École Centrale de Lyon

MOD 3.2 - Deep Learning IA
Option Informatique
Rapport

TD1 – Kppv et réseaux de neurones pour la classification d'images

Élèves :
Thibaud MIQUEL

Enseignant :
Thomas DUBOUDIN

Table des matières

Table des figures

1 Présentation du TD

1.1 Objectif du TD

L'objectif de ce TD est d'écrire en langage Python un programme complet de classification d'images. Deux modèles de classification seront successivement développés et expérimentés : les k-plus-proches-voisins et les réseaux de neurones. Le module numpy sera utilisé pour la manipulation des matrices et le module scikit-image 1 pour la manipulation des images.

1.2 Présentation du data set Cifar 10

Le jeu de données CIFAR-10 est composé de 60 000 images couleur 32x32 réparties en 10 classes, avec 6 000 images par classe. Il y a 50000 images d'entraînement et 10000 images de test.

Le jeu de données est divisé en cinq lots d'entraînement et un lot de test, chacun contenant 10 000 images. Le lot de test contient exactement 1000 images choisies au hasard dans chaque classe. Les lots d'entraînement contiennent les autres images dans un ordre aléatoire, mais certains lots d'entraînement peuvent contenir plus d'images d'une classe que d'une autre. Entre eux, les lots d'entraînement contiennent exactement 5000 images de chaque classe.

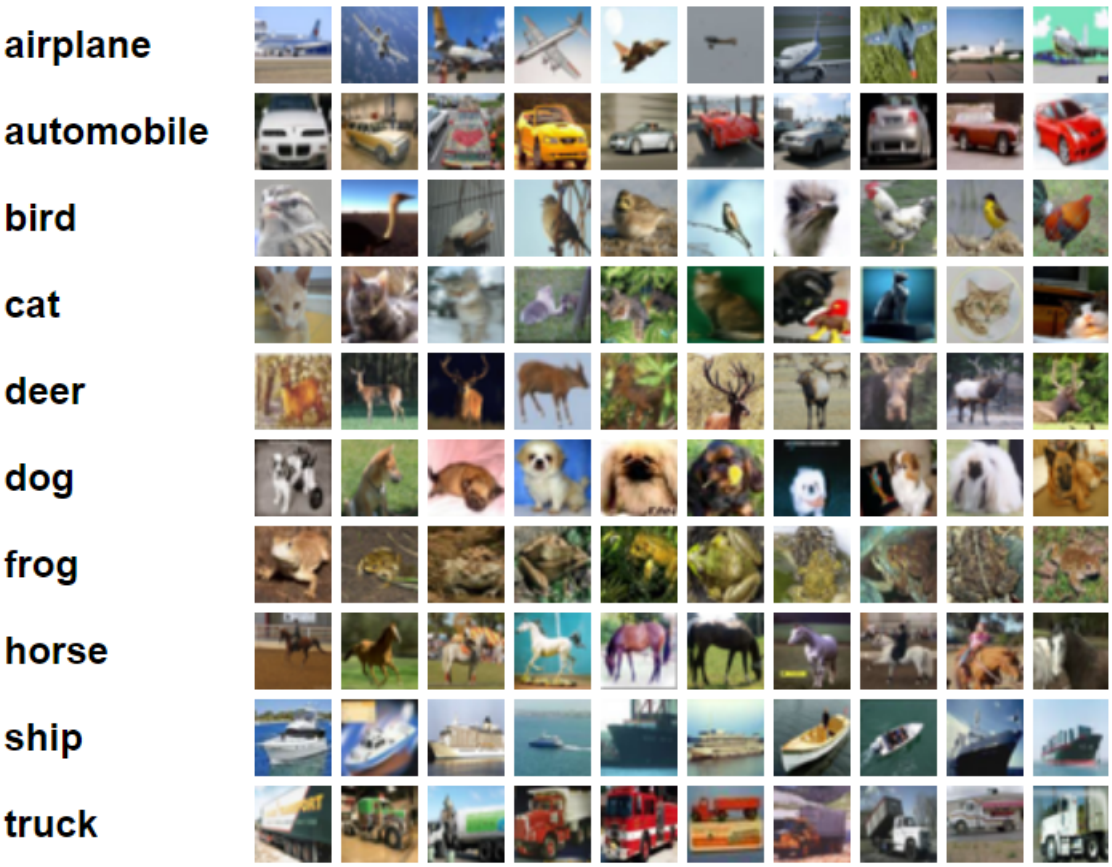

FIGURE 1 – Voici les classes de l'ensemble de données, ainsi que 10 images aléatoires de chaque classe

Je dois prévenir que du à une capacité de calcul très faible tous mes tests et toutes mes exécutions des différents algorithmes ne sont fait qu'avec seulement 10000 images (correspondant au data batch 1). On peut déjà supposer que les résultats pourraient être meilleurs si on prenait les 60000 images (notamment pour la solution de classification à base de réseau de neurones)

2 Système de classification à base de l'algorithme des kppv

2.1 Présentation des fonctions

- **unpickle :** Cette fonction prend en argument le nom du fichier à ouvrir et en permet la lecture. Elle renvoie un dictionnaire contenant les données du fichier ouvert.
- **lecture cifar :** Cette fonction prend en argument le nom du fichier et le chemin vers ce fichier pour y extraire les données. Elle renvoie 2 tableaux numpy, un avec les données (images) et un avec les labels de chaque image.
- **visualisation classe :** Cette fonction prend en argument les données ainsi que leur label, le nombre d'échantillon par classe à montrer et le nom des classes. Elle permet de visualiser les images indépendamment du type de descripteur (RGB, HOG, LBP) en triant par classe.
- **normalisation :** Cette fonction prend en argument les données et renvoie la même matrice avec toutes les valeurs comprisent entre 0 et 1 en divisant les valeurs initiales par 255.
- **decoupage donnees :** Cette fonction prend en argument les données ainsi que leur label et un coefficient compris entre 0 et 1 (par défaut fixé à 0.8) qui déterminera la proportion du set initial de donnée pour créer le set d'entraînement. Elle renvoie 4 matrices qui sont les données d'entraînement ainsi que de test avec leur label associé.
- **kppv distances :** Cette fonction prend en argument le set de donnée d'entraînement et le set de test pour calculer la distance L2 entre chaque donnée du set d'entrainement et de test. Cette fonction renvoie une matrice de taille DxN, correspondant à la distance entre le set d'entrainement et le set de test, où N est le nombre de donnée d'entrainement et D est le nombre de donnée de test. Chaque valeur est déterminée grâce aux calculs suivant :

$$c_{ij} = \sum_{k=0}^{N} a_{kj}^2 + \sum_{k=0}^{N} b_{ki}^2 - \sum_{k=0}^{N} a_{kj} b_{ki} \tag{1}$$

  Ici c_{ij} correspond au terme de la matrice renvoyée par la fonction. a_{kj} correspond au kème pixel de la jème image d'entrainement. a_{ki} correspond au kème pixel de la ième image de test. N correspond au nombre de pixel sur chaque image (3072 ici).
- **kppv predict :** Cette fonction prend en argument les labels d'entrainements, la matrice de distance entre données d'entraînement et de test et un nombre de k plus proche voisin. Elle renvoie les prédictions pour le set de test. En fonction du nombre de k plus proche voisin la fonction va déterminer pour chaque données de test les k distance les faibles et va donner comme prédiction le label avec le plus d'occurence parmis ces k plus proches voisins.
- **evaluation classifieur :** Cette fonction prend en argument les prédictions réalisées et les vrais labels et renvoie un nombre réel entre O et 1 traduisant la précision de l'algorithme. Il calcul le quotient entre le nombre de prédictions corrects et le nombre de données de test.
- **plot confusion matrix :** Cette fonction prend en argument les prédictions réalisées, les vrais labels ainsi que le nom des classes et renvoie la matrice de confusion permettant de valider le modèle.
- **cross validation :** Cette fonction prend en argument les données, leur label, un nombre naturel traduisant le nombre de sous-ensemble de la validation croisée ainsi qu'une liste de paramètre k à tester. Elle renvoie un dictionnaire avec la liste des précision obtenue pur chaque paramètre k.
- **hog prep :** Cette fonction prend en argument les données et les transforme afin de renvoyer ces même données en descripteurs HOG.

— **rgb to gray :** Cette fonction prend en argument les données et converti le codage RGB en nuance gris (une image aura 1024 pixel au lieu de 3072).
— **lbp prod :** Cette fonction prend en argument les données, un nombre de point, un rayon et une méthode. Elle renvoie 2 tableau numpy, un comprenant les histogrammes des descriteurs LBP et l'autre les images décritent avec les descripteurs LBP.

2.2 Résultats avec algorithme de base

L'algorithme avec les images en RGB utilisées telles qu'elles en y appliquant simplement une normalisation du nombre de pixel renvoie en moyenne une précision de l'ordre de 30%. Cet algorithme est donc plus performant qu'un pur choix aléatoire (qui serait d'une précision d'environ 10%).

