Rapport de développement Projet Mondrian Algorithme 2 (Mondrian2.py):

Fonctionnalités:

Entrée: Image de taille quelconque, format JPEG ou PNG.

Sortie : Image « Mondrianisée », 512x512 pixels par défaut mais peut facilement être changé pour une taille plus grande avec très peu de pertes puisque l'image ne contient que des formes rectangulaires.

Paramètres: Prend comme seul argument le fichier contenant l'image à utiliser comme entrée. Les paramètres par défaut ont été optimisés expérimentalement pour produire un résultat satisfaisant en moyenne sur un grand ensemble d'images d'entrée, certes ce résultat n'est pas toujours le meilleur possible pour une image spécifique mais la modification des paramètres peut entrainer des résultats imprévisibles c'est pourquoi ils sont définis dans le corps du script et non lors de son appel. Ceci dit, si nécéssaire ils peuvent encore être modifiés en changeant leur définition dans le script. Les plus importants sont le nombre de couleurs différentes désiré (par défaut: 5) et le facteur de normalisation (par défaut: 10). De nombreux autres paramètres permettent d'influencer l'output mais ils ne sont pas pratiques à manipuler et ont généralement un effet minime.

Dépendances: Numpy, PIL (Pillow), OpenCV2, dlib*, face_recognition*

* Ces librairies sont uniquement nécessaires si on souhaite utiliser la reconnaissance faciale.

Conception:

L'algorithme se base sur la stratégie « split and merge » avec quelques adaptations pour mieux se rapprocher du résultat voulu. L'implémentation se fait comme suit: dans la phase « split » l'image est séparée en 256 régions carrées de tailles égales et chaque région est rendue homogène. Puis vient la phase « merge », si deux régions voisines sont homogènes entres elles (de la même couleur) alors elles sont fusionnées.

Plus précisément, homogénéiser les régions revient à appliquer une seule couleur à chaque pixel dans la région. On se demande alors quelle couleur choisir. Le premier choix est de calculer la moyenne des valeurs des pixels, c'est un choix attirant surtout parce que la moyenne d'une liste est très peu complexe à calculer et c'est un critère important puisque même une image de 512 par 512 pixels contient tout de même 262 144 pixels alors un calcul trop complexe pourrait être relativement long. Cependant le point négatif de ce choix est que la moyenne d'une liste est une valeur qui très souvent n'est pas contenue dans cette même liste, ce qui veut dire dans notre cas que ceci introduit des couleurs dans l'image de sortie qui n'étaient pas dans l'image de départ et ainsi on se retrouve avec une mosaïque de couleurs différentes or Mondrian utilise très peu de couleurs différentes. La médiane et le mode sont plus complexes à calculer mais donnent des valeurs contenues dans la liste de départ donc d'un point de vue purement artistique les deux sont préférables à la moyenne, ceci dit la médiane peut dans certains cas produire des couleurs peu représentatives de l'image de départ alors en général le meilleur choix est de calculer la

