IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

PROJET CODEVSI N°110 : CATEGORISATION D'IMAGES PLEIN CIEL DE L'OBSERVATOIRE ASTRONOMIQUE

Rapport technique

À l'intention de Monsieur Dominique FABRE

AUTEURS :
AUBE Iris – étudiante FISE A1 à IMT Atlantique (campus de Brest)
BODOT Eve – étudiante FISE A1 à IMT Atlantique (campus de Brest)
ROLLET Julie – étudiante FISE A1 à IMT Atlantique (campus de Brest)
TRAN BINH Matias – étudiant FISE A1 à IMT Atlantique (campus de Brest)

A destination du COPIL de l'UE CODEVSI, de nos encadrants M. Frédéric Maussang, M. Alain Peden et de notre interlocuteur à l'Observatoire Astronomique De la Pointe Du Diable, M. Dominique Fabre.

Date : 23/05/2023
Version : 3.0

Résumé

Rédacteur : Iris / Relecteur : Eve – Iris– Julie - Matias

L'observatoire de la Pointe du Diable possède une caméra plein ciel qui photographie toutes les minutes le ciel depuis Plouzané. Cela génère des clichés analysables par les astronomes et passionnés du club. Néanmoins, cela produit également une importante quantité de photographies haute résolution non exploitables en raison de la météo, ou du fait qu'il fasse jour. Ce projet vise à implémenter un algorithme de tri automatique, permettant d'étiqueter les photographies de cette base de données en fonction de leur exploitabilité. Il vise à récupérer les images de l'observatoire, à les prétraiter à l'aide d'un tri jour/nuit, en implémentant l'ajout d'un cache et en les convertissant en niveaux de gris, avant de les soumettre à un algorithme de tri utilisant du machine learning. Ce dernier utilise la méthode des K voisins et a été préalablement entraîné sur une base de données fournie par l'observatoire, triée manuellement et comportant une importante quantité d'images. La finalité du projet est avant tout pratique, mais également écologique. En effet, il permet une exploitation plus simple des clichés pris par la caméra, et simplifie la suppression des photographies inutilisables pour éviter la saturation des disques durs de l'observatoire. Cet objectif a été en très grande partie atteint, le tri jour/nuit étant fonctionnel et le machine learning permettant d'obtenir un résultat avec une précision de plus de 80%.

Table des matières

I. Introduction

Rédacteur : Julie / Relecteur : Eve – Iris – Julie - Matias

L'observatoire astronomique de la Pointe du Diable situé à côté du campus d'IMT Atlantique Brest voit chaque année des étudiants, des membres du personnel et des personnes extérieures à l'école se rencontrer autour d'une passion commune : l'espace. Afin d'améliorer les capacités de cet observatoire, les étudiants d'IMT Atlantique ont pour habitude de participer à des projets en collaboration avec celui-ci. Le système conçu s'inscrit de ce fait dans une série de projets proposés au fil des ans autour de la caméra plein ciel ou « Allsky »* (par la suite tous les mots suivis d'un astérisque* seront définis dans le glossaire) de l'observatoire astronomique, mise en service il y a deux ans. Les premières étapes ont été de créer une interface avec la caméra sous forme de page web, puis de faire un récapitulatif des éphémérides*. L'objectif final de cette succession de projets est d'optimiser l'utilisation de la caméra plein ciel* et de rendre ses images accessibles par tous. De plus, ce projet permettra par la suite de mieux exploiter les photographies peu utilisées à ce jour en dehors de la page web présentant une image en temps réel (2 minutes entre chaque photographie) ainsi qu'un film des photographies de la nuit passée.

L'évolution actuelle de ce projet de longue date porte sur le tri des images prises par la caméra de la coupole. En effet, à ce jour l'observatoire stocke près de 2 ans d'images de qualités diverses, notamment du fait des conditions atmosphériques. Pour éviter de saturer ce stockage et ne garder que les images d'intérêt, une classification automatique de ces images serait préférable à un traitement manuel. C'est à cette problématique que Monsieur Dominique Fabre, désigné ci-dessous comme le Client, souhaite répondre. Le projet détaillé ci-dessous cherche à répondre aux deux objectifs particuliers de cette problématique. Le premier est de réduire l'empreinte carbone de l'observatoire, car les photographies sont stockées dans un NAS*, donc dans un cloud*, ce qui le rend polluant. De plus, sachant que le volume de données existant est de 3To, cette nouvelle implémentation a pour but de régler également la question de la capacité limitée du stockage du cloud de l'observatoire, et de préparer la base de photographies pour une utilisation ultérieure (recherche d'évènements astronomiques particuliers, ...).