On peut constater sa performance sur la matrice de confusion :

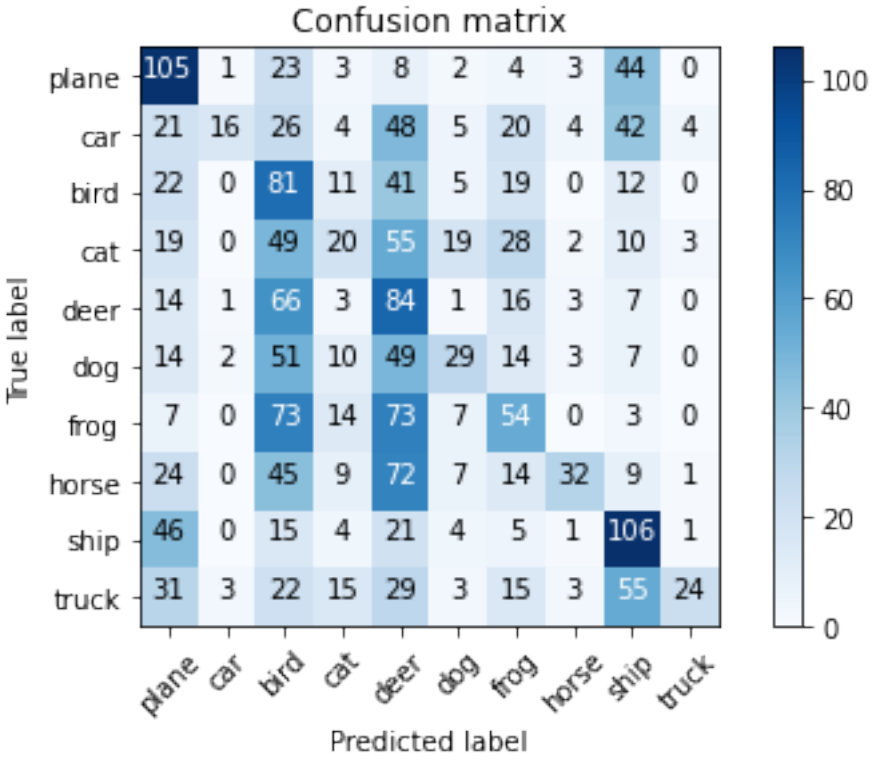

FIGURE 2 – Matrice de confusion avec l'algorithme knn sur image de base

On peut constater que l'algorithme a tendance à confondre les animaux entre eux (pour 205 images de chats il y a 20 bonnes prédictions, 153 d'autres animaux notamment 49 pour l'oiseau et seulement 32 prédictions de moyen de transport). A l'instar les moyens de transport ont aussi tendance à être confondu entre eux : pour ship (bateau) il y 203 images de camions, 106 sont correctement prédites et 47 sont des images d'autres transports (notamment l'avion avec 46 prédictions) contre donc 51 prédictions d'animaux (sachant qu'il y a 6 classes d'animaux contre 4 classes de transport).

En faisant varier k en gardant les même set de train et test on observe que la précision décroit plus k est grand et donc plus on considère de voisin pour la valeur à prédire.

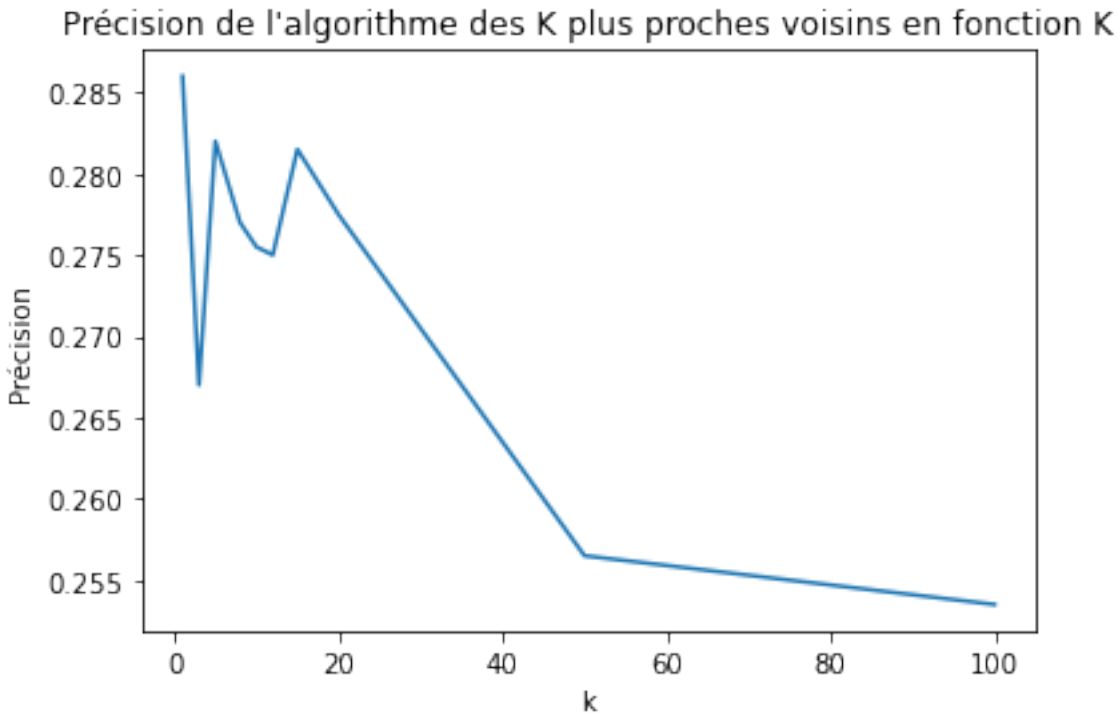

FIGURE 3 – Evolution de la précision en fonction de k sur le même set d'entrainement

2.3 Résultats avec la validation croisée

La validation croisée nous permet de déterminer de manière appropriée le paramètre optimal k en divisant le set d'entraînement en 5 sous ensembles sur lequel on entrainera l'algorithme sur 4 sous ensemble et on le testera sur le 5eme en répétant ce processus 5 fois pour que tous les sous ensembles soient au moins 1 fois le set de test. On peut observer le résultat de cette validation croisée sur le graphique suivant ou la courbe représente la moyenne des précision obtenue pour un k donnée et les points représentent la précision obtenu pour un k donnée sur chaque set de test.

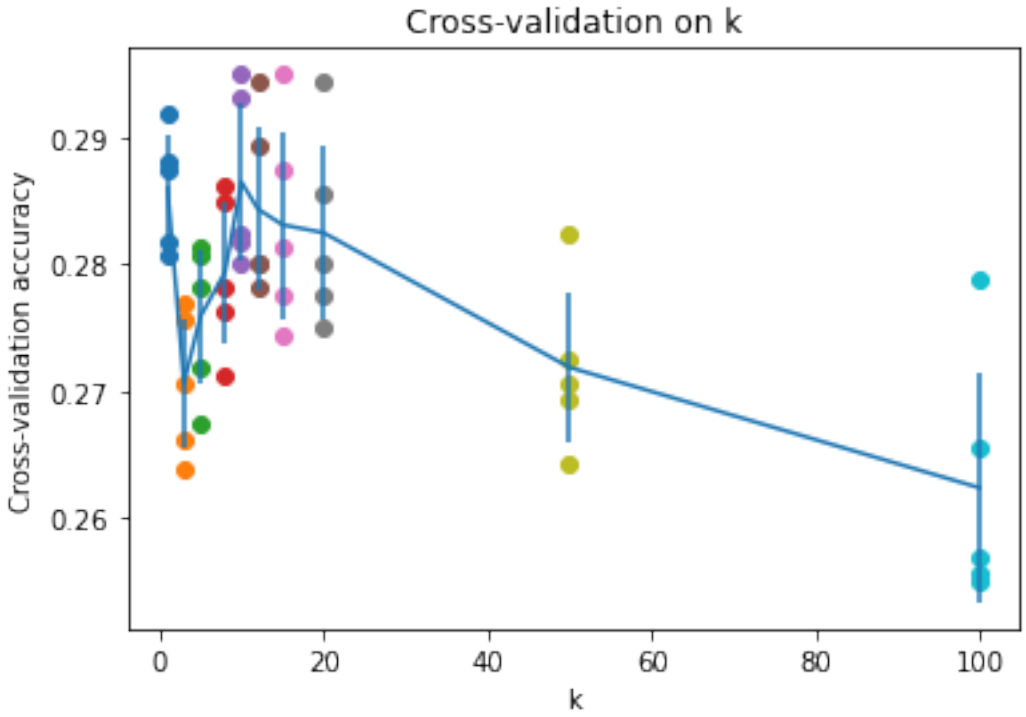

FIGURE 4 – Graphique de la validation croisée sur k

On constate que l'hyperparamètre k optimal obtenu via la validation croisée est $k = 10$.

2.4 Résultats avec l'utilisation de descripteur HOG

Après avoir développer l'algorithme en utilisant les images RGB telles quelles on va ensuite essayer d'utiliser les images avec des descripteurs HOG : histogramme de gradient orienté. L'idée importante derrière le descripteur HOG est que l'apparence et la forme locale d'un objet dans une image peuvent être décrites par la distribution de l'intensité du gradient ou la direction des contours. Ceci peut être fait en divisant l'image en régions adjacentes de petite taille, appelées cellules, et en calculant pour chaque cellule l'histogramme des directions du gradient ou des orientations des contours pour les pixels à l'intérieur de cette cellule. La combinaison des histogrammes forme alors le descripteur HOG.

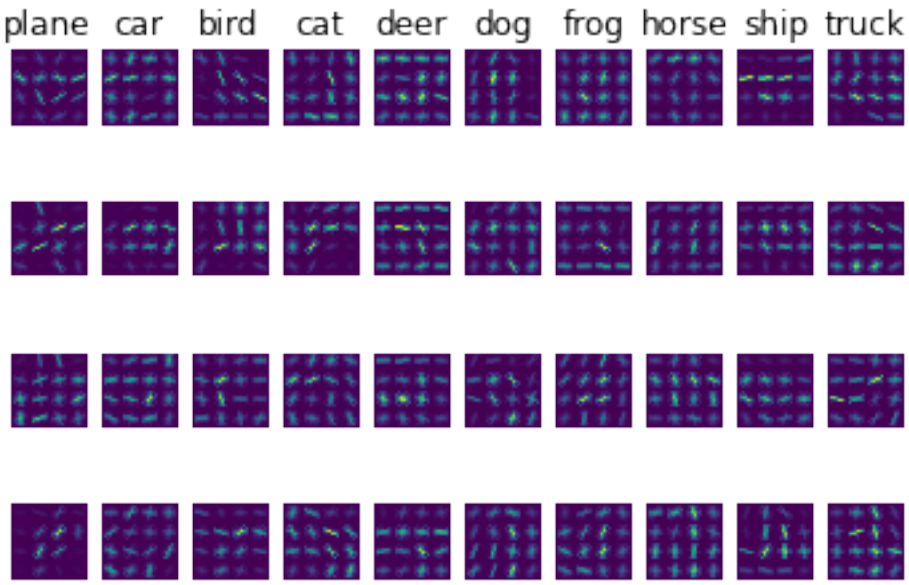

FIGURE 5 – Images triées par classe en descripteur HOG

Ce format de donnée permet d'améliorer la précision. En effet on obtient une précision variant entre 40% et 45% selon k :