valeur de pixel la plus fréquente (en sachant que le temps de calcul de quelques secondes est tolérable). De plus le temps de calcul du mode de la liste des pixels de chaque région peut être réduit en diminuant le nombre d'éléments uniques dans cette liste, si on a une liste de n éléments tous différents alors le calcul se fait en $O(n^2)$ opérations mais si deux éléments de la liste sont identiques alors ils apparaissent le même nombre de fois et on n'a pas besoin de recalculer ce nombre ce qui réduit le nombre d'opérations nécessaires. La solutions dans notre cas est d'appliquer une fonction de normalisation à tous les pixels de l'image avant de calculer le mode. Par exemple si on arrondit les valeurs de chaque pixel au multiple de 10 le plus proche alors il n'y a plus que 26 valeurs possibles au lieu de 256, puisqu'un pixel est un triplet de ces valeurs ceci veut dire qu'une telle normalisation réduit le nombre de couleurs différentes possibles de plus de 16 millions à moins de 16 000, la probabilité d'avoir des valeurs qui se répètent dans la liste pour laquelle on cherche à calculer le mode est maintenant nettement plus élevée.
Cette normalisation avait aussi initialement pour but d'augmenter les chances que deux régions voisines soient considérées de la même couleur (puisqu'au tout début le critère de comparaison était une égalité stricte) mais cette utilité a disparue avec l'évolution du critère de comparaison qui sera détaillée plus loin. La normalisation garde néanmoins une utilité en plus de réduire la complexité, qui est d'augmenter les chances que le mode calculé soit bien la couleur la plus représentative de sa région. En effet par exemple une image pourrait contenir trois pixels de valeur (255,255,255), trois de valeur (254,254,254) et 4 de valeur (0,0,0), dans ce cas intuitivement on penserait que l'image est majoritairement blanche puisque 6 pixels sont blancs contre 4 noirs mais en réalité sans normalisation la couleur majoritaire de cette image serait le noir or le noir ne représente pas bien cette image. La normalisation permet donc aussi de réduire la probabilité de se retrouver dans un cas similaire.
Revenons maintenant au critère de comparaison. Comme dit précédemment la première version du critère de comparaison demandait une égalité stricte entre les couleurs de deux régions voisines pour les fusionner. Même avec une normalisation de facteur 10 il y avait souvent des régions de couleurs très similaires qui n'étaient pas fusionnées parce qu'une de leur valeur différait très légèrement. On pourrait alors augmenter le facteur de normalisation à nouveau pour que ceci se produise encore moins souvent mais si ce facteur est trop élevé alors la normalisation modifie trop les couleurs de l'image de départ et le résultat est peu satisfaisant. La solution était donc de calculer la distance euclidienne entre les vecteurs RGB des régions voisines et si cette distance est inférieure à une constante choisie alors on considère que les couleurs sont identiques.
Ceci introduit un nouveau problème, on se retrouvait avec des régions considérées comme homogènes alors que visuellement on pouvait distinguer des variations dans les couleurs d'une même région. Il fallait alors à nouveau choisir la couleur la plus représentative de chaque région fusionnée et appliquer cette couleur à chaque carré dans la région. On introduit alors une méthode qui regarde la liste des couleurs des 256 carrés et si deux d'entre elles sont considérées comme identiques par le critère de comparaison alors celle qui apparait le moins souvent est sortie de la liste, on obtient alors une liste dont chaque couleur est réellement unique et si au moment de recolorer chaque pixel on avait obtenu un mode qui n'est pas dans cette liste unique alors au lieu de prendre ce mode on utilise la couleur la plus proche de celui-ci dans la liste de couleurs uniques. Ceci garanti l'homogénéité au sein de chaque région fusionnée, mais au delà de ça grâce à cette liste de couleurs uniques on a un moyen très efficace de contrôler le nombre de couleurs différentes dans l'output. Or Mondrian dans ses oeuvres les plus populaires n'utilise principalement que 5 couleurs (noir, blanc, rouge, bleu et jaune). On définit alors une méthode qui incrémente la tolérance du critère de comparaison jusqu'à ce que la liste de couleurs finales ne contienne que 5 valeurs (ce nombre peut être changé en modifiant un paramètre dans le script si besoin, il est par ailleurs aussi possible de choisir qu'au moment de la normalisation des pixels on applique à chaque pixel la couleur de Mondrian qui lui est la plus proche mais le résultat est assez imprévisible et on perd souvent trop de détail pour reconnaître l'image de départ même si elles sont côte à côte, il est préférable d'utiliser les couleurs de l'image de départ mais en ne prenant que ses 5 couleurs les plus fréquentes).
Mais tout changement apporte également ses problème et cette fois ci n'était pas une exception à cette règle, en effet n'utiliser que 5 couleurs différentes produisait souvent des régions fusionnées très grandes, rarement rectangulaires et il restait ainsi peu de longues lignes noires pour rappeler le style de Mondrian. On choisi alors de tracer quelques lignes droites qui traversent toute l'image horizontalement et verticalement aux endroits où il reste encore le plus de petites lignes du quadrillage initial qui vont dans le même sens. Dès lors on obtient une image plus carrée et qui évoque mieux une oeuvre de Mondrian.