Afin de répondre aux demandes du Client le projet présenté ici répond à la problématique du développement d'un algorithme de catégorisation d'images utilisant du machine learning* et s'adaptant à l'environnement de l'observatoire astronomique. Pour ce faire, le projet a été soumis à une phase de conception en vue de déterminer les méthodes et critères à implémenter et de répartir le projet au sein du groupe. Par la suite, un prétraitement a été appliqué aux photographies dans l'objectif de filtrer les premières images inexploitables. Les photographies ont ensuite été triées à l'aide de machine learning. Enfin, une série de tests a permis de valider le livrable et l'implémentation du projet au sein de l'observatoire.

II. Choix de la méthodologie et des critères de catégorisation

Rédacteur : Julie – Eve / Relecteur : Eve – Iris – Julie - Matias

La réalisation du projet a débuté par une phase de réflexion autour de la répartition des tâches à réaliser et des méthodes choisies afin de répondre au besoin du Client, tout cela en étroite collaboration avec ce dernier et les encadrants. De ce fait, des réunions ont été mises en place toutes les deux semaines pour assurer un meilleur suivi du projet.

a. Division du projet en parties

Rédacteur : Julie / Relecteur : Eve – Iris – Julie - Matias

Tout d'abord, le projet a été divisé en deux parties distinctes au niveau de la façon de coder et de concevoir le livrable. Cette organisation s'est avérée nécessaire pour respecter les délais en vue de la complexité et de la charge de travail à fournir et ainsi livrer une solution satisfaisante au Client.

En effet, la partie machine learning a été séparée du reste du projet puisqu'elle nécessite une connaissance fine des méthodes existantes. De plus, ce découpage permet aux personnes en charge du machine learning de se concentrer sur cette partie difficile sur laquelle repose une part importante du projet.

Une autre partie du groupe s'est de son côté employée à trier la base de données, créer le code de tri jour/nuit, le reste des prétraitements et enfin la forme de sortie du programme. De ce fait, un tri manuel d'une base de données d'un mois d'images, soit 33 448 images données par le Client, a été effectué en fonction des critères de tri définis par la suite avec ce dernier. Cette base de données étiquetée a été essentielle pour les tests de l'algorithme de tri jour/nuit ainsi que pour toutes les étapes du machine learning, notamment les tests.

b. Structure du logiciel

Rédacteur : Julie –Eve / Relecteur : Eve – Iris – Julie - Matias

Dans l'intention de faciliter la compréhension des différentes parties du rapport et du système, il est important de mettre en avant la structure de ce dernier. Ceci permet de voir la contribution des différentes facettes du projet à l'algorithme final, dont la structure est présentée par la figure 1. Cette structure sera détaillée de façon plus approfondie tout au long du rapport.

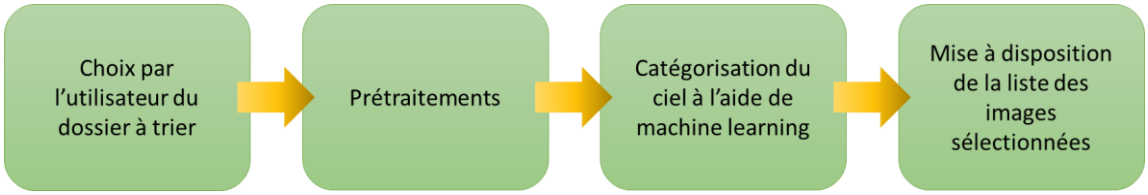


Figure 1 - Logigramme de la structure de l'algorithme final

c. Choix des critères de tri

Rédacteur : Julie / Relecteur : Eve – Iris – Julie - Matias

De manière à trier en différentes catégories toutes les photographies, il est nécessaire de définir le nombre de catégories souhaité et les critères de choix. Divers critères de tri sont disponibles [1] et un choix a donc dû être effectué en amont de l'écriture des différents algorithmes. Cinq catégories ont été retenues tout au long du projet : jour, nuit, ciel dégagé, ciel avec une couverture nuageuse partielle et ciel avec une couverture nuageuse totale (voir figures 2, 3 et 4). Ces catégories correspondent à la demande du Client et les critères suivants ont été établis avec lui.

