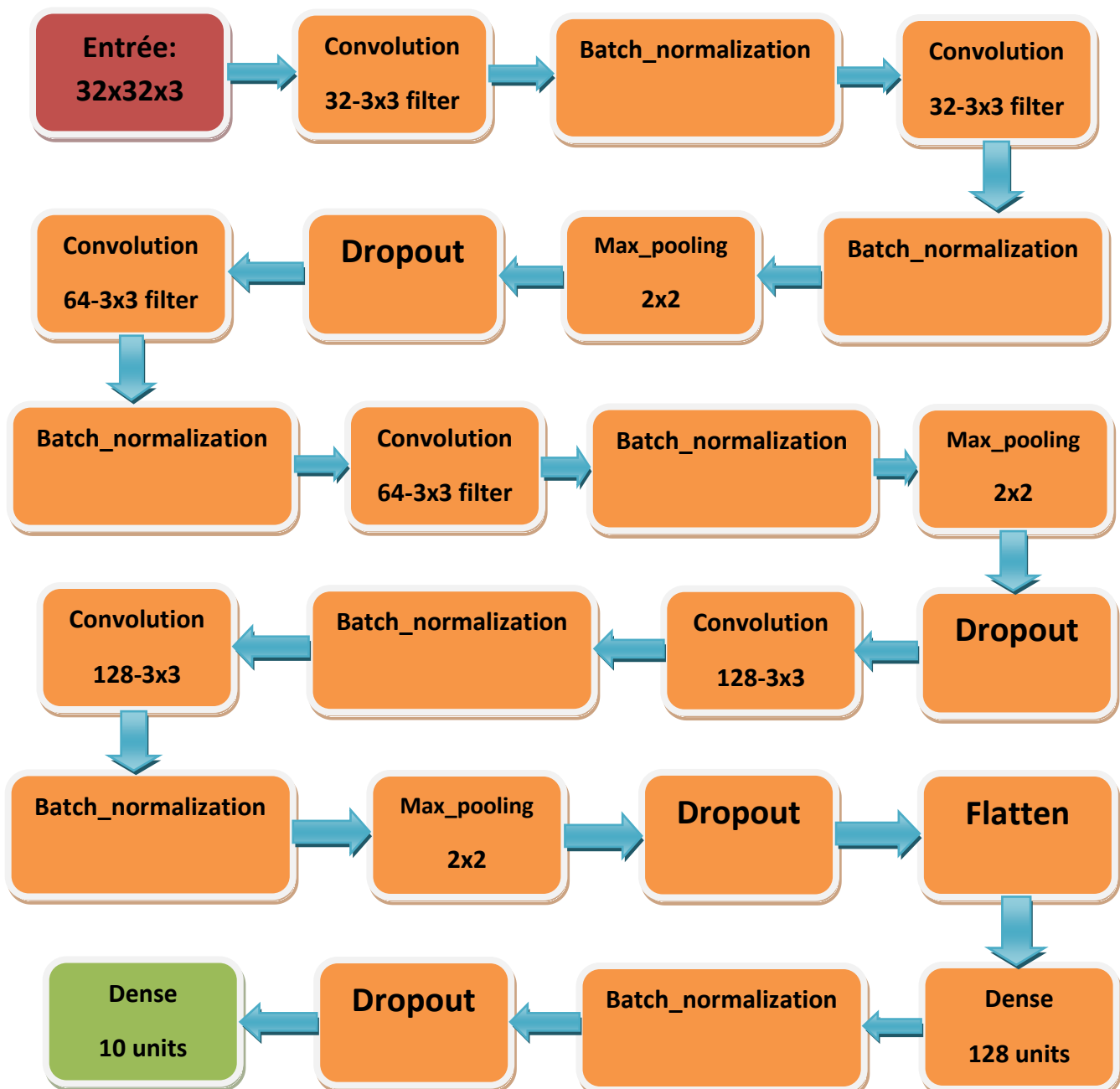
#### 1. Architecture générale :



## 2. Architecture du modèle :

Les images sont de taille 32 x 32. On convertit la matrice d'image en un tableau, on varie les valeurs du tableau entre 0 et 1 et le redimensionne pour qu'il soit de taille 32 x 32 x 3 pour alimenter l'entrée du réseau.

Voila l'architecture du réseau de neurones :



### 3. Paramétrage du modèle :

Pour réaliser le paramétrage du modèle, on a utilisé l'architecture VGG qui réfère généralement à un réseau convolutionnel profond pour la reconnaissance d'objets développé et formé par le célèbre Visual Geometry Group (VGG) d'Oxford, qui a obtenu de très bonnes performances sur l'ensemble de données ImageNet.