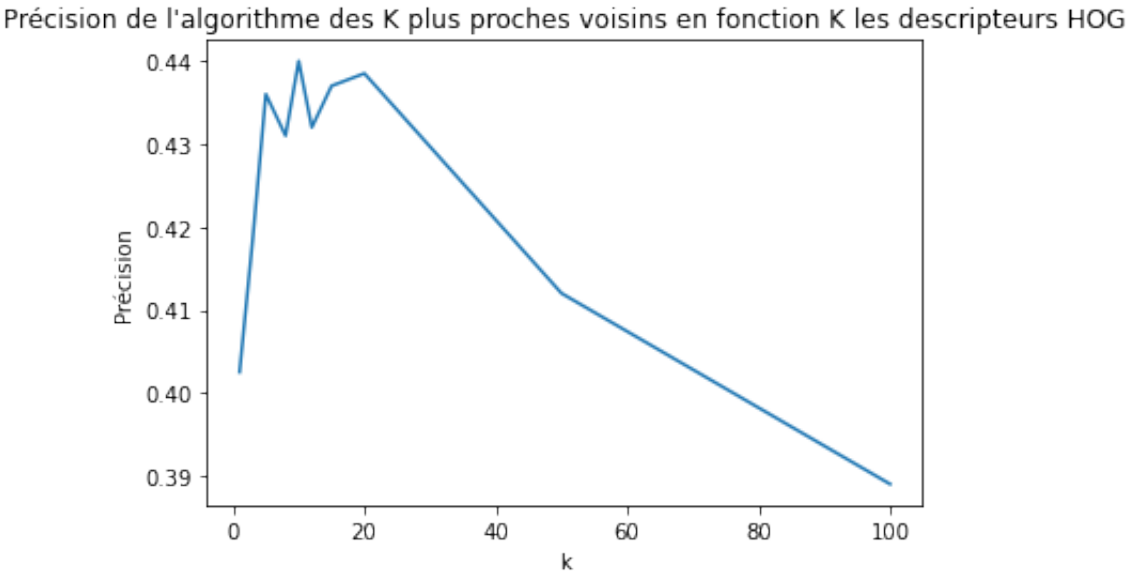

FIGURE 6 – Précision en fonction de k avec les données avec descripteurs HOG

Avec k égale à 15 on obtient la matrice de confusion suivante, qui est nettement mieux qu'avec les images en RGB :

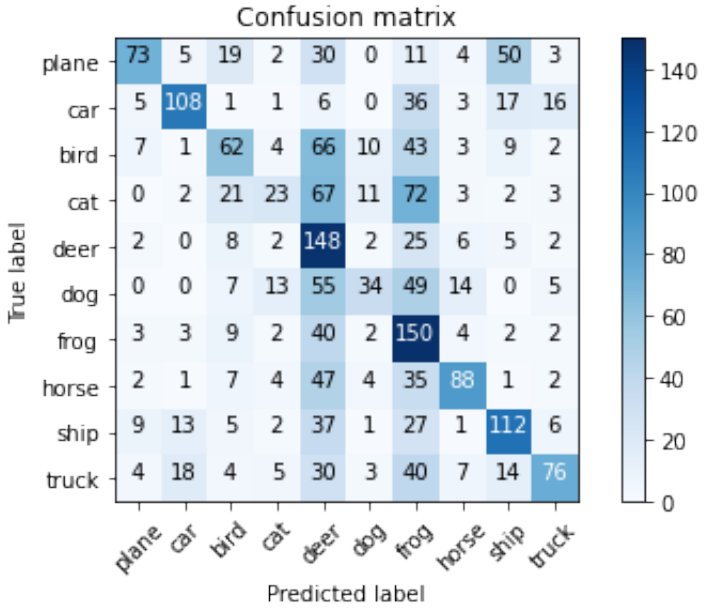

FIGURE 7 – Matrice de confusion pour knn avec descripteur HOG

On observe qu'avec les descripteurs HOG on arrive à augmenter la précision de 15% à 20% ces descripteurs sont donc plus adaptés.

2.5 Résultats avec l'utilisation de descripteur LBP

On va ensuite tester les descripteurs LBP(Local Binary Pattern). Le principe général est de comparer le niveau de luminance d'un pixel avec les niveaux de ses voisins. Cela rend donc compte d'une information relative à des motifs réguliers dans l'image, autrement dit une texture. Selon l'échelle du voisinage utilisé, certaines zones d'intérêt tels des coins ou des bords peuvent être détectés par ce descripteur. Ces descripteurs sont énormément utilisés pour la détection de visage étant donné que cela construit une texture distincte et régulière du reste de l'image.

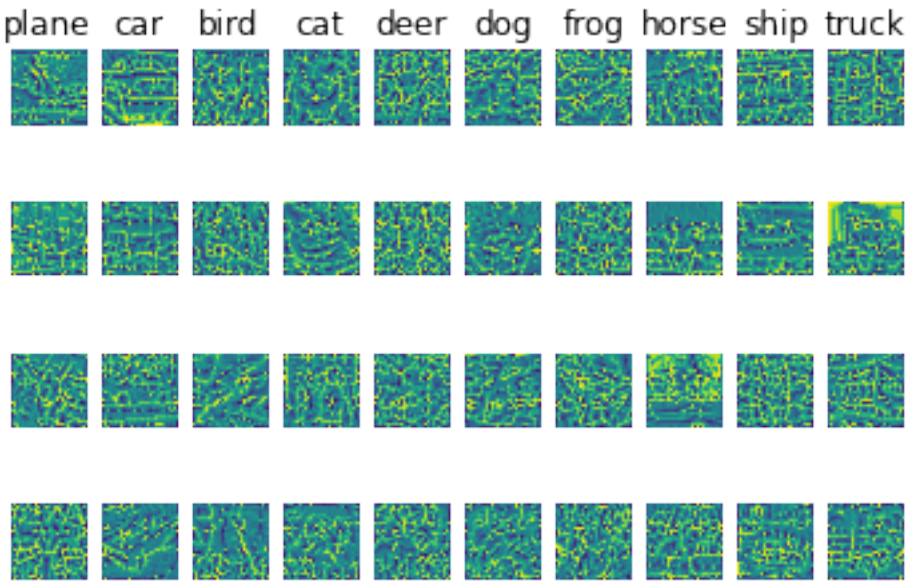

FIGURE 8 – Images triées par classe en descripteur LBP

Avec les descripteurs LBP on obtient une précision de l'ordre de 25%. Sans surprise ces descripteurs ne sont pas pas particulièrement adaptés à la reconnaissance d'image mais plutôt à la détection de motif sans forcément y donner un label. On voit que ces descripteurs amènent une irrégularité étant donnée la variation de la précision en fonction du nombre k, qui devrait décroître avec k.

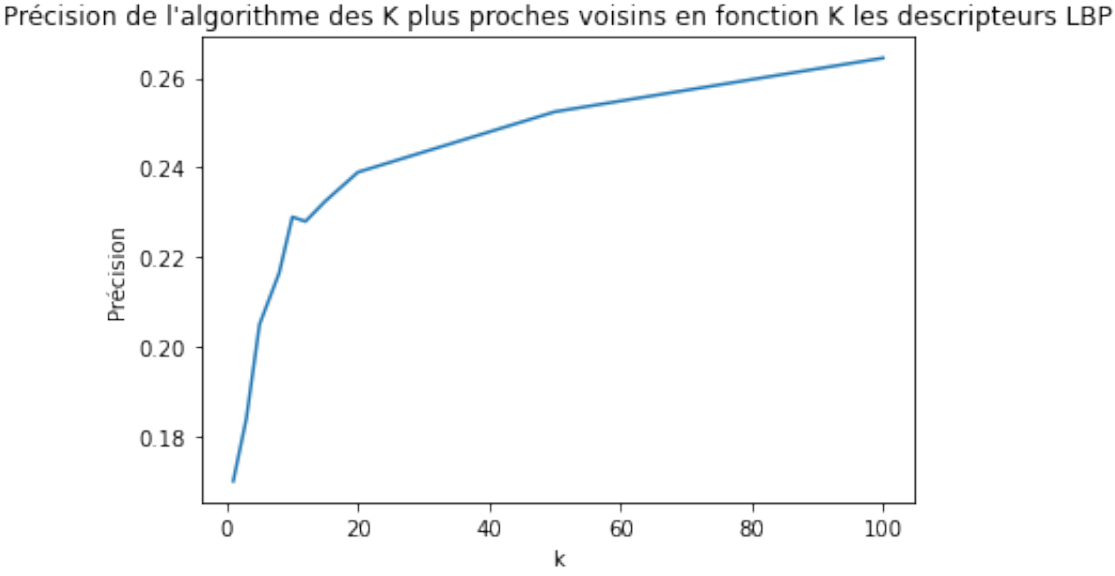

FIGURE 9 – Précision en fonction de k avec les données avec descripteurs LBP

De même on voit avec la matrice de confusion que ces descripteurs ne sont pas du tout adaptés :