Photographies illustrant chaque catégorie de nuit :

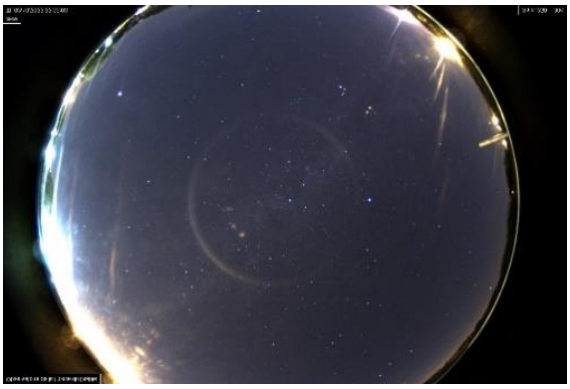


Figure 2 - Photographie illustrant la catégorie ciel dégagé

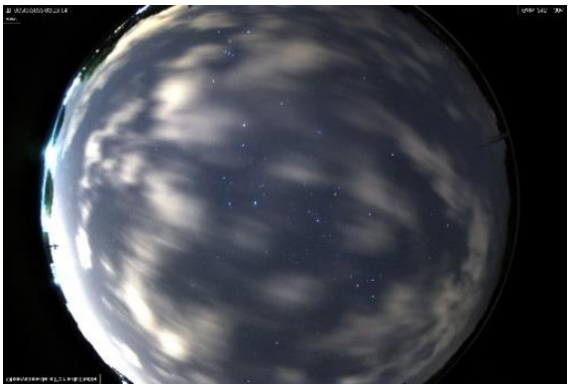


Figure 3 - Photographie illustrant la catégorie ciel partiellement nuageux

Figure 4 - Photographie illustrant la catégorie couverture nuageuse totale

i. Différentiation Jour/ Nuit

Après avoir défini ces catégories, il a été essentiel de déterminer les critères d'acceptation ou de rejet d'une image. La différentiation entre jour et nuit se base sur le début de la nuit astronomique*, soit environ une heure et demie après l'heure du coucher du soleil comme demandé par le Client. Il s'est révélé nécessaire de préciser ce "environ" en regardant les photographies de la base de données. Les critères principaux pour déterminer si la photographie a été prise la nuit ou non, ont été l'apparition des premières étoiles et la disparition de la couleur bleue dans le ciel. Cette sélection s'est faite sur des photographies de la base de données présentant un ciel dégagé et partiellement couvert afin d'étudier l'impact des nuages. De plus, il est important d'adapter la sélection de l'heure limite entre le soir et le matin aux critères précédemment décrits car l'angle du Soleil n'est pas le même. En effet, le ciel ne présente pas la même évolution de couleurs ni la même vitesse d'évolution entre ces deux périodes de la journée. Cette différentiation est cependant difficile à faire au vu de l'impact de la lumière des lampadaires sur les photographies.

De ce fait, la référence du crépuscule astronomique, c'est-à-dire la position du Soleil dépassant les 18° sous l'horizon, a servi de base à cette phase du projet. Cependant, cette limite ne permettait pas de voir les étoiles à cause d'un contraste trop faible provoqué par la pollution lumineuse environnante. Nous avons donc relevé la limite jour/nuit à une heure et demie après le coucher du soleil et avant le lever du Soleil, toujours en accord avec le Client. Nous avons alors obtenu un angle de différentiation de 26° (explication plus détaillée dans la partie V.b. Implémentation, Tests et validation). Cette différentiation peut être remarquée dans le niveau de bleu des photographies en figures 5 et 6.