On a initialement commencé par un modèle VGG d'un block qui consiste à réaliser deux couches de convolution d'une profondeur de **32**, chaque couche utilisera la fonction d'activation **ReLU** et l'initialisation du poids **He**, suivi d'une couche de regroupement maximale (max\_pooling) pour avoir un résultat de **65%** comme précision sur **300** époques.

Ensuite on a réalisé un modèle VGG de deux blocks qui englobe les couches du modèle VGG d'un block à l'addition de deux couches de convolution d'une profondeur de **64** suivi d'une couche de regroupement maximale (max\_pooling) pour avoir un résultat de **69%** comme précision sur **300** époques.

Finalement, on passe à un modèle VGG de trois blocks qui englobe lui-même les couches du modèle VGG de deux blocks à l'addition de deux couches de convolution d'une profondeur de **128** suivi d'une couche de regroupement maximale (max\_pooling) pour avoir un résultat de **73%** comme précision sur **300** époques. Celui là sera notre modèle de base pour arriver à une précision satisfaisante.

## Les solutions appliquées au modèle de base sont :

### Augmentation de données :

L'augmentation des données consiste à faire des copies des exemples dans l'ensemble de données de formation avec de petites modifications aléatoires.

Cela a un effet de régularisation car il étend à la fois l'ensemble de données d'apprentissage et permet au modèle d'apprendre les mêmes caractéristiques générales, bien que d'une manière plus généralisée.

Il existe de nombreux types d'augmentation des données qui pourraient être appliqués. Étant donné que l'ensemble de données est composé de petites photos d'objets, on ne veut pas utiliser d'augmentation qui déforme trop les images, afin que les fonctionnalités utiles des images puissent être préservées et utilisées.

Les types d'augmentations aléatoires qui pourraient être utiles incluent un retournement horizontal, des décalages mineurs de l'image et peut-être un petit zoom ou un recadrage de l'image.

On va appliquer une augmentation simple sur l'image de base, en particulier les retournements horizontaux et les décalages de 10% de la hauteur et de la largeur de l'image.

```
#Générateur de données  
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
```

## Normalisation de données :

Pour augmenter la stabilité d'un réseau neuronal et peut-être accélérer le processus d'apprentissage, la normalisation par batch ou lot normalise la sortie d'une couche d'activation précédente en soustrayant la moyenne des batchs et en la divisant par l'écart-type des batchs.

Cependant, après ce décalage des sorties d'activation par certains paramètres initialisés au hasard, les poids dans la couche suivante ne sont plus optimaux.

```
model.add(BatchNormalization())
```

## Dropout :

Le dropout est une technique simple qui supprimera au hasard les nœuds du réseau. Il a un effet de régularisation car les nœuds restants doivent s'adapter pour prendre le relais des nœuds supprimés.

Dropout peut être ajouté au modèle en ajoutant de nouvelles couches Dropout, où la quantité de nœuds supprimés est spécifiée comme paramètre. Il existe de nombreux modèles pour ajouter Dropout à un modèle, en termes d'emplacement dans le modèle pour ajouter les couches et de la quantité de dropout à utiliser.



Dans ce cas, on ajoutera des couches de dropout après chaque couche de regroupement maximale (max\_pooling) et après la couche entièrement connectée, et utiliseront un taux de dropout de **20%, 30%, 40%, 50%** pour remédier à l'accélération d'apprentissage qu'offre la normalisation de batch.

```
model.add(Dropout(0.2))  
model.add(Dropout(0.3))  
model.add(Dropout(0.4))  
model.add(Dropout(0.5))
```

### Optimiseur :

Notre model vas être optimisé par « **Adam** » qui est un algorithme d'optimisation qui peut être utilisé à la place de la procédure classique de descente de gradient stochastique pour mettre à jour les poids de réseau itératifs basés sur les données d'entraînement. C'est un algorithme populaire dans le domaine de Deep Learning car il permet d'obtenir rapidement de bons résultats. Il est relativement facile à configurer là où les paramètres de configuration par défaut fonctionnent bien sur la plupart des problèmes.

