UNIVERSITE
VINCENNES - SAINT-DENIS

Traitement Du Signal Et Des Images

CLASSIFICATION D'IMAGES

Encadrant:

Larbi BOUBCHIR

Elaborer par:

Youcef RAMOUCHE

Ahmed ZERHOUNI

2017-2018

Sommaire

1. Principe du projet:

Comme on le constate sur le titre du projet, cela consiste à classifier une image. celle-ci est particulièrement une figure d'oiseau que l'utilisateur choisie par rapport à notre base de donnée qui contient 6 catégories différentes d'oiseaux. Le résultat s'affiche en bas de notre fenêtre(GUI) où le rendu contiendra le nom de la catégorie à laquelle appartient cette image.

2. Elaboration du projet:

I. Fonctions prédéfinis:

Le concept de notre classification est basé sur ce qu'on appelle le "matching", qui est un terme utilisé lorsque on essaie de déterminer si deux images sont identiques. Pour cela on utilise quelques fonctions de l'algorithme déjà implémenté nommé le SURF (**Speed Up Robust Features).** Il est basé sur deux phase essentielles. D'abord, la phase de détection de ce qu'on appelle les "intrest points"(points d'intérêts) qui se passe à différents endroits de l'image, ils sont généralement situé dans les coins ou bien au niveau d'intersections d'objets importants dans l'image. leur détection dépends majoritairement de leur répétition dans l'image, le resultat obtenu est connu sous le nom d'un ***DETECTOR*** (fonction utlisé pour ça **imdetect_SURF(image)**). Ensuite, Se passe la phase d'extraction qui inclus le voisinage de chaque "intrest points"(points fort) ce qu'on appelle des caractéristiques locales le résultat obtenu est connu sous le nom d'un ***DESCRIPTOR***(fonction utlisé pour ça **imextract_DescriptorSURF(image,feat)**)***.*** Ce dernier est à la fois distinctive et robuste aux bruits...etc. D'ailleurs, ces "descriptor" après leur transformation sous forme de vecteur permettent de tester si deux images sont identiques (matching). celui-là est

notamment déterminer en calculant les distances entre les vecteurs des deux images(pour notre cas, ça sera la distance euclidienne).

immatch_BruteForce(W,W): Prends en paramètre deux descriptor qui font référence à deux images, il nous renvoie une matrice du matching brute forcer entre ces dernières.

imbestmatches(X1,X2,Y,Z): Prends en paramètre les features des images traité(Xi) sachant que chaque image après lui avoir appliqué le SURF, on obtient des features(points d'intérêts) et un descriptor. Le 3ème paramètre est obtenu à l'issue de la fonction "immatch_BruteForce", le dernier sera le nombre de features qu'on veut obtenir.

On notera aussi l'utilisation d'autres fonctions prédéfinis tels que le "uicontrol()" qui est nécessaire pour la manipulation de tout ce qui est sur l'interface utilisateur, que ce soit des boutons ou des zones textes. Il y a aussi l'autre fonction "uigetfile()" qui s'occupe notamment du chargement d'un fichier(image dans notre cas) nécessaire pour le déroulement du projet.

II. Les différentes parties du projet:

Notre projet est divisé en deux fichiers, le premier contient le des fonctions qui assure le fonctionnement interne, chacune à un rôle précis où elle dépends du résultat de l'autre pour arriver à l'objectif souhaité à la fin. Tandis que le deuxième fichier gère la partie graphique(interactions avec l'utilisateur).

i. Fonctions Principales:

- **Init_data()**

Tout d'abord on charge notre base de donnée qui contient 6 classes d'oiseau où chacune contient 100images.

On applique à chaque image la fonction "features_extract" puis on stocke le tout dans une liste où chaque élément contient(la matrice du descriptor, Sa classe et ces features(points forts))

- **features_extract(image)**

Une fonction qui est basé sur le SURF et ces deux fonctions:

->imdetect_SURF(image): celle-ci détecte les points forts de l'image passé en paramètre.

->imextract_DescriptorSURF(image,feat)): On obtient une matrice de descriptor de cette image.

- **extr_bst_match(im_model, list_mat):**

Elle prends en paramètre l'image modèle et la liste renvoyé par la fonction init_data() qui contient les "descriptor" d'images de notre base de donnée.

Tout d'abord, on applique la fonction "features_extract(im_model)" sur notre image modèle, le résultat de cette dernière sera stocké dans les deux variables M_descp et M_feat. Ensuite, on itère sur la liste qui contient les descriptor d'images de notre B.D et on applique les deux fonctions suivantes dans l'ordre suivant:

1) **immatch_BruteForce(M_descp, list_mat(i)(1)):**

   Elle prends en paramètre le descriptor de notre image modèle ainsi que les descriptor des images de notre base de données(une à la fois). une matrice de matching stocker dans "m" est renvoyé après avoir brute forcer le matching entre ces deux descriptor.

2) **imbestmatches(X1,X2,Y,Z):**

   Elle prends en paramètre les features des deux images brute forcé auparavant (Xi) ainsi que la matrice de matching "m"(Y) renvoyé par la fonction d'avant et enfin le nombre de points forts(Z) à extraire de notre matrice de matching. Une matrice "mout" est renvoyé par cette fonction qui contiendra les best_match(points forts) de la matrice de matching des deux images comparées avant.