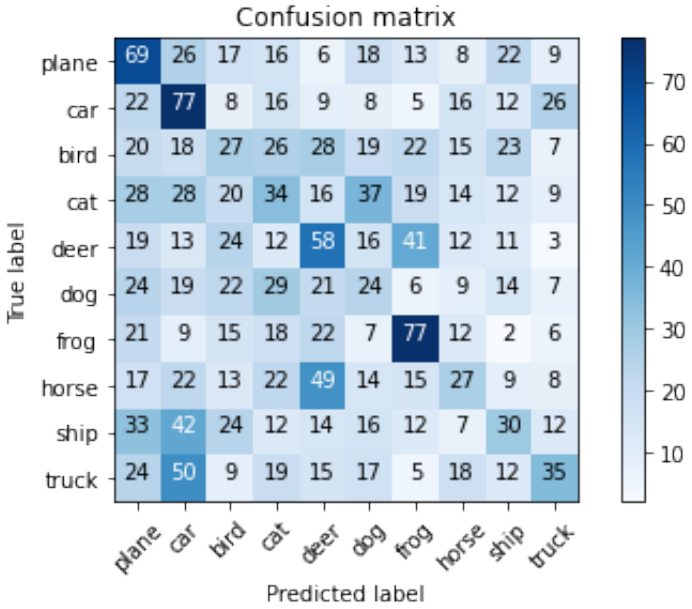

FIGURE 10 – Matrice de confusion pour knn avec descripteur LBP

3 Système de classification à base de réseaux de neurones

3.1 Présentation des fonctions

— **sigmoid :** Cette fonction prend en argument une matrice et retourne cette matrice à laquelle la fonction sigmoid a été appliqué à chaque élément.
— **sigmoid backward :** Cette fonction prend en argument une matrice et retourne cette matrice à laquelle la fonction sigmoid dérivée a été appliqué à chaque élément.
— **relu :** Cette fonction prend en argument une matrice et retourne cette matrice à laquelle la fonction leaky relu a été appliqué à chaque élément.
— **relu backward :** Cette fonction prend en argument une matrice et retourne cette matrice à laquelle la fonction leaky relu dérivée a été appliqué à chaque élément.
— **compute cross entropy cost :** Cette fonction prend en argument 2 matrices, les prédictions de l'algorithme et les vraies label et renvoie leur coût via la fonction de cross entropy.
— **compute MSE cost :** Cette fonction prend en argument 2 matrices, les prédictions de l'algorithme et les vraies label et renvoie leur coût via la fonction de mean squared error.
— **reshape label input :** Cette fonction prend en argument les labels des images afin de convertir une matrice de taille Nx1 où N est le nombre de donnée en matrice Nx10 où chaque ligne i comporte 9 0 et 1 fois 1 correspondant à l'index du label de l'image i. Cela permet lors de la comparaison entre les prédictions de l'algorithme et les vrais label d'avoir le même format pour calculer le coût.
— **batch normalization :** Cette fonction prend en argument une matrice de donnée et soustrait à chaque image l'image moyenne de cette matrice et divise chaque image par la variance de cette matrice. Cela permet une régularisation de l'algorithme afin de prévenir du sur-apprentissage.
— **pre processing :** Cette fonction prend en argument une matrice de donnée et vient normaliser chaque terme de la matrice en divisant par 255. Ensuite cette fonction vient soustraire l'image moyenne à chaque image.
— **iterate minibatches :** Cette fonction prend en argument une matrice de donnée, leur label ainsi qu'un nombre naturel traduisant la taille des batchs à produire. Elle renvoie une itération des sous ensembles de la matrice de donnée et leur label de la taille définie par le nombre naturel donné.
— **initialize parameters deep :** Cette fonction prend en argument une liste contenant le nombre de neurones sur chaque couche. Elle renvoie un dictionnaire contenant tous les paramètres de poids, de biais et de moments initialisés (poids initialisés avec la méthode de "Xavier", biais initialisés à 0, moments initialisés à 0 selon la méthode de Adam).
— **linear forward :** Cette fonction prend en argument une matrice de donnée (X), une matrice de poids (W) et une matrice de biais (b) et renvoie une matrice qui est le calcul correspondant au produit de la matrice d'entrée d'une couche de neurones auquel on vient ajouter les biais de cette couche :

$$Z = WX + b \tag{2}$$

— **linear activation forward :** Cette fonction prend en argument une matrice de donnée (X), une matrice de poids (W) et une matrice de biais (b) et une fonction d'activation spécifiée. Elle renvoie une matrice correspondant à l'application de la fonction d'activation choisie au résultat de "linear forward".
— **L model forward :** Cette fonction prend en argument une matrice de donnée, un dictionnaire contenant les paramètres du réseau de neurones à construire, la liste du nombre

de neurones sur chaque couche et la fonction d'activation à utiliser dans le réseau de neurones. Elle renvoie 2 listes de matrices : l'une contenant pour chaque couche de neurones la matrice résultant de la combinaison linéaire entre les poids, la matrice d'entrée de la couche de neurones et les biais de cette couche et l'autre contenant pour chaque couche de neurones la matrice résultant de l'application de la fonction d'activation choisie à la combinaison linéaire à l'issu de la couche de la de neurones. Cette fonction consiste donc à appliquer successivement à chaque couche de neurones "linear forward" ainsi que "linear activation forward" et à stocker les matrices obtenues.

— **linear backward :** Cette fonction prend en argument le gradient de la combinaison linéaires entre poids, données et biais à la couche N (dZ), les poids et les biais entre la couche N-1 et N (W,b) ainsi que la matrice résultant de l'application de la fonction d'activation à la couche N-1 (A_{prev}) . Elle retourne les gradients des poids, des biais entre la couche N-1 et N et du résultat de la fonction d'activation de la couche N-1 de la manière suivante, m=nombre de donnée d'entraînement :

$$dW = dZ A_{prev}/m \tag{3}$$

$$db = \sum dZ/m \tag{4}$$

$$dA_{prev} = W dZ \tag{5}$$

— **linear activation backward :** Cette fonction prend en argument la combinaison linéaires entre poids, données et biais à la couche N (Z), la gradient de la matrice résultant l'application de la fonction d'activation à Z (dA), les poids et les biais entre la couche N-1 et N (W,b) ainsi que la matrice résultant de l'application de la fonction d'activation à la couche N-1 (A_{prev}) et le type de fonction d'activation utilisée. Cette fonction applique la dérivée de la fonction d'activation à Z et dA pour obtenir dZ et applique ensuite "linear backward". Cette fonction renvoie donc les gradients de A_{prev} de W et de b.

— **L model backward :** Cette fonction prend en argument les vrais labels, les 2 listes résultants de "L model forward", le dictionnaire de paramètre, une fonction d'activation et une fonction de coût. Elle renvoie un dictionnnaire contenant les gradients des paramètres de poids, biais ainsi les gradients de la sortie des fonctions d'activation sur chaque couche. cette fonction consiste à appliquer successivement en remontant les couches du réseau de neurones la fonction "linear activation backward" et à en stocker les sorties.

— **update parameters :** Cette fonction prend en argument les paramètres W et b, les gradients de W et b sur toutes les couches de neurones, l'hyperparamètre d'apprentissage : learning rate, la liste du nombre de neurones par couche, un booléen (L2) pour utiliser ou non la régularisation L2. Grâce à la méthode Adam (almost) on met à jour chaque poid et biais en fonction de leur gradient calculé.

— **evaluation network classification :** Cette fonction prend en argument une matrice de données, les labels correspondant, un dictionnaire de paramètre de poids et biais, le type de fonction d'activation à utiliser et le nombre de neurones par couche. Elle renvoie un nombre entre 0 et 1 traduisant la proportion de bonne prédiction réalisée sur la matrice de donnée en comparant ces prédictions aux vrais labels.

— **fit network :** Cette fonction prend en argument un set d'entraînement et un set de test avec leur label correspondant, le nombre de neurones par couche pour le réseau à construire, le nombre d'itération qui permettront d'entraîner le réseau, un hyperparamètre : learning rate, la taille des sous ensembles d'entraînement, le type de fonction d'activation à utiliser et un booléen traduisant l'utilisation ou non d'un terme de régularisation (L2). Cette fonction permet de créer et d'entraîner un réseau de neurones, elle renvoie la liste des paramètres optimaux, la liste des coûts obtenus à chaque mise à jour des paramètres et les listes des précisions sur le set d'entraînement et de test à chaque

mise à jour des paramètres. Afin de stopper l'algorithme quand celui-ci n'évolue plus (ou trop peu) un early stop permet d'arrêter l'entraînement lorsque la différence en valeur absolue entre la moyenne des 10 dernières précisons et la dernière précision obtenue est inférieure à 0.5 sur le set d'entraînement. Cette fonction consiste à appliquer successivement pour chaque itération sur l'ensemble des données d'entraînement (découper ou non en sous ensemble) les fonctions suivantes :
  — 1/ "L model forward"
  — 2/ "L model backward"
  — 3/ "update parameters"
— **predict :** Cette fonction prend en argument un set de donnée, la liste de nombre de neurones par couche et les paramètres du réseau utilisé ainsi que la fonction d'activation utilisée. Elle renvoie les prédictions pour le set de donnée utilisée réalisées par le réseau de neurones utilisé.

3.2 Résultats avec algorithme de base

En utilisant un réseau de neurones de base avec en entrée 3072 neurones, en couche cachée 80 neurones et en sortie 10 neurones, un learning rate de 0.001, une taille de sous-ensemble (batch) de 128, une fonction d'activation sigmoide, une fonction de côût MSE et un nombre d'itération max de 10000 on obtient une précision maximale sur le set de test d'environ 40%. Le seul traitement apporté ici aux données est une normalisation des données en utilisant "pre processing" et la normalisation des batchs au seins de l'entraînement en utilisant "batch normalization".

On obtient la courbe d'entraînement suivant sur laquelle on observe bien le coût décroître au fil de l'entraînement ce qui nous confirme que le réseau de neurones apprend.

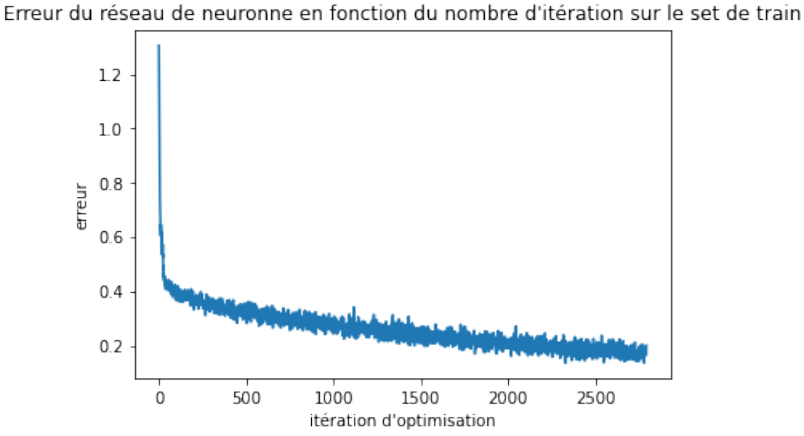

FIGURE 11 – Coût en fonction du nombre d'itération

Et on peut voir ici les courbes de précision d'entraînement et de test où on voit la précision de l'entraînement allant jusqu'à 100% dans le cas du sur-apprentissage et le décrochage aux environs de 35 à 40% de la précison de test.

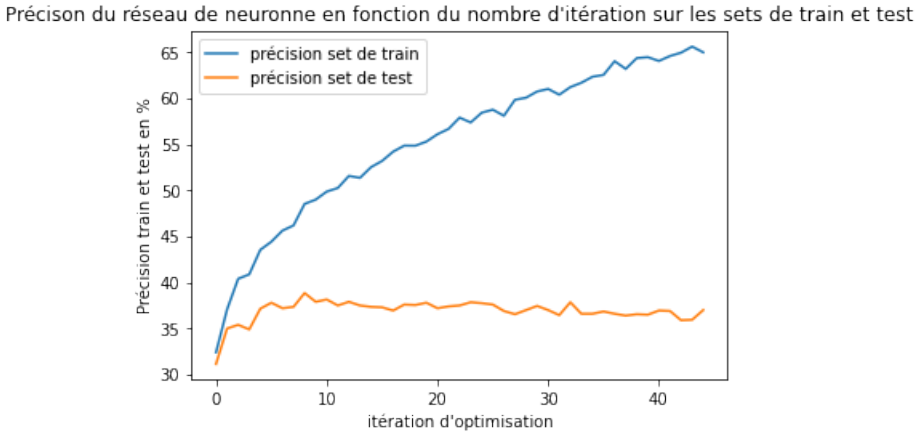

FIGURE 12 – Précision en fonction du nombre d'itération sur le set d'entraînement et de test

3.3 Résultats avec algorithme de base en faisant varier le nombre de neurones sur la couche cachée

En gardant les mêmes paramètres mais en faisant varier le nombre de neurones sur la couche cachée on va essayer de trouver le nombre optimal de neurones sur la couche caché pour obtenir un compromis entre vitesse d'entraînement et performance.