Le modèle optimisera la fonction de perte d'entropie croisée catégorielle requise pour la classification multi classe et surveillera la précision de la classification.

```
#Compilation du modele  
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

## 4. Résumer du modèle :

Layer (type)	Output Shape	Param #	
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896	batch_normalization_7 (Batch Normalization) (None, 128) 512
batch_normalization_1 (Batch Normalization)	(None, 32, 32, 32)	128	dropout_4 (Dropout) (None, 128) 0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	9248	dense_2 (Dense) (None, 10) 1290
batch_normalization_2 (Batch Normalization)	(None, 32, 32, 32)	128	=====
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 32)	0	Total params: 552,874
dropout_1 (Dropout)	(None, 16, 16, 32)	0	Trainable params: 551,722
conv2d_3 (Conv2D)	(None, 16, 16, 64)	18496	Non-trainable params: 1,152
batch_normalization_3 (Batch Normalization)	(None, 16, 16, 64)	256	
conv2d_4 (Conv2D)	(None, 16, 16, 64)	36928	
batch_normalization_4 (Batch Normalization)	(None, 16, 16, 64)	256	
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 64)	0	
dropout_2 (Dropout)	(None, 8, 8, 64)	0	
conv2d_5 (Conv2D)	(None, 8, 8, 128)	73856	
batch_normalization_5 (Batch Normalization)	(None, 8, 8, 128)	512	
conv2d_6 (Conv2D)	(None, 8, 8, 128)	147584	
batch_normalization_6 (Batch Normalization)	(None, 8, 8, 128)	512	
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 128)	0	
dropout_3 (Dropout)	(None, 4, 4, 128)	0	
flatten_1 (Flatten)	(None, 2048)	0	
dense_1 (Dense)	(None, 128)	262272	

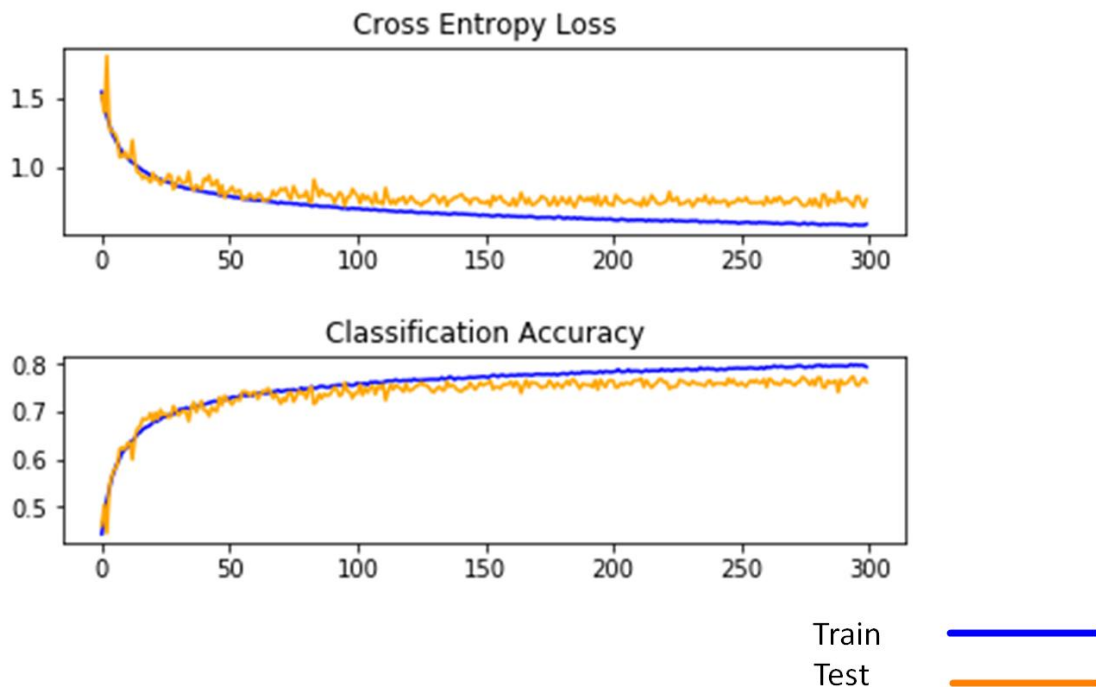
## III. Etapes d'exécution de la phase 2 :

- ✓ Importation des images CIFAR-10.
