LA RECHERCHE RAPIDE DANS DE GRANDE BASES DE DONNÉES D'IMAGES

{TONG ZHAO} ENCADRÉ PAR MATHIEU AUBRY

LE PROJET DE MOPSI - ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES

MOVITATION

La banalisation des appareils photos et l'augmentation de la capacité de stockage permet l'analyse de dataset de plus en plus important. Etant donnée une image, un algorithme efficace pour chercher des images similaires dans une base de données est très demandé.

Figure 1: Le but du projet

L'idée principale de ce projet est de **faire de la recherche rapide dans de grande bases de données d'images en utilisant des descripteurs générés par un réseau de neurones**. Pour cela, des algorithmes de K-plus proches voisins(KNN) adaptés[2] sont proposés. Dans ce projet, l'algorithme de **Locality Sensitive Hashing (LSH)**[7] et l'algorithme de **Dynamic Continuous Indexing(DCI)**[6] sont implémentés et testés.

OBJECTIF

- Calculer les descripteurs des images en utilisant le réseau de neurones(Convolutional Neural Network, CNN) (Python + Keras[1])
- Implémenter l'algorithme de Locality Sensitive Hashing(LSH) basé sur quatres mesures de la distance différente - Cosinus, Jaccard, Hamming, Euclidean (C++)
- Implémenter l'algorithme de Dynamic Continuous Indexing(DCI)[6] (C++)
- Evaluer les modèles et comparer les résultats (Python)

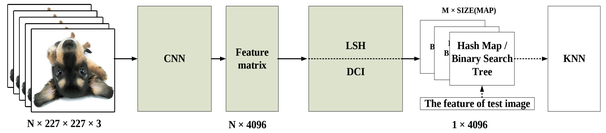


Figure 2: Architecture du système

APPLICATIONS

- **Le moteur de recherche**: Une application web permettant de trouver des ressources à partir d'une requête sous forme d'image
- **Le système de recommandation**: Trouver des images qui nous intéressent selon le cookie de HTTP

ALGORITHMES

1 L'Extraction des Descripteurs

Le réseau de neurones convolutif(CNN) ont révolutionné la classification d'images. Leur architecture nous permet d'extraire des caractéristiques sémantiques de nos images. Dans le but d'extraire des descripteurs d'images nous utilisons le réseau AlexNet Pré-entraîné[4][5] sur ImageNet. Pour être plus précis nous utilisons la sortie de la couche 'fc7'.

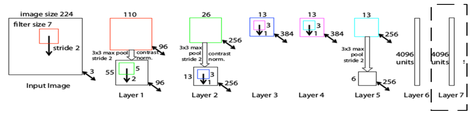


Figure 3: L'architecture d'Alexnet

2 Locality Sensitive Hashing[7]

LSH vise à accélérer le K-plus proches voisins en utilisant une famille de fonction de hachage choisies telles que des points proches dans l'espace d'origine aient une forte probabilité d'avoir la même valeur de hachage. Pour un point requête, on trouve les points hachés à la même position que ce point dans tous les tableaux correspondants.
On appelle $\mathscr{F}$ une famille LSH si tous les foncions $h : \Re^d \to \Re$ dans la famille satisfont les conditions suivantes. Pour deux points quelconques $p, q \in \Re^d$, un seuil$R > 0$ et un facteur d'approximation $c > 1$, on a:

- Si $d(p, q) \leq R$, alors $Pr_h[h(p) = h(q)] \geq P_1$
- Si $d(p, q) \geq cR$, alors $Pr_h[h(p) = h(q)] \leq P_2$
- $P_2 < P_1$

Dans ce projet, on implémente LSH basé sur 4 mesures de distance, pour chacunes d'entre elles nous choisissons une famille LSH.

2.1 La distance du cosinus

Soit $A_i = \begin{bmatrix} a_1 & a_2 & ... & a_{4096} \end{bmatrix} \in \Re^{4096}$ le descripteur de la i^{eme} image, $V_1, V_2, ..., V_n \in \Re^{4096}$ les vecteurs de hachage générés aléatoirement par la distribution Gaussienne $\mathscr{N}(0, 1)$, on calcule:

$$\begin{bmatrix} a_1 & a_2 & ... & a_{4096} \end{bmatrix}_{1\times 4096} \times \begin{bmatrix} V_1 & V_2 & ... & V_n \end{bmatrix}_{4096\times n} = \begin{bmatrix} x_1 & x_2 & ... & x_n \end{bmatrix}_{1\times n}$$

On prend une chaîne de caractères s de taille n telle que $s[j] = 1$ si $x_j > 0$ sinon 0. Ensuite nous insérons l'indice de l'image i dans le tableau de hachage dont la clé est $Hash(s)$ où $Hash$ la fonction standard en C++11. Pour augmenter la probabilité de collision, nous construisons m tableaux de hachage.