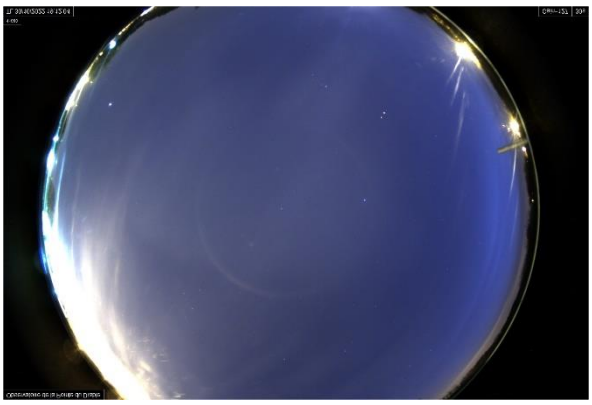


Figure 5 - Photographie considérée comme de jour

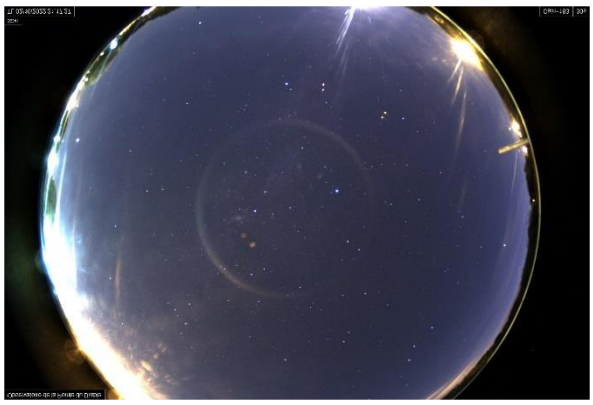


Figure 6 - Photographie considérée comme de nuit

ii. Différents niveaux de couverture nuageuse

Il restait ensuite à définir les critères d'acceptation et de rejet pour les différentes catégories de ciel nocturne. Ces distinctions ont été nécessaires pour faire le tri initial et manuel de la base de données. Une image considérée comme "dégagée" ne présente quasiment aucun nuage dans le centre de l'image. La photo étant prétraitée, ses bords sont en effet masqués par un cache (voir la partie III. Prétraitements et première phase de tri), et la présence ou non de nuages dans la zone en bordure d'image n'a pas été prise en compte. Une image contenant des gros aplats de nuages masquant une très grande majorité de la zone centrale (plus de 70%) est considérée comme "totalement nuageuse".

Cependant, une catégorie intermédiaire a été élaborée, contenant des images nuageuses mais laissant apercevoir des points étoilés exploitables. Cette catégorie contient les images nuageuses dont les nuages recouvrent moins de 70% de la surface, en accord avec le cahier des charges et donc du Client. De façon à mettre en évidence les limites de chaque catégorie une comparaison entre différentes photographies et les catégories qui leur ont été attribuées est proposée ci-dessous.

Pour le seuil de couverture nuageuse entre un ciel dégagé et un ciel partiellement couvert on peut réaliser la comparaison suivante, (figures 7 et 8) avec les critères explicités précédemment :

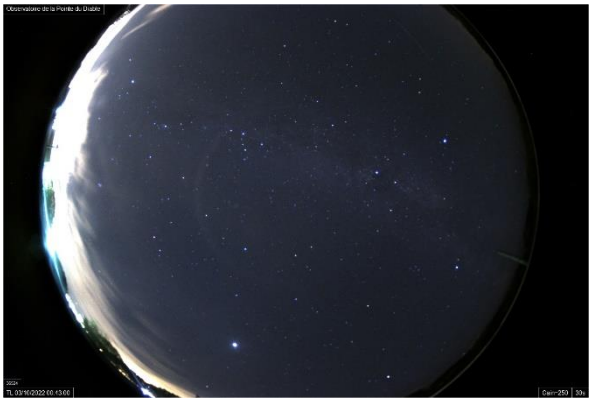


Figure 7 - Photographie de ciel dégagé

Figure 8 - Photographie de ciel partiellement couvert

En effet, dans la figure 8 le nuage restera visible même après l'application du cache, c'est donc un ciel partiellement nuageux. Sur la figure 7, cependant les seuls nuages présents se trouvent sur le bord inférieur gauche qui se trouveront masqués par la suite, c'est donc un ciel dégagé.

