

**TP3: Indexation de contenu pictural**

**François-Xavier Legault, 1876882**

**Jean-Sébastien Boulianne, 1828563**

**INF8770**

**Technologies Multimédia**

**Présenté à Guillaume-Alexandre Bilodeau**

**le 7 décembre 2020**

## Question 1 - Performance des algorithmes d’indexation

Alice : L’approche d’Alice lui permet d’identifier avec exactitude la vidéo où chaque image a été tirée ainsi que le moment auquel l’image apparaît. Puisqu’elle détecte une distance euclidienne nulle pour retourner un résultat, il y a exactement un seul résultat possible pour un tel algorithme en format PNG. Cependant, le coût d’une telle exactitude est élevé. La base de données dont Alice a besoin pour sa méthode est d’une grande taille. Conserver toutes les vidéos en format mp4 dans le cadre de ce travail prend 270 MB d’espace pour 50 vidéos. Dans un cas réel, avec des vidéos quelconques de longueur aléatoire, conserver les vidéos en leur entièreté dans un format d’origine devient rapidement coûteux. De plus, parcourir chaque trame de chaque vidéo est une manière très inefficace, quoiqu’exacte, d’identifier la trame correspondante. Le calcul pour la différence euclidienne entre deux images n’est pas long à lui seul, cependant lorsque ce test est effectué pour chaque trame d’une vidéo à 30 trames par secondes, la quantité devient grande très rapidement, et le temps requis pour l’opération totale augmente au même rythme. En format JPEG, les opérations effectuées pour compresser l’image risquent de rendre cette méthode moins efficace. Le résultat de la différence euclidienne ne sera jamais nul, donc il sera presque impossible pour cette méthode de fonctionner. Il faudrait trouver la différence la plus petite parmi toutes les distances calculées pour assurer un résultat. Outre cette petite différence, le temps d’exécution pour chaque type d’image ne change pas, seulement une petite partie dans la logique doit être modifiée et la méthode fonctionne de manière identique.

Bob : L’approche de Bob est exactement la même que celle d’Alice, à une différence près. Plutôt que de parcourir toutes les vidéos en leur entièreté, il décide de stocker toutes les trames de chaque vidéo sous forme d’images séparées. Ceci semble comme une bonne idée à vue rapide, cependant Bob n’a pas pris en compte les différences entre le temps d’ouverture d’un fichier d’image différent par rapport au temps requis pour passer à la trame suivante d’une vidéo en format MP4. En effet, le temps requis pour ouvrir un nouveau fichier d’image de type JPEG est en moyenne 5 fois plus long que de passer à la trame suivante dans une vidéo. Ce résultat a été obtenu en python avec la librairie OpenCV pour les deux opérations. Donc compte tenu de cela, l’algorithme de Bob pourrait être 5 fois plus long que celui d’Alice simplement basée sur cette statistique. De plus, l’encodage MP4 pour les vidéos sauvegarde grandement les différences entre chaque trame plutôt que toutes les trames en leur entièreté pour être plus efficace quant à l’espace disque utilisé, ce qui cause probablement la rapidité d’exécution du changement de trame, et donc une vidéo prend moins d’espace sur le disque que toutes les trames séparément. Donc l’algorithme de Bob est non seulement plus lent que celui d’Alice due au temps supérieur requis pour passer à la trame suivante, mais prend également plus d’espace sur le disque en sauvegardant chaque trame séparément. Finalement, pour ce qui est du format de l’image recommandé pour sa base de données, plusieurs cas sont possibles et chacun offre des avantages et des inconvénients. Si la recherche est effectuée avec une image de format PNG, utiliser ce même format pour la base de données ferait en sorte que le seul résultat nul de la distance euclidienne serait le bon, cependant l’espace nécessaire pour stocker les images serait très grand. Si les formats d’images utilisés pour la recherche et pour la base de données ne sont pas les mêmes (PNG pour recherche et JPEG pour la base de données, et vice versa), le résultat ne sera jamais nul, dû aux artéfacts de compression du format JPEG. Donc, si le format utilisé pour la recherche n’est pas connu, il serait idéal d’ajuster l’algorithme pour trouver le résultat le plus petit plutôt qu’un résultat nul, ainsi que d’utiliser des images en format JPEG pour la comparaison. De cette manière, la taille de la base de données serait minimale pour cette méthodologie, et Bob ne dépend pas d’un format précis pour que son algorithme fonctionne.