2.2 La distance de hamming

Soit $A_i \in \Re^{4096}$ le descripteur de la i^{eme} image, on prend un nouveau vecteur binaire $B_i = \{b_1, b_2, ..., b_{4096}\} \in \{0, 1\}^{4096}$ où $b_i = 1$ si $a_i > 0$ sinon $b_i = 0$. La famille LSH est:
$\mathscr{F} = \{h : \{0, 1\}^{4096} \to \{0, 1\} | h(b) = b_i \text{ pour certain } i \in \{1, ..., 4096\}\}$
On prend une chaîne de caractères s de taille n telle que $s[j] = h_j(A_i)$ pour tout $j \in \{|1, n|\}$ comme la clé du tableau correspondant.

2.3 La distance d'euclidean (E2LSH) [2]

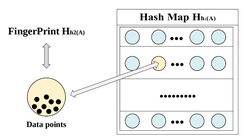


Figure 4: Le processus de calcul

On pose deux fonctions de hachage

- $h_1(a_1, a_2, ..., a_k) = ((\sum_{i=1}^{k} r_i a_i) \bmod C) \bmod Size_{table} \to$ pour choisir le tableau où on insère ce point
- $h_2(a_1, a_2, ..., a_k) = (\sum_{i=1}^{k} r_i^{'} a_i) \bmod C \to$ pour calculer la position où on insère ce point dans le tableau choisi

2.4 La distance de Jaccard

On prend le même vecteur B_i que pour hamming, mais on choisit MinHash[8] comme famille LSH.
Après avoir permuté aléatoirement les colonnes de B_i, on calcule la fonction $h(B_i) = argmin(B_i^{'})$. La chaîne de caractères s est construite comme suit: $s[4j : 4(j + 1)] = h_j(B_i)$ pour tout $j \in \{|1, n|\}$.

3 Dynamic Continuous Indexing[6]

Contrairement à l'algorithme de LSH, cet algorithme est basé sur la construction d'indices continus des points dans la base, ce qui nous permettent de faire des recherches rapidement. Ici on utilise l'arbre binaire de recherche au lieu du tableau de hachage.

L'algorithme est comme suivants:

- La famille LSH: $\mathscr{F} = \{h : \mathscr{R}^{4096} \to \mathscr{R} | h(b) = b_i \text{ pour certain } i \in \{1, ..., 4096\}\}$
- On regroupe L indices composites de taille m en utilisant $h_{ij} \in \mathscr{F}$ pour tout $i \in \{|1, m|\}$, $j \in \{|1, L|\}$. Pour chaque indice simple, on maintient un arbre binaire de recherche.

Ayant une image requête,

- Chaque indice simple vote pour un point plus proche dans l'arbre
- On se concentre sur chaque indice composite. S'il y a un point qui est déjà voté par tous indices simples, on l'ajoute dans la liste de candidats.
- Si la taille de la liste atteint N, on s'arrête, sinon on revient à la première étape.

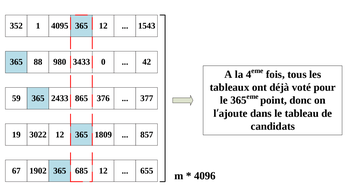


Figure 5: Le choix de points candidats dans DCI

RÉSULTAT EN IMAGENET

Concrètement, on calcule les descripteurs par Keras[1][4] et on implémente les algorithmes en C++. L'évaluation est basée sur ImageNet[3], une grande base de données de taille 1,000,000 en 1000 catégories prédéfinies. Compte tenu de la capacité de calcul de l'ordinateur, les tests suivants utilisent une base de données de taille 20,000 ayant 200 catégories.

1 L'influence des paramètres sur le modèle

Le stratégie de la validation croisée est:

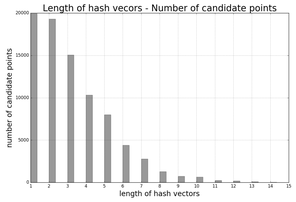
- Choisir aléatoirement 200 catégories dans ImageNet et 100 images dans chaque catégorie pour construire le train dataset.
- Choisir 100 images dans les catégories qu'on a choisies pour construire le validation dataset.
- On répète 10 fois ce processus et calcule la moyenne.