Pour le seuil de couverture nuageuse entre un ciel totalement couvert et un ciel partiellement couvert on peut réaliser la comparaison suivante (figures 9 et 10):

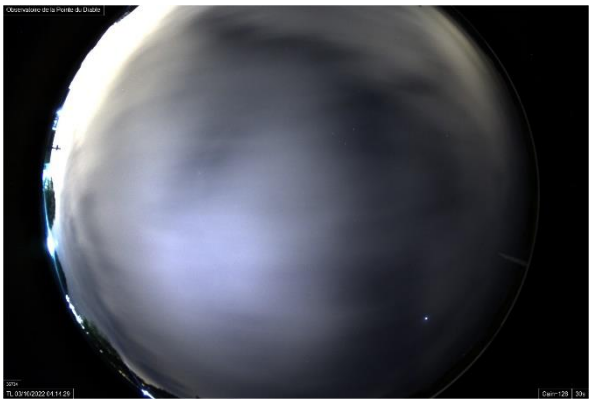


Figure 9 - Photographie de ciel totalement nuageux

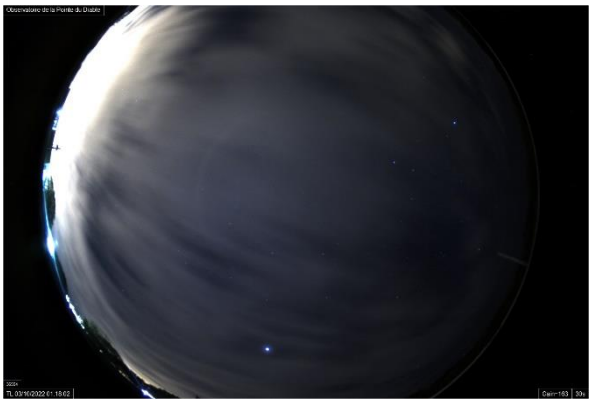


Figure 10 - Photographie de ciel partiellement couvert

Les nuages recouvrent en effet plus de 70 % du ciel dans la figure 9, c'est donc un ciel totalement couvert. Quant à la figure 10, les nuages bien que fins recouvrent environ un peu moins de 60% de la photo et des étoiles sont visibles. On a donc un ciel partiellement nuageux.

Maintenant que les catégories et les critères de choix ont été spécifiés, le cœur du projet peut être abordé, c'est à dire le machine learning et le codage. En conséquence, il a d'abord été nécessaire de se former à ces techniques de codages spécifiques et à l'environnement astronomique dans lequel évoluera le système. C'est pourquoi des séances d'auto-formation ont été nécessaires au début du projet, notamment pour l'apprentissage de diverses méthodes de machine learning. A la suite de cela, il a été établi qu'avant de débuter le machine learning, un premier tri devait être effectué, de même qu'un prétraitement des images, afin de faciliter leur catégorisation future. Ces différentes étapes suivent la structure établie précédemment et permettent ainsi de répondre à la problématique.

III. Prétraitements et première phase de tri

Rédacteur : Eve-Iris / Relecteur : Eve – Iris – Julie - Matias

Afin de pouvoir exploiter de manière optimale les images devant être gardées et les classer dans les catégories énoncées précédemment, une phase de prétraitement a été nécessaire.

a. Tri jour/nuit

Rédacteur : Eve / Relecteur : Eve – Iris – Julie - Matias

Une première partie du prétraitement des images permet de les catégoriser comme photographiant le ciel diurne ou nocturne. Pour cela, un algorithme de tri jour/nuit a été réalisé (voir figures 20 à 26 de la partie b. de l'annexe). Celui-ci est découpé en cinq phases, présentées dans la figure 11 ci-dessous :