On observe les performances suivantes sur le set d'entraînement et sur le set de test :

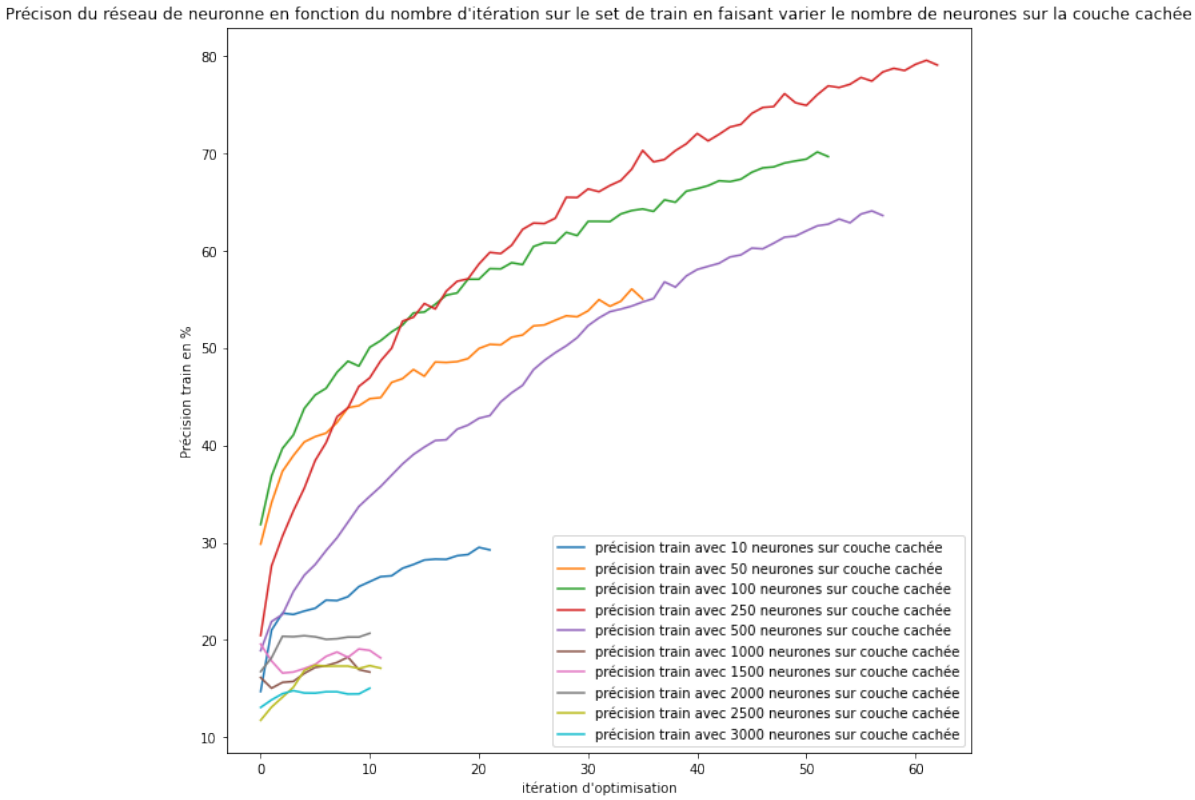

FIGURE 13 – Précision en fonction du nombre d'itération sur le set d'entraînement selon le nombre de neurones sur la couche cachée

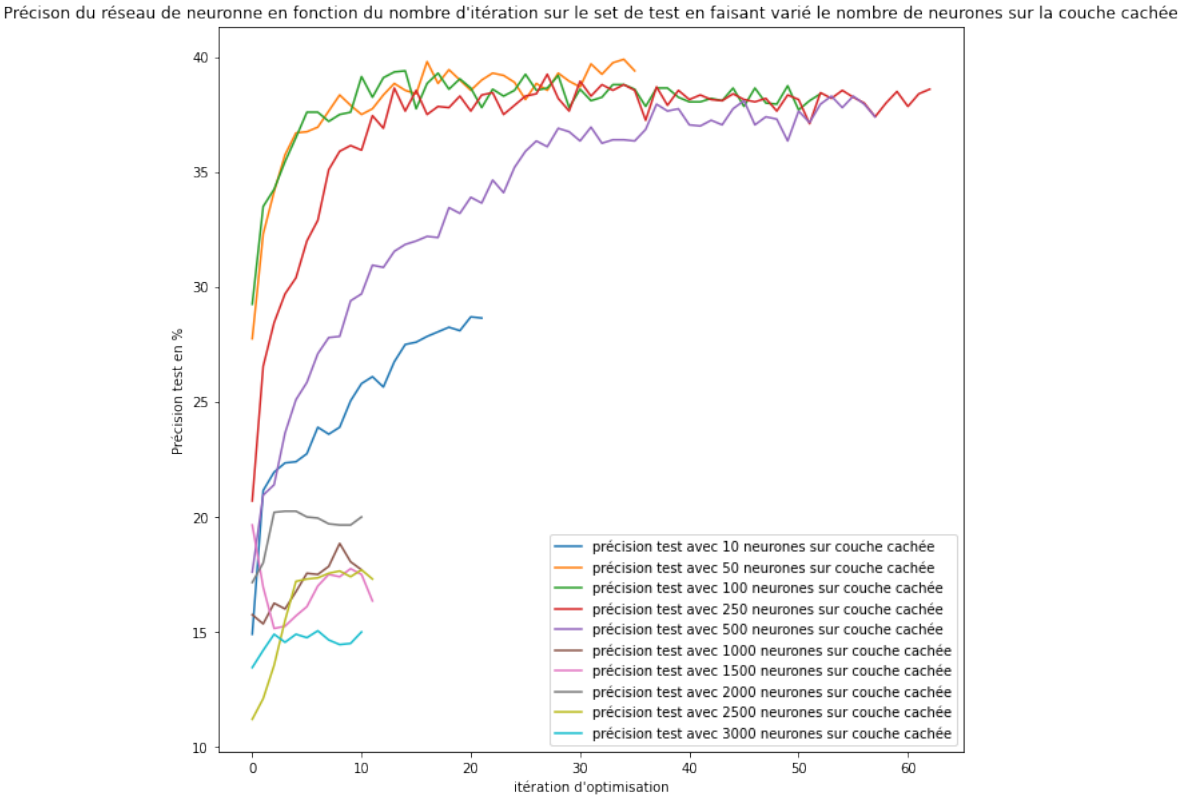

FIGURE 14 – Précision en fonction du nombre d'itération sur le set de test selon le nombre de neurones sur la couche cachée

On voit que le nombre de réseau de neurones optimal est entre 50 et 100 on pourra donc conserver 80 neurones sur la couche cachée pour la suite.

3.4 Résultats avec algorithme de base en ajoutant une couche cachée

En gardant les même paramètres avec la première couche cachée à 80 neurones et en ajoutant une 2ème couche cachée avec 50 neurones on peut voir sur les courbes de précisions qu'il n'y a pas d'amélioration particulière que ce soit en performance ou en vitesse. On gardera donc une configuration avec une seule couche de neurones.

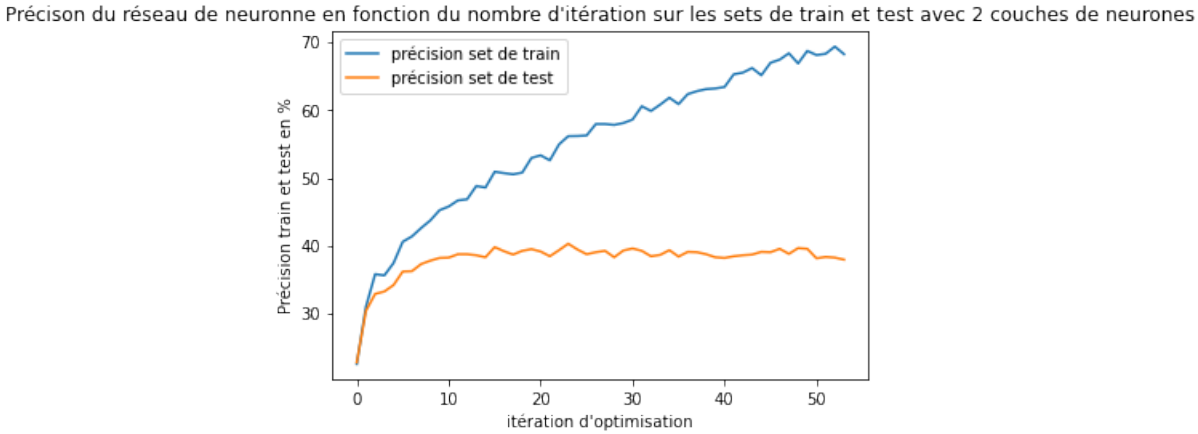

FIGURE 15 – Précision en fonction du nombre d'itération sur le set d'entraînement et de test pour 2 couches cachées

3.5 Résultats avec algorithme de base en faisant varier le learning rate

En gardant les mêmes paramètres mais en faisant varier l'hyper paramètre : learning rate lors de l'entraînement on va essayer de trouver l'hyperparamètre optimal pour entrainer le réseau de neurones.