Carol : L’algorithme de Carol corrige beaucoup de problèmes que les solutions d’Alice et Bob ont. Sauvegardez une trame par seconde et aucune vidéo ne réduit considérablement la taille de la base de données. Grâce à un simple outil en ligne pour calculer la taille d’une vidéo, on peut conclure qu’une seconde d’une vidéo 720p, à 29.97 trames par seconde, ayant un *bitrate* de 2306 kb/s, prend 2.29 MB, alors qu’une image en format JPEG de même format (720p) dépasse rarement le seuil du MB. En plus de cela, le temps requis pour comparer deux histogrammes de couleurs est drastiquement moindre que la différence euclidienne. Finalement, la méthode de Carol utilise une base de données de moins grande taille, et les calculs sont plus rapides que les calculs des méthodes précédentes. Cependant, puisque Carol n’utilise qu’une trame sur 30 dans le cas des vidéos fournies, il est impossible pour elle de trouver le moment exact de la trame. Dans le pire des cas, elle aura une erreur d’une demi-seconde avec la réalité. Dans le contexte d’une application réelle, cette différence aurait peu d’importance. Le type d’image utilisé pour la recherche ou la base de données n’importe peu, puisque l’histogramme de couleurs sera très similaire pour la même image en format PNG ou JPEG. De plus, Carol trouve la distance la plus petite, donc il serait idéal pour elle d’utiliser des images JPEG.

## Question 2 - Implémentation d’un algorithme d’indexation

L’algorithme que nous avons décidé d’implémenter est celui d’Alice. Le temps total d’exécution en JPEG était de 4 heures et 25 minutes, soit environ 80 secondes par image en moyenne. En PNG, nous avons seulement testé les 50 premières images, et en moyenne le temps de recherche par image était 1.12 fois plus rapide. Ceci amène le temps total projeté à 3 heures 57 minutes, soit 71 secondes en moyenne par image. Ceci semble court à première vue, cependant le temps maximal requis pour une seule image est de 426 secondes pour une image JPEG, ou 380 secondes pour un PNG, ce qui équivaut à 7 minutes et 6 minutes 30 respectivement. Le minimum quant à lui est de 0.72 et 0.64 seconde pour JPEG et PNG respectivement. Dans une application réelle, cette marge est beaucoup trop élevée, il est impensable d’utiliser Shazam par exemple et de devoir attendre plus de 5 minutes pour obtenir un résultat. La raison de cette grande marge est simple : Alice itère à travers toutes les vidéos une à une, dans le même ordre pour chaque recherche. Donc, il se peut qu’une recherche soit instantanée, si l’image cherchée se trouve dans les premiers instants de la première vidéo. Cependant, si elle se trouve à la fin de la dernière vidéo, le temps nécessaire pour itérer à travers toutes les vidéos précédentes est inutile. Il serait idéal qu’Alice implémente une recherche rapide qui cerne la recherche à quelques vidéos, ou une seule idéalement, avant de parcourir cette dernière au complet.

## Question 3 - Implémentation de notre algorithme d’indexation

Pour notre propre algorithme, nous nous sommes basés sur l’algorithme de Bob ainsi que celui d’Alice, avec quelques modifications. Grâce à une banque d’images représentatives pour chaque vidéo, nous trouvons d’abord l’image ayant la distance euclidienne la plus près de l’image de recherche. Ensuite, nous itérons dans la vidéo correspondante pour trouver la distance euclidienne la plus basse afin de trouver le moment exact dans la vidéo où l’image apparaît. Grâce à cette première étape de raffinement de la recherche, nous sauvons une grande quantité de temps de recherche qu’Alice et Bob passent à itérer sur des vidéos inutilement.

Pour sa banque d’images, Bob décide de stocker toutes les trames de chaque vidéo sous forme d’image. Il se peut qu’une vidéo aie peu ou pas de mouvement entre chaque trame (c’est d’ailleurs le cas pour la vidéo *v28.mp4*). Donc, pour construire notre base de données d’image sans avoir trop d’images, nous avons d’abord itéré sur chaque vidéo, et calculions la distance euclidienne d’une trame avec les trames suivantes. Lorsque la différence était trop élevée selon un seuil, nous sauvegardions cette trame et recommencions le processus avec cette dernière comme trame de comparaison. De cette manière, si les mouvements de caméra sont minimes dans une vidéo, nous avons un minimum d’images clefs pour la définir, tandis que si les mouvements sont élevés ou grands, il y aura plus de trames sauvegardées. Par exemple, la vidéo 28 n’a qu’une seule trame sauvegardée, tandis que la vidéo 40, qui a beaucoup de mouvement, en a 44.