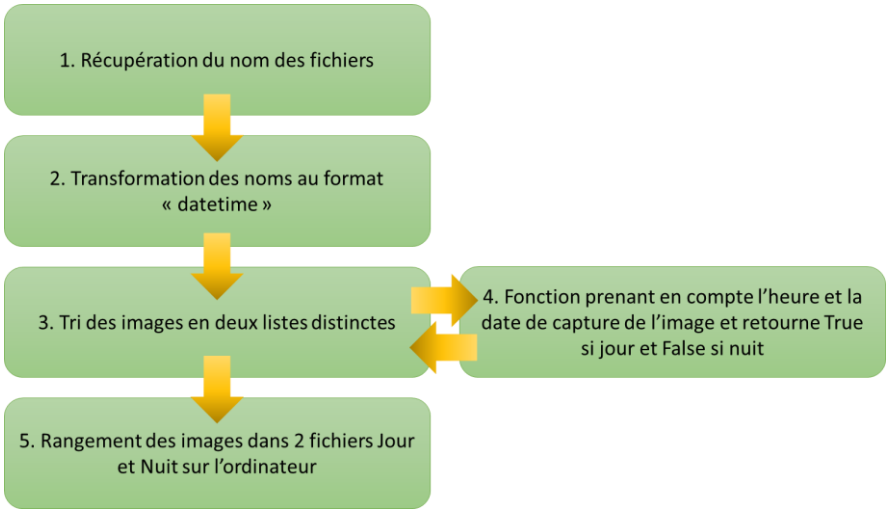


Figure 11 - Diagramme illustrant le fonctionnement de l'algorithme de tri jour/nuit

La première phase permet de récupérer le nom de tous les fichiers à trier. Néanmoins, le nom de ceux-ci est sous la forme « année_mois_jour__heure_minutes_secondes.jpg ». La deuxième phase a donc été de transformer leur nom au format « datetime » afin qu'ils soient exploitables dans l'étape numéro quatre. La troisième consiste à trier les images en deux listes distinctes : jour et nuit. Ces premières étapes servent de support à la quatrième. En effet, celle-ci consiste à utiliser les heures de

capture des images pour savoir si elles ont été prises lorsqu'il faisait jour ou nuit. L'algorithme prend en compte la nuit comme environ une heure et demie après le coucher du Soleil, et en retourne respectivement True ou False. Cette partie du code n'a cependant pas été entièrement écrite par notre groupe. Une base a été trouvée sur le forum du site Openclassrooms [2] et a été modifiée par la suite dans l'objectif de répondre aux attentes du Client. La dernière étape consiste quant à elle à ranger les images dans deux dossiers distincts sur l'ordinateur.

Toutes ces parties forment un algorithme final dont le fonctionnement est expliqué ci-après. La première partie est composée d'une unique fonction. On entre tout d'abord un chemin d'accès au dossier où sont situés les fichiers à trier. Puis, à l'aide de la bibliothèque « os » de Python on récupère le nom de tous les fichiers contenus dans le dossier. Enfin, la fonction les place dans une liste. Cependant, les noms des fichiers ne sont pas sous la bonne forme, comme expliqué précédemment. C'est pourquoi la deuxième partie utilise une fonction prenant la liste antécédente et pour chaque fichier de cette liste, transforme son nom au bon format « datetime » à l'aide d'une autre fonction créée pour le faire, puis les implémente dans une nouvelle liste. Par la suite, une nouvelle étape entre en jeu. Une fonction prenant en argument la nouvelle liste créée permet de la trier en deux nouvelles listes, nuit et jour. Pour cela elle utilise un algorithme qui détermine si la photographie a été prise le jour ou la nuit. Cet algorithme [2] prend en compte la latitude et la longitude du lieu de capture des photographies ainsi que la date et l'heure sous forme « datetime » à laquelle elles ont été prises.

Après les calculs d'heure limite entre jour et nuit, l'algorithme retourne True s'il fait jour et False s'il fait nuit, ce qui permet de les catégoriser en deux listes. Pour garantir la demande du Client de ne garder que les images prises une heure et demie après le coucher du Soleil et une heure et demie avant le lever du Soleil, quelques essais ont été effectué. Ils ont permis de déterminer la limite de cette nuit « astronomique », qui est lorsque le Soleil est à 26 degrés en dessous de l'horizon pour les latitudes et longitudes prises en compte. Cette valeur a ainsi été implémentée dans l'algorithme car la base trouvée ne répondait pas à la demande du Client ni au cahier des charges établi avec lui. Enfin, à l'aide des bibliothèques « shutil » et « os », on place les éléments des deux listes dans des dossiers « Jour » et « Nuit » créés sur l'ordinateur. Pour cela, l'utilisateur indique l'emplacement où il veut créer ces dossiers et des boucles "for" s'occupent ensuite de diriger les éléments des listes dans les bons dossiers.