On observe les performances suivantes sur le set d'entraînement et sur le set de test :

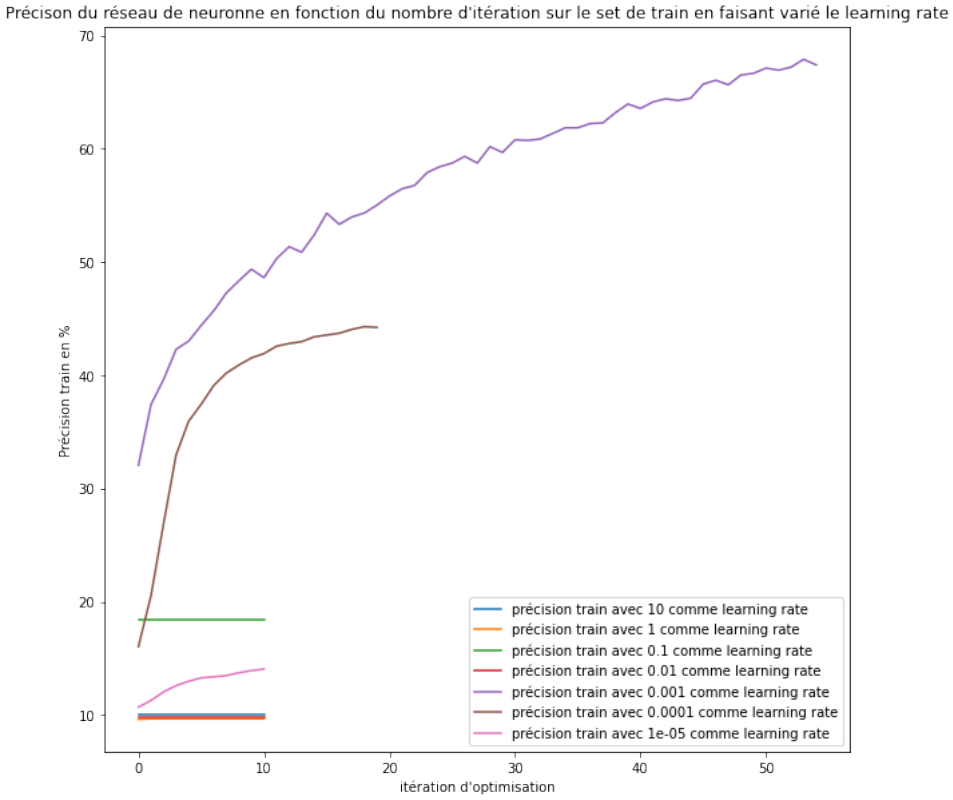

FIGURE 16 – Précision en fonction du nombre d'itération sur le set d'entraînement selon le learning rate

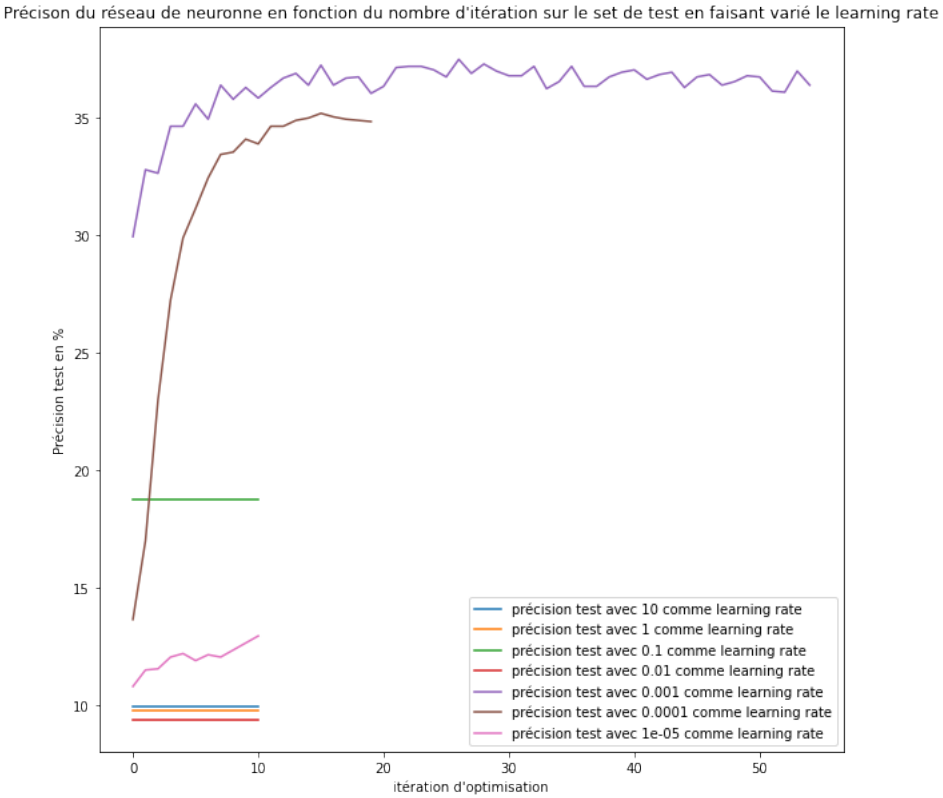

FIGURE 17 – Précision en fonction du nombre d'itération sur le set de test selon le learning rate

On voit que pour des learning rate trop élevés l'apprentissage n'est tout simplement pas possible. Avec des learning rate trop petits l'algorithme est trop lent à apprendre et se bloque avant d'atteindre sa précision maximale. On conservera comme learning rate optimal 0.001.

3.6 Résultats avec algorithme de base en faisant varier la taille des sous ensembles (batch)

En gardant les mêmes paramètres mais en faisant varier le nombre de données par sous-ensemble (batch) lors de l'entraînement on va essayer de trouver le nombre de donnée optimale permettant un entraînement rapide du réseau de neurones.

On observe les performances suivantes sur le set d'entraînement et sur le set de test :

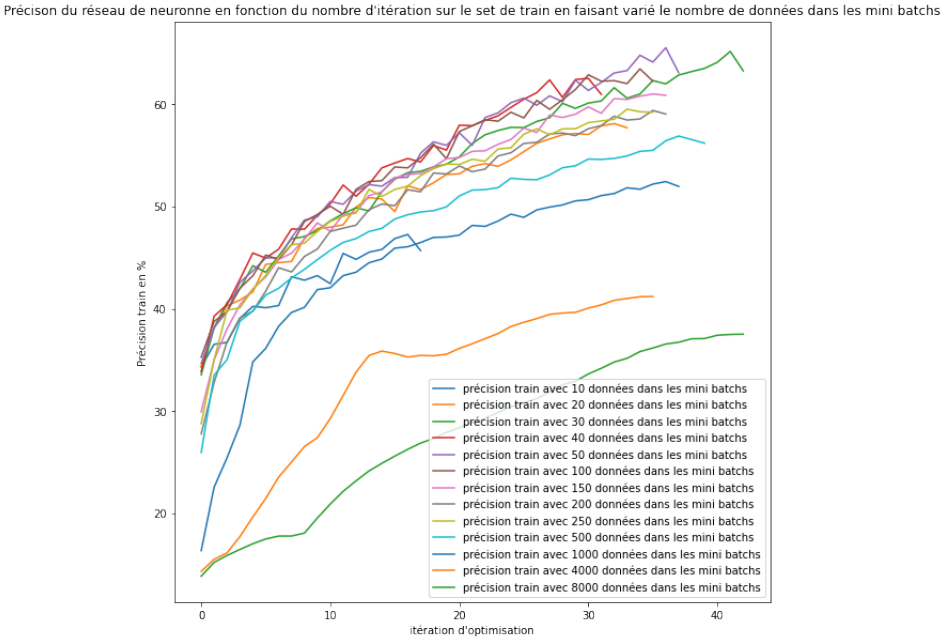

FIGURE 18 – Précision en fonction du nombre d'itération sur le set d'entraînement selon la taille des batchs

On remarque qu'en utilisant des mini-batchs entre 50 et 150 données par batch on permet d'être beaucoup plus rapide sur l'entraînement du réseau de neurones par rapport à utiliser directement toutes les données d'un coup pour entraîner le réseau.

3.7 Résultats en ajoutant un terme de régularisation

En gardant les mêmes paramètres mais en ajoutant un terme de régularisation (L2) à la fonction de coût lors de l'entraînement on va constater l'impact de ce terme de régularisation.

On observe l'évolution suivante du coût en ajoutant un terme de régularisation à cette fonction lors de l'entraînement :

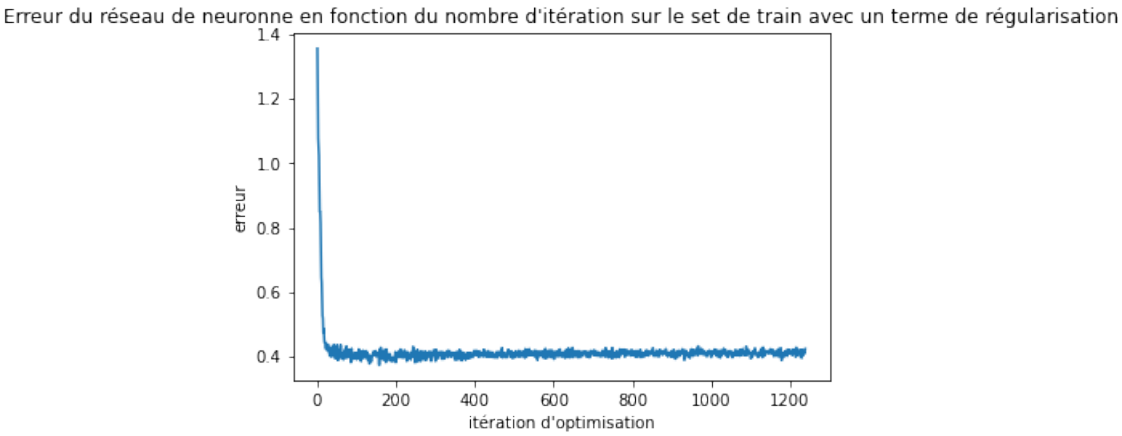

FIGURE 19 – Coût en fonction du nombre d'itération en ajoutant un terme de régularisation

On observe les performances suivantes sur le set d'entraînement et sur le set d'entraînement et de test :

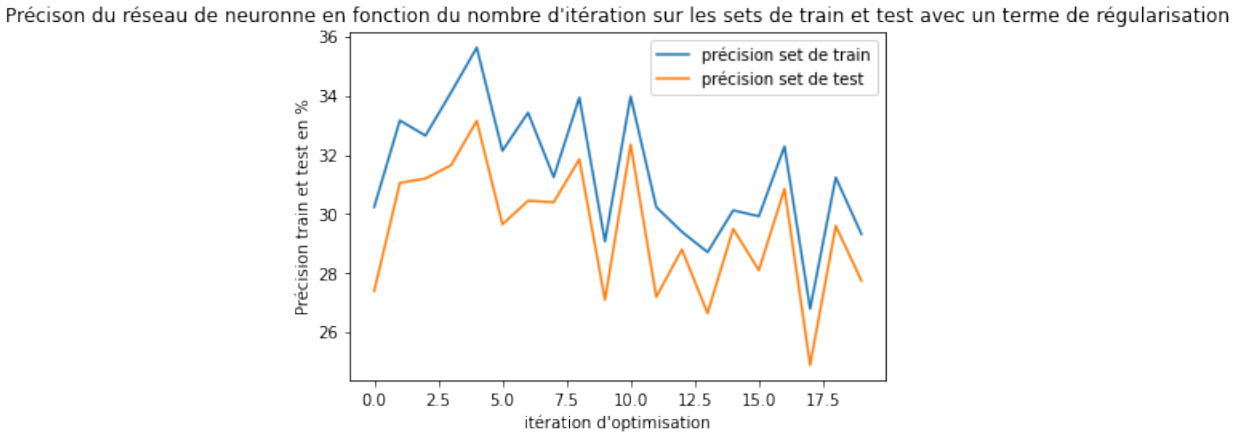

FIGURE 20 – Précision en fonction du nombre d'itération sur le set d'entraînement et de test en ajoutant un terme de régularisation

On constate que ce terme de régularisation permet d'empêcher le sur apprentissage étant donnée qu'il n'y a pas de décrochage entre la précision sur le set d'entraînement et le set de test cependant il ne contribue à l'amélioration de la précision.