Notre méthode utilise également moins d’espace sur le serveur que la méthode d’Alice. Elle sauvegarde toutes les vidéos afin de faire sa recherche, alors que notre algorithme ne sauvegarde que les images clefs. Dans la majorité des cas, le volume total des images clefs ne dépassera pas le volume des vidéos entières. Dans certains cas il est possible que cela arrive, puisque le format MP4 est plus efficace en stockant seulement les différences entre les trames plutôt que l’image entière. Dans le cadre du travail, la vidéo 41 obtient 50 images clefs, qui prennent à eux un volume de 13 MB, alors que la vidéo fait moins de 6 MB. Cependant, dans une utilisation réelle, la moyenne des vidéos devrait être plus efficace en volume de stockage que les vidéos entières. De plus, nous avons simplement utiliser une fonction d’OpenCV directement pour sauvegarder les trames clefs. Nous aurions pus faire plus de compression sur ces images afin de réduire la taille, puisqu’elles ne servent que de guide pour cerner la recherche à une seule vidéo.

Voici l’extrait de code qui s’occupe de trouver l’image clef qui correspond le plus avec l’image de recherche :

for filename in os.listdir('data/video/frames'):

        frame = cv2.imread('data/video/frames/' + filename)

        if sum(img.mean(axis=0).mean(axis=0)) > sum(frame.mean(axis=0).mean(axis=0)):

            result = cv2.subtract(img, frame)

        else:

            result = cv2.subtract(frame, img)

        average\_result = result.mean(axis=0).mean(axis=0)

        brightness = sum(average\_result)

        if brightness < allTimeLow:

            allTimeLow = brightness

            nameAllTimeLow = filename

Lorsque le plus petit résultat est trouvé, le nom de la vidéo correspondante est sauvegardé, et la recherche parmi les trames de la vidéo débute :

while True:

        frameNum = frameNum + 1

        success, frame = vidcap.read()

        if not success:

            #fin de video

            break

        if sum(img.mean(axis=0).mean(axis=0)) > sum(frame.mean(axis=0).mean(axis=0)):

            result = cv2.subtract(img, frame)

        else:

            result = cv2.subtract(frame, img)

        average\_result = result.mean(axis=0).mean(axis=0)

        brightness = sum(average\_result)

        if brightness < allTimeLow:

            allTimeLow = brightness

            allTimeLowFrame = frameNum

            if brightness <= 30:

                found = True

        if found and brightness >= 50:

            break

Nous avons deux seuils pour la différence euclidienne, afin d’être sûrs que la méthode fonctionne avec le format JPEG. Comme déjà mentionnés, les images JPEG ne produiront pas un résultat nul, donc nous avons 2 seuils : un pour détecter la ressemblance, et le deuxième est simplement pour comparer les trames suivant un résultat positif, afin de s’assurer de conserver le résultat le plus petit. Pour une recherche en PNG, nous n’avions pas besoin de ces seuils, et nous avons utilisé un seuil nul. Comme discuté, lorsque la différence est faite avec un format PNG, le seul résultat nul possible est la trame correspondante.

Notre algorithme a réussi à trouver toutes les images en PNG dans la bonne vidéo, seulement une trame ou deux trop tôt. Donc notre erreur moyenne est de -0.02 seconde. Pour le format JPEG, cette erreur moyenne est de -0.03 seconde. Ceci est probablement dû aux quelques vidéos où les mouvements sont minimes, donc une caméra immobile ainsi que peu de mouvement parmi les sujets observés.

|  |  |  |
| --- | --- | --- |
| Temps d’exécution | PNG (projeté pour Alice) | JPEG |
| Algorithme d’Alice | 10 heures 46 minutes | 12 heures 6 minutes |
| Notre algorithme | 28 minutes | 31 minutes |

Les métriques de temps montrent l’efficacité de notre banque d’images clefs. Le fait que nous cherchons seulement la vidéo concernée sauve une quantité phénoménale de temps lorsque comparé à l’algorithme d’Alice. Le temps requis pour notre méthode ne représente que 4.3% du temps requis pour l’algorithme d’Alice.

|  |  |
| --- | --- |
| Volume de stockage | |
| Algorithme d’Alice | 270 MB |
| Notre algorithme | 92 MB |

Finalement, le volume de stockage requis pour notre base de données est effectivement nettement plus petit que celui d’Alice. Comme discuté précédemment, il y a des cas extrêmes où notre volume serait plus élevé, mais en pratique notre méthode reste plus efficace. Nous avons donc une base de données qui équivaut à 34% du volume de celle d’Alice.