Enfin, cette fonction renvoie une liste qui contient la matrice des points forts "mout" ainsi que la classe du descriptor comparé au descriptor de l'image modèle.

- **bst_match(list_desc, list_size):**

Elle prend en paramètre la liste renvoyé par la fonction init_data() ainsi que la taille de cette dernière.

Elle itère sur les 6 classes d'images de notre base de donnée. Premièrement, pour chaque descriptor de classe on lui applique la fonction "immatch_BruteForce" entre ce dernier et les autres descriptor de cette classe ce qui nous renvoie à chaque fois une matrice de matching. Deuxièmement, on applique la fonction "imbestmatches" qui nous renvoie une matrice contenant les best matching(points forts) de cette celle du matching par rapport au

nombre passé en argument. Finalement, On stocke la matrice des best matching ainsi que leur classe dans une liste(bst_matrice).

- **dist_eucl(im_mod_list_,final_list):**

Elle prend deux arguments en paramètre, le premier est la liste des best matching incluant la comparaison avec l'image modèle (renvoyé par la fonction "extr_bst_match") et le deuxième est la liste des matrices modèle comparé entre elles-mêmes (renvoyé par la fonction "bst_match").

Cette fonction est divisé en trois partie. Tout d'abord, une itération est faites sur la liste des best matching matrices de notre base de donnée "final_list", on remarque notamment un test où l'on calcule la somme de la matrice du descriptor d'une image si cette dernière est inférieur ou égale à 110 donc il s'agit de la même du coup on la prends pas en compte, ce qui optimise un peu le temps de calcul. Une fois le test vérifié, on transforme toutes ces matrices en vecteurs grâce à l'utilisation de l'opérateur "***(:)***", ce dernier sera stocké ainsi que sa classe dans une liste(finalList_vect). Ensuite, une itération est faite sur la liste des best matching matrices de notre image modèle "im_mod_list" afin de transformer la matrice des descriptor en vecteur, cette dernière est stocké dans une liste(matModel_vect). Enfin, on calcule la distance euclidienne entre chacun des vecteur de 'finalList_vect' et les vecteurs de 'matModel_vect', le résultat de cette distance sera stocké avec la classe du vecteur de 'finalList_vect' dans la liste 'vect_dist' qui sera retourner par la fonction.

- **cmp_mat(model_image, list_mat):**

Cette fonction fait appel à celle qui calcule la distance euclidienne dist_eucl() entre l'image modèle et les autres images de notre B.D, Elle se base notamment sur le principe du matching.

On voit une itération sur 'vect_dist' qui contient ce qu'à renvoyé la fonction dist_eucl(), pour déterminer la classe de l'image modèle on fait une comparaison qui permet de piocher la plus petite distance et renvoie ainsi la classe où était cette distance.

- **cluster(image_model):**

Elle s'exécuté lorsqu'on appuie sur le bouton *cluster* sur l'interface utilisateur. Il s'agit de la fonction principale qui fait tourner le projet, elle fait appel à deux fonctions, init_data() qui traite les images contenus dans notre base de donnée ainsi que la fonction cmp_mat().

Enfin, après exécution des deux fonctions, le résultat est renvoyé sous forme de chaines de caractères mentionnant la classe à laquelle appartient l'image entrée par l'utilisateur. Elle s'affiche en bas de notre GUI.

ii. Gestion de l'interface Graphique:

cette partie consiste à gérer les interactions avec l'utilisateur qui se base sur le chargement d'image et sa classification grace à des fonctions prédéfinis qui assure le bon fonctionnement du GUI tels que les callback fonctions qui sont associé à des entités graphiques présentent dans notre environnent.

3. Répartition du travail:

Il s'agissait d'une collaboration, ce qui veut dire qu'on a commencé par faire un plan de travail ensemble qui contenait les fonctionnalités souhaitées(les fonctions à développer). La plupart du temps ça commençais par un échange d'idées et de développement séparément qui se terminais par une combinaison entre les algorithmes développés en choisissant notamment les parties les plus performantes de ce dernier mais surtout le plus optimales puisque notre programme est basé essentiellement sur des calcules.

4. Problèmes et solutions envisagées:

On a dû réviser notre choix de comparaison des images plusieurs fois puisqu'on était partis au tout début sur une comparaison de matrice pure et dur(case par case) mais après consultation avec notre encadrant et d'autres recherches sur le web, il s'est avéré que la meilleure manière de tester le matching entre deux images était de calculer la distance euclidienne entre ces deux dernières.

5. Conclusion:

On a réussi à faire fonctionner le programme pour qu'il trouve un matching entre deux images étudiées avant les délais prévus mais on s'était lancé le défis de faire un programme qui arrive à reconnaitre des images n'appartenant pas à notre base de donnée autrement dit sur lesquelles il n'a aucune information, les seules données qu'il a c'est ce qu'il récolté sur une petite partie de cette classe.
Il s'est avéré qu'on avait besoin d'un réseau de neurone qui étudie les images d'une manière plus profonde.