3.8 Résultats avec la validation croisée

La validation croisée nous permet de déterminer de manière appropriée le paramètre optimal du nimbre de neurones sur la couche cachée en divisant le set d'entraînement en 5 sous ensembles sur lequel on entraînera l'algorithme sur 4 sous ensemble et on le testera sur le 5eme en répétant ce processus 5 fois pour que tous les sous ensembles soient au moins 1 fois le set de test. On peut observer le résultat de cette validation croisée sur le graphique suivant ou la courbe représente la moyenne des précision obtenue pour un nombre de neurones donnés et les points représentent la précision obtenu pour un nombre de neurones donnés sur chaque set de test.

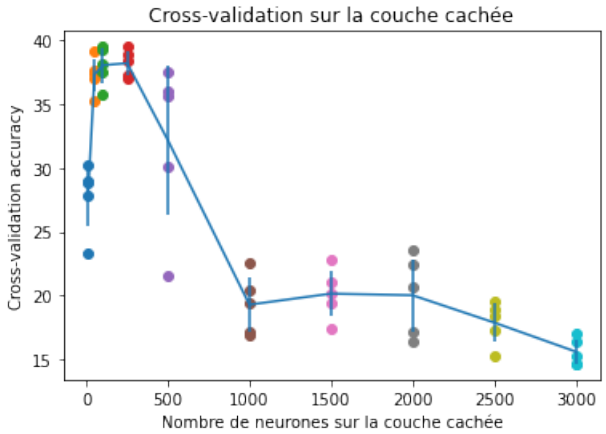

FIGURE 21 – Graphique de la validation croisée sur le nombre de neurones de la couche cachée

On constate que le nombre de neurones de la couche cachée optimal obtenu via la validation croisée est pour des nombres inférieurs à 500 neurones sur la couche cachée et que la précision décroît avec le nombre de neurones sur la couche cachée.

3.9 Résultats avec une fonction d'activation type sigmoîd et une fonction de coût type cross-entropy

On remplace ici la fonction de coût par une fonction type cross-entropy en conservant la fonction d'activation sigmoide et on obtient l'évolution du coût suivant :

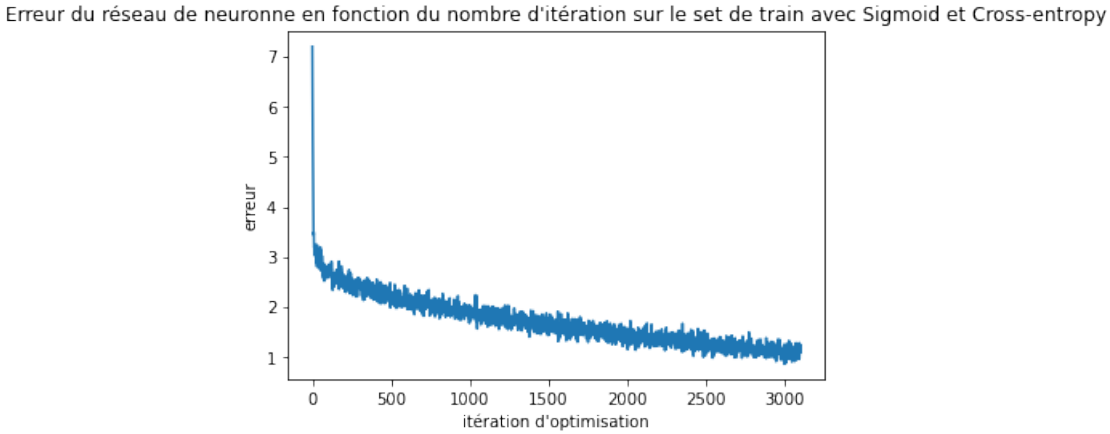

FIGURE 22 – Coût en fonction du nombre d'itération en utilisant la cross-entropy

Ainsi que les précisions sur les sets d'entraînement et de test suivantes :

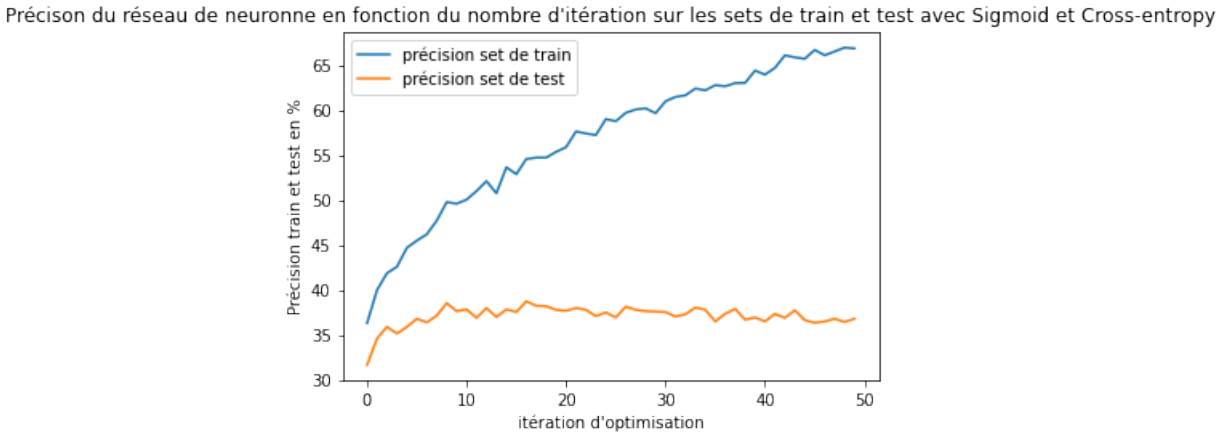

FIGURE 23 – Précision en fonction du nombre d'itération sur le set d'entraînement et de test avec une fonction de coût de type cross-entropy

On remarque que l'algorithme apprend correctement, on obtient une cours de coût diminuant avec le nombre d'itération ainsi qu'une précision augmentant jusqu'à 100% pour le set d'entraînement. En revanche on retrouve la même stagnation de la précision du set de test aux alentours de 35% et de plus on voit sur la courbe avec la précision qu'il y a besoin d'une régularisation supplémentaire

dans ce modèle car dés les premières itérations il y a un net décrochage entre la précision du set d'entraînement et du set de test.

3.10 Résultats avec une fonction d'activation type relu et une fonction de coût type MSE

On remplace ici la fonction d'activation par une fonction type Relu (leaky Relu pour être exacte) en conservant la fonction de coût MSE et on obtient l'évolution du coût suivant :

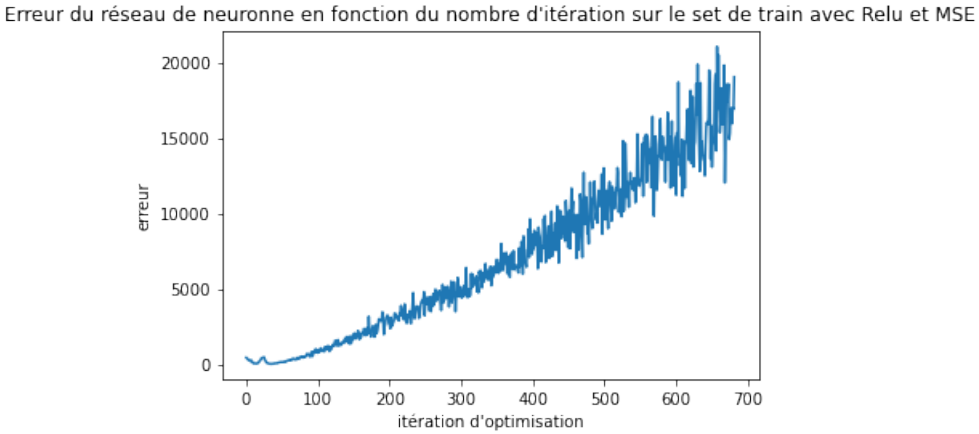

FIGURE 24 – Coût en fonction du nombre d'itération en utilisant la fonction leaky Relu

Ainsi que les précisions sur les sets d'entraînement et de test suivantes :

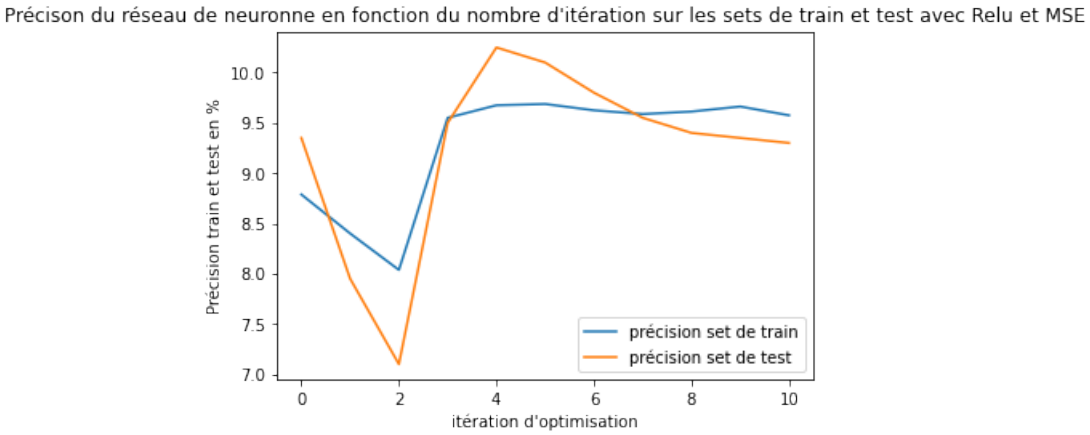

FIGURE 25 – Précision en fonction du nombre d'itération sur le set d'entraînement et de test avec une fonction d'activation leaky Relu

On remarque que l'erreur diverge vers l'infini et que la précision est de l'ordre de 10%, cela signifie que l'algorithme n'apprend pas et que la fonction d'activation leaky Relu n'est pas adaptée ici avec les paramètres utilisés.