La première étape du prétraitement permet d'éliminer près de 50% des photographies avant qu'elles subissent les étapes suivantes. Ceci représente donc un gain de temps pour les algorithmes mis en place par la suite.

b. Ajout d'un cache

Rédacteur : Iris / Relecteur : Eve – Iris – Julie - Matias

Une fois que l'algorithme jour/nuit garantit que toutes les photographies sont celles du ciel nocturne, qui pourraient donc être intéressantes à exploiter, il faut encore traiter les photographies afin de les rendre exploitables par l'algorithme de machine learning. Pour cela, il faut commencer par établir un cache autour de l'image, et ce pour deux raisons. La première est que la caméra AllSky prend

des photographies circulaires, et qu'une bordure noire entoure donc l'information utile, comme le montre par exemple la figure 12.

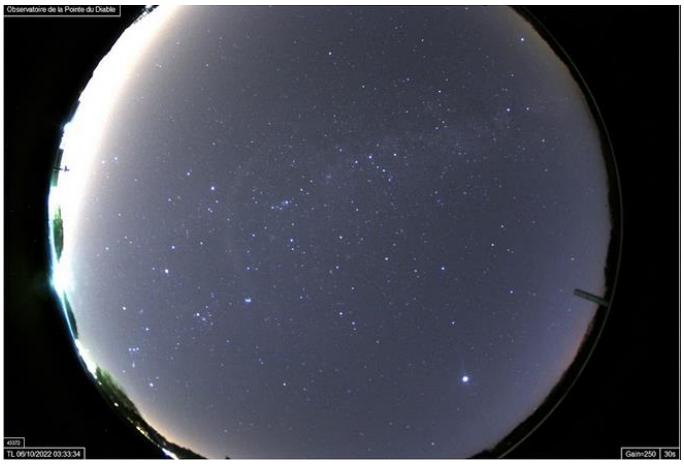


Figure 12 - Image nocturne de la caméra AllSky tirée de notre base de données, référence 2022_06_04_43_38

Afin de simplifier les données d'entrée de notre algorithme, nous allons donc recadrer les bords de l'image pour enlever ce contour noir inutile. La seconde et principale raison de ce masque est la présence de lumières parasites sur le contour de la photo, ici visibles dans la partie en haut à gauche. Ces lumières, visibles sur une majorité des photographies nocturnes, sont liées aux lumières environnants l'observatoire, notamment celles du RAK (Restaurant Associatif de Kernévent), allumées jusqu'à minuit, ou encore celles des lampadaires de la rue voisine. Si elles ne gênent pas l'observation des étoiles du centre de l'image, la présence ou non de cette source lumineuse aurait de grands impacts sur les résultats fournis par l'algorithme de tri, aussi la décision a été prise de créer un cache légèrement trop grand qui masquerait également ces sources lumineuses indésirables.

Le cache a été réalisé grâce au module Python « Pillow » (PIL), permettant la manipulation d'images. Il faut tout d'abord créer une image blanche de la même taille que l'image originale. Ensuite, il faut découper au sein de cette image blanche une ellipse de la dimension souhaitée, ici un cercle de rayon légèrement inférieur à celui de l'image de la caméra AllSky, soit un rapport de rayon de 1,8. Enfin, il faut appliquer ce masque à l'image, comme représenté en figure 13.