- ✓ Initialisation des paramètres du modèle des réseaux de neurones.
- ✓ Lancement de l'apprentissage avec 300 époques.

- ✓ Evaluation du résultat.
- ✓ Sauvegarde du modèle dans un fichier .h5

## IV. Résultat :

Comme résultat, on obtient les courbes d'apprentissage suivantes tout au long des 300 époques :



**Figure 8** – Courbes de précision et de loss du modèle de Deep Learning

On remarque que l'apprentissage évolue de façon rapide jusqu'à l'époque 50 pour prendre un élan normal à la limite de **78%** de précision et **0.34** de loss dans la 300ème époque.

# Implémentation des phases

## I. Obtention des images test :

On se dirige vers internet pour obtenir des images au hasard que leurs étiquettes existent dans les classes de CIFAR-10.

Par exemple :



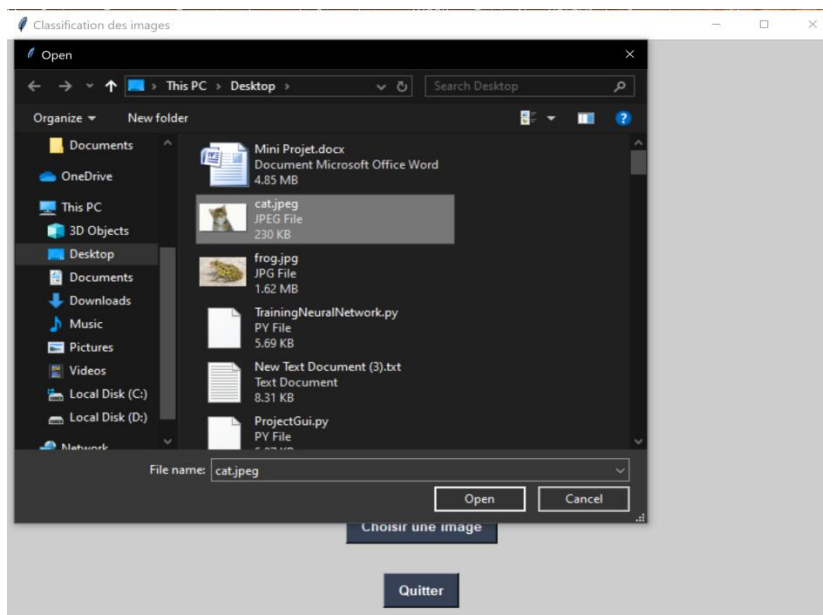
## II. Prédiction de l'étiquette de chaque image suivant les deux modèles :

### 1. Importation de l'image :

On lance le code de l'interface pour afficher la fenêtre suivante :



On choisit l'image à importer ensuite :



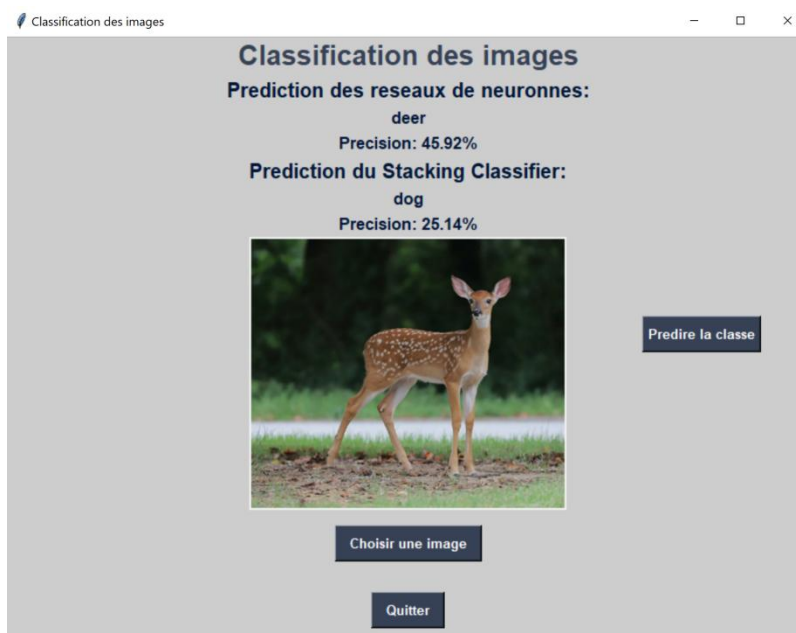
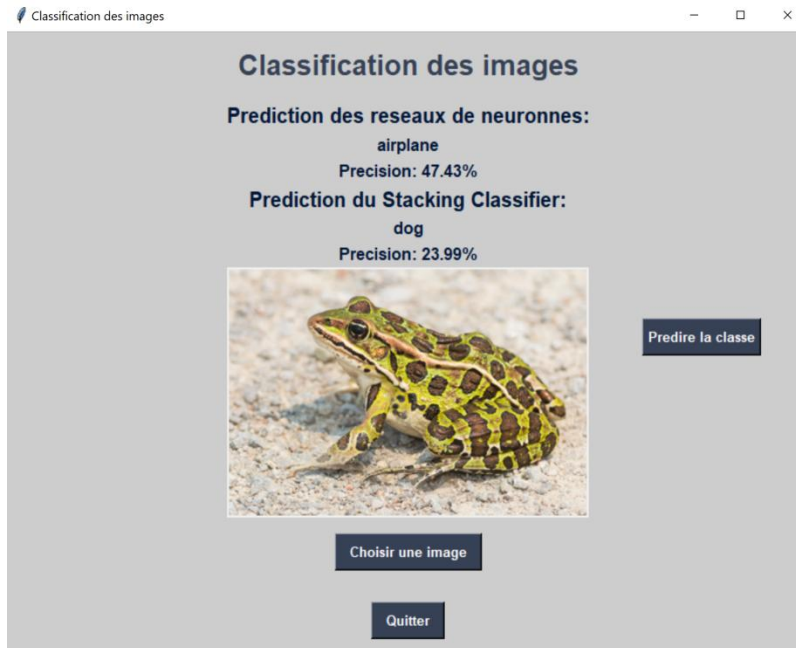
On arrive à la fenêtre suivante :



## 2. Prédiction des classes :

On refait les étapes précédentes pour chaque image en vis de générer les résultats suivants :





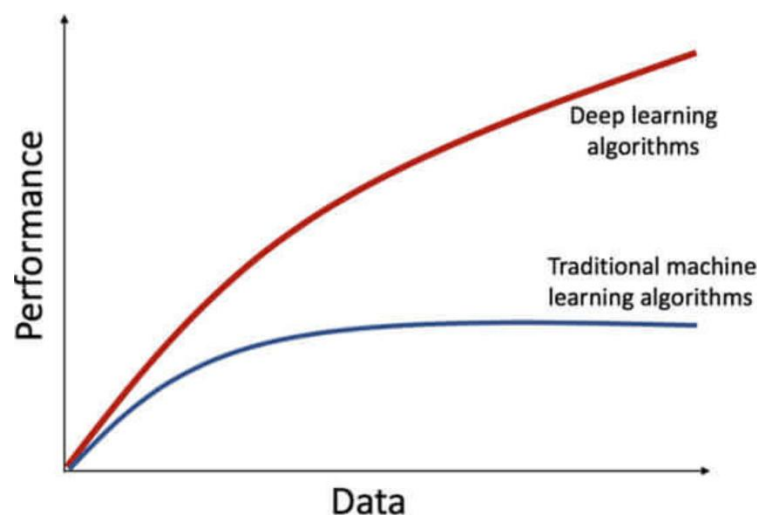
### 3. Résultats :

Depuis les prédictions faites, on remarque que le modèle du Deep Learning a été le plus correct avec une prédiction de **2/3** tant dis que le modèle du Machine Learning à valider **0/3** prédictions.