3.11 Résultats avec l'utilisation de descripteur HOG

En utilisant un réseau de neurones de base avec en entrée 1024 neurones, en couche cachée 80 neurones et en sortie 10 neurones, un learning rate de 0.001, une taille de sous-ensemble (batch) de 128 et un nombre d'itération max de 10000 on obtient une précision maximale sur le set de test d'environ 30%. On transforme ici les images originalement en RGB en descripteur HOG puis on effectue la normalisation des données en utilisant "pre processing" et la normalisation des batchs au seins de l'entraînement en utilisant "batch normalization".

On obtient la courbe d'entraînement suivante sur laquelle on observe bien le coût décroître au fil de l'entraînement ce qui nous confirme que le réseau de neurones apprend.

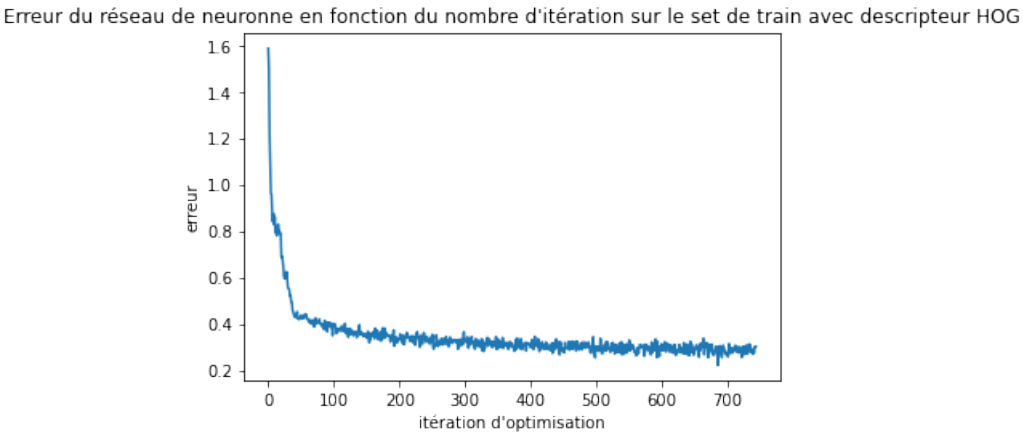

FIGURE 26 – Coût en fonction du nombre d'itération pour descripteur HOG

Et on peut voir ici les courbes de précision d'entraînement et de test où on voit la précision de test suivant correctement la précision d'entraînement.

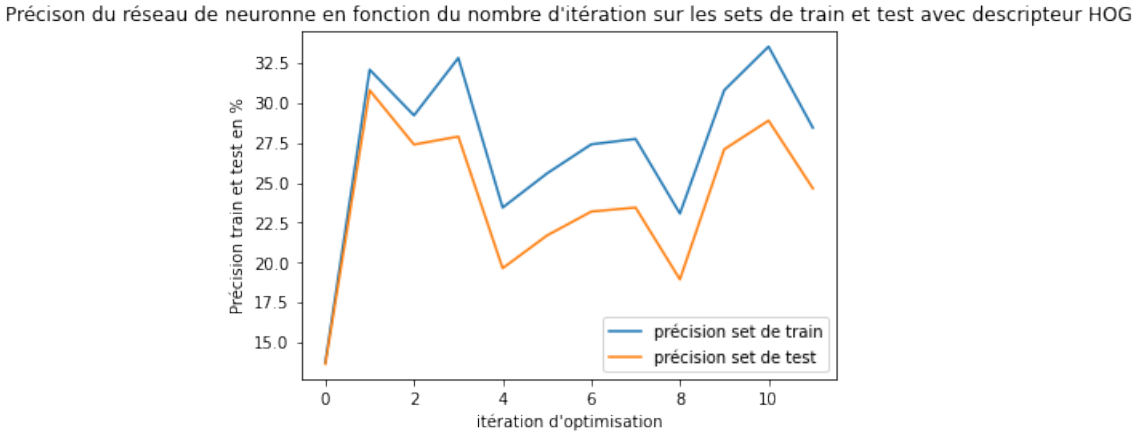

FIGURE 27 – Précision en fonction du nombre d'itération sur le set d'entraînement et de test pour descripteur HOG

On constate contrairement à l'algorithme de knn que l'utilisation de descripteur HOG avec les paramètres choisis n'améliore pas la précision de l'algorithme et la baisse même aux environs de 30% avec les mêmes paramètres. En revanche ces descripteurs semblent aider à ne pas sous ou

sur apprendre et améliorent la vitesse du fait de la diminution de nombre d'entrée.

3.12 Résultats avec l'utilisation de descripteur LBP

En utilisant un réseau de neurones de base avec en entrée 1024 neurones, en couche cachée 80 neurones et en sortie 10 neurones, un learning rate de 0.001, une taille de sous-ensemble (batch) de 128 et un nombre d'itération max de 10000 on obtient une précision maximale sur le set de test d'environ 15%. On transforme ici les images originalement en RGB en descripteur LBP puis on effectue la normalisation des données en utilisant "pre processing" et la normalisation des batchs au seins de l'entraînement en utilisant "batch normalization".

On obtient la courbe d'entraînement suivant sur laquelle on observe bien le coût décroître au fil de l'entraînement ce qui nous confirme que le réseau de neurones apprend.

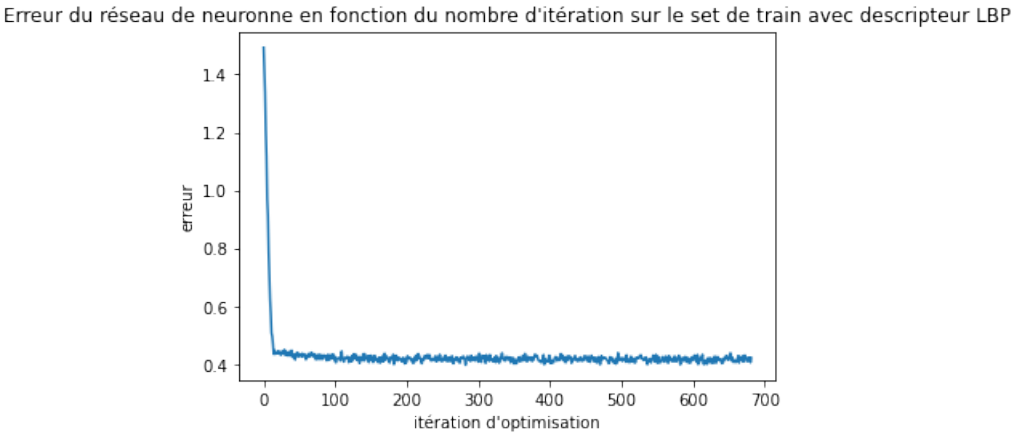

FIGURE 28 – Coût en fonction du nombre d'itération pour descripteur LBP

Et on peut voir ici les courbes de précision d'entraînement et de test où on voit que les précisions sont très médiocres et varient entre 10% et 15%.

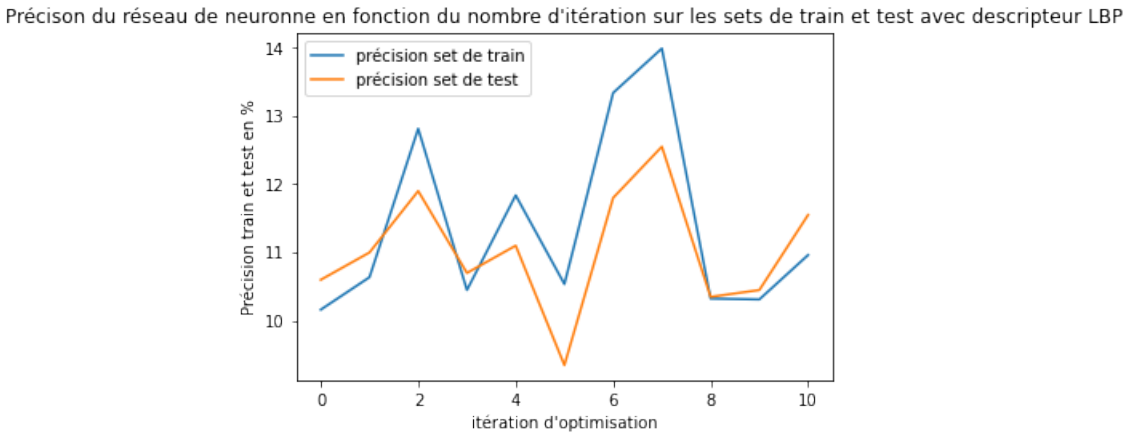

FIGURE 29 – Précision en fonction du nombre d'itération sur le set d'entraînement et de test pour descripteur LBP

On constate à l'instar de l'algorithme de knn que l'utilisation de descripteur LBP avec les paramètres choisis n'améliore pas la précision de l'algorithme à base de réseau de neurones et la baisse nettement.

4 Conclusion

Pour conclure ce TD on a pu voir qu'avec les données de base le réseau de neurones donne un meilleur résultat que l'algorithme des k plus proche voisins. En revanche en utilisant des descripteurs HOG l'algorithme des k plus proches voisins donne de meilleur résultats que le réseau de neurones. On peut néanmoins supposer que la marge d'amélioration du réseau de neurones est largement plus importantes que l'algortihme knn si on pouvait utiliser l'ensemble des 60000 images au lieu de 10000.