Il reste alors un seul problème. De nos jours les selfies sont très populaires mais en appliquant cet algorithme à de telles images on observe que les yeux sont souvent trop petits pour être repérés et la couleur des lèvres est souvent trop proche de celle de la peau pour ressortir sur l'image de sortie et sans yeux ni bouche c'est souvent difficile de reconnaitre qu'il s'agit d'un visage, c'est ce qui justifia le choix de donner l'option d'intégrer une librairie de reconnaissance faciale pour trouver la position des yeux et de la bouche (si l'image d'entrée contient un visage) afin de mettre en valeur ces régions sur l'image de sortie. Les yeux sont alors représentés en blanc mais puisque la couleur des lèvres peut varier relativement beaucoup d'une personne à une autre il est difficile d'assigner une seule couleur à la bouche quelque soit l'image. Le programme regarde donc les quelques couleurs les plus fréquentes dans les régions qui contiennent la bouche et choisi la couleur qui apporte le plus de contraste (la plus grande distance euclidienne moyenne) par rapport aux régions voisines (qui sont on suppose de la couleur de la peau) mais le résultat obtenu peut être assez approximatif, pour certaines images la bouche ne ressort pas très bien sur l'image de sortie.
Finalement quelques mots sur les limitations de ce script, au cours du développement de ce projet il est devenu apparent qu'Euclide n'avait pas pour habitude de travailler avec des vecteurs RGB. En effet sa définition de la norme vectorielle ne coïncide pas exactement avec la façon dont nos yeux perçoivent les changements d'une couleur à l'autre, ceci peut produire des résultats qui semblent incohérents mais restent mathématiquement corrects. Par exemple la norme de la différence entre cette couleur et celle-ci est moindre que entre cette même couleur et celle-ci. Il semblerait que les changements de luminosité des couleurs sont perçus comme moindre par nos yeux que des changements de même amplitude mais de teint. Une tentative de remédier à ce problème était de représenter les couleurs des pixels en format HSV et non RGB puisque ce format sépare le teint et la brillance de la couleur ainsi il est possible de pondérer le calcul de la différence entre deux couleurs de façon à mettre plus d'importance sur la différence de teint que sur celle de brillance. Ceci était efficace dans le sens où dans la liste des 5 couleurs à utiliser il y avait un plus grand contraste entre ces couleurs et en principe plus de contraste implique plus de détails mais l'inconvénient était que ceci impliquait aussi qu'au moment de repeindre les régions, si un mode d'une région ne faisait pas partie de cette liste de 5 couleurs (et qu'il fallait alors trouver la couleur la plus proche parmi ces 5 et appliquer celle-ci au lieu) il y avait souvent un écart bien plus significatif entre le mode de chaque région et la couleur qui est finalement appliquée à cette même région. Donc bien que cela rendait certaines parties de l'image finale plus détaillées, d'autres parties de la même image semblaient souvent anormales puisque leur couleur était trop modifiée, globalement les avantages ne compensaient pas les inconvénients. Il en est de même pour de nombreux autres aspects de cet algorithme et ceci sera la conclusion, durant le développement de ce script de nombreux problèmes sont survenus les uns après les autres, il n'y avait jamais une seule solution donc il y avait toujours un choix à faire et certaines images fonctionnent mieux avec certaines solutions que d'autres, rares étaient les cas où un choix était meilleur peu importe l'image testée et pour chaque problème de nombreuses solutions alternatives ont été essayées ce qui fait que ce script n'est autre qu'une collections de choix produisant le meilleur résultat observé en moyenne sur un grand ensemble d'images testées mais il est impossible de garantir que c'est le meilleur résultat possible pour une image donnée.

OSCAR THOMSEN