# Conclusion

D'après les résultats qu'on a acquis tout au long de ce projet, il est remarquable que pour le dataset CIFAR-10, le modèle de Deep Learning a prouvé une efficacité face aux algorithmes de Machine Learning. Cela revient à plusieurs facteurs :

- ✓ Le Deep Learning prend beaucoup plus de temps que le Machine Learning en apprentissage.
- ✓ Le Deep Learning requiert une GPU pour apprendre efficacement alors que le Machine Learning peut se contenter du CPU seulement.
- ✓ Le Deep Learning peut être configuré avec plusieurs différentes approches au contraire du Machine Learning qui est limitée en matière de paramètres.
- ✓ Le Deep Learning requiert un large volume de données pour fournir une haute précision bien meilleure que le Machine Learning qui peut se contenter de peu de données.





# Références

[Image Classification using Python and Scikit-learn – Gogul Ilango](#)

[Convolutional Neural Networks in Python - DataCamp](#)

[How to Develop a CNN From Scratch for CIFAR-10 Photo Classification](#)

[Stacking Classifiers for Higher Predictive Performance](#)

[How to Visualize a Deep Learning Neural Network Model in Keras](#)

[Stacking Ensemble Machine Learning With Python](#)

[scikit-learn: Save and Restore Models](#)

[Classification multi-classe et stacking — papierstat](#)

<https://www.igi-global.com/dictionary/color-histogram/4519>

<https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0212110>

<https://www.learnopencv.com/shape-matching-using-hu-moments-c-python/>

<https://dataanalyticspost.com/Lexique/random-forest/#:~:text=Random%20Forest,seul%2C%20am%C3%A9liorant%20ainsi%20leurs%20performances.>

<https://dataanalyticspost.com/Lexique/svm/>

<https://dataanalyticspost.com/Lexique/k-nearest-neighbours/>

<https://www.lebigdata.fr/machine-learning-et-big-data>