Sachant que pour chaque algorithme il y a des paramètres à optimiser, on choisit LSH avec la distance du cosinus comme objet d'analyse. Il y a deux paramètres dans cet algorithme:

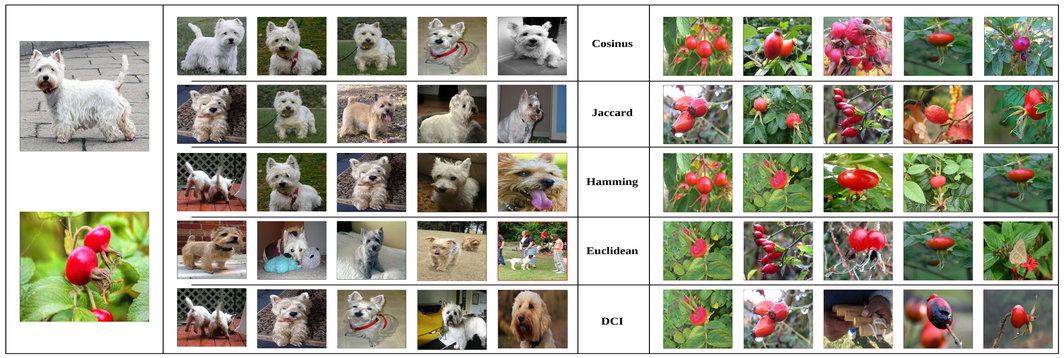
- L La longueur de la valeur de hachage: Plus la valeur est longue, moins de points candidat seront trouvés. Toutefois, la performance sera dégradée. Il faut trouver la valeur optimale pour ce paramètre sous la contrainte en minimisant l'erreur.
- n Le nombre de tableau de hachage: Plus nous avons de tableaux de hachage, plus de points candidats seront trouvés. Donc on veut maximiser ce paramètre en limitant le nombre de points candidats.

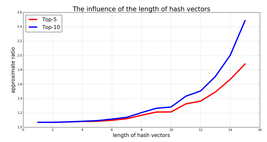
On utilise deux critères:

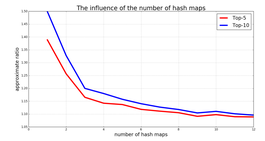
- Le nombre de points candidats trouvés
- Le ratio d'approximation, défini par la proportion entre le rayon de la boule qui contient les k-plus proches voisins approximatifs et celui de celle qui contient les vrais k-plus proches voisins.



3 Un exemple







En comparant les résultats précédents, on choisit $L = 6$, $n = 8$ comme paramètres pour la suite.
On utilise la même méthode pour les autres et on choisit les paramètres optimals suivants.

Table 1: Les paramètres choisis de l'algorithme

	LSH Cosinus	LSH Jaccard	LSH Hamming	DCI
la taille du vecteur	6	16	8	20
le nombre de tableaux	8	4	8	2
le nombre de candidats	-	-	-	3000

2 Le comparaison des méthodes différents

On compare les quatre algorithmes selon deux critères.
1) L'exactitude
Posons n le nombre de vrais k-plus proches voisins trouvés par l'algorithme, le critère pour l'exactitude est la proportion entre n et k.

Table 2: L'exactitude de l'algorithme

Algorithme	k = 1	k = 5	k = 10	k = 15	k = 25	k = 30
LSH Cosinus	70.00%	39.40%	24.10%	16.53%	9.92%	8.27%
LSH Jaccard	68.00%	36.00%	23.20%	16.53%	10.32%	8.67%
LSH Hamming	73.00%	47.80%	33.30%	25.20%	16.20%	13.67%
DCI Euclidean	69.00%	36.40%	20.30%	14.20%	8.72%	7.27%

2) Le temps
On estime la durée pour faire la recherche d'une image requête dans la base de données.

Table 3: Le temps d'exécution de l'algorithme

	LSH Cosinus	LSH Jaccard	LSH Hamming	DCI
Durée Moyenne	0.11823s	0.01283s	0.03852s	0.59722s(*)

(*) Le problème d'optimisation

CONCLUSION

- Le discripteur de reseaux de neurones se distingue de l'image originale en ce qu'il caractérise le contexte de l'image. Il a une meilleure performance en l'utilisant pour trouver des images similaires au lieu de trouver des images identiques.
- LSH simplifie les k-plus proches voisins effectivement, mais il faut trouver la valeur optimale pour chaque paramètre dans le modèle.
- Les distances se concentrent sur les caractérisations différentes(Hamming - Couleur, Cosinus - Forme...). Il faut plus de tests pour trouver leurs correspondances.
- On peut améliorer la méthode de binarisaition(Ex. la méthode d'Otsu).
- Il serai intéressant de faire une interface graphique dans laquelle nous pourrions visualiser automatiquement les résultats.

REFERENCES

[1] François Chollet. keras. https://github.com/fchollet/keras, 2015.

[2] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[4] heuritech. convnets-keras. https://github.com/heuritech/convnets-keras, 2016.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] Ke Li and Jitendra Malik. Fast k-nearest neighbour search via dynamic continuous indexing. *arXiv preprint arXiv:1512.00442*, 2015.

[7] Wikipedia. Locality-sensitive hashing — wikipedia, the free encyclopedia, 2016. [Online; accessed 14-February-2017].

[8] Wikipedia. Minhash — wikipedia, the free encyclopedia, 2017. [Online; accessed 14-February-2017].