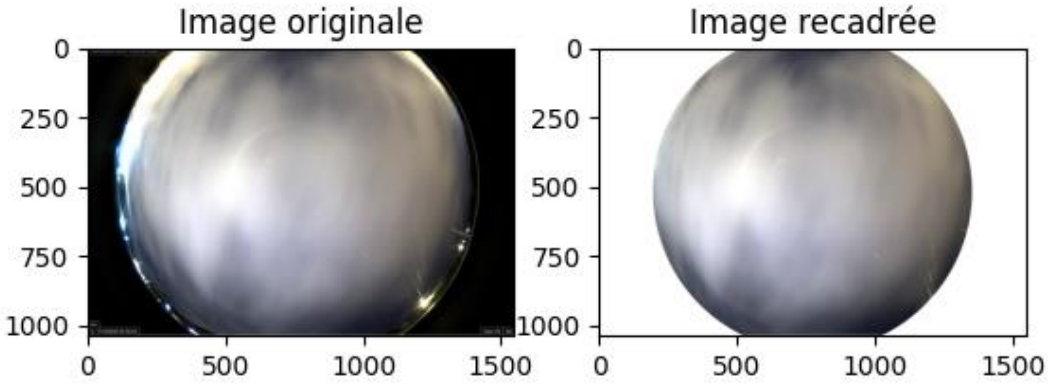


Figure 13 - Exemple avec une photo tirée de la base de données avant et après l'application du masque

c. Traitement en niveaux de gris

Rédacteur : Iris / Relecteur : Eve – Iris – Julie - Matias

Une dernière étape est nécessaire avant que les photographies à trier puissent être traitées par l'algorithme de machine learning. Il s'agit de convertir les images en niveaux de gris. Cela sera plus pratique pour toutes les manipulations futures, notamment pour convertir les images en tableaux de pixels. Les couleurs de l'image ne font pas partie des paramètres essentiels qu'utilisera l'algorithme de machine learning, aussi peut-on alléger l'image en la convertissant en noir et blanc. Enfin, c'est un procédé indispensable pour le fonctionnement d'une des deux tentatives de machine learning de ce projet, celle utilisant les transformées de Fourier en 2D des images, détaillée plus loin dans ce rapport. Les transformées de Fourier des images ne peuvent en effet être calculées que pour des images en noir et blanc [5].

Pour réaliser cette conversion en niveaux de gris, il faut récupérer l'image avec son masque et utiliser la fonction du module PIL "convert()" qui crée une copie transformée de l'image. On a utilisé "convert("L")", qui change le mode de l'image vers un mode contenant 256 nuances de gris. Un exemple de cette conversion peut être visualisé à la figure 14, révélant bien que la suppression des couleurs ne s'accompagne pas d'une perte de données utiles, mais seulement d'un gain de mémoire.

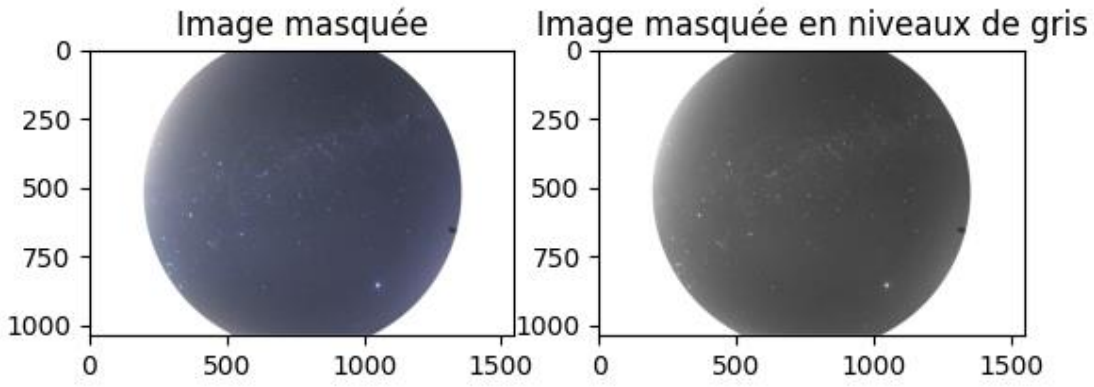


Figure 14 - Image de notre base de données en couleur et en niveaux de gris

Ainsi, les images de la base de données fournie par l'observatoire n'ont pas pu être exploitées telles quelles et ont dû être copiées et modifiées afin d'obtenir un résultat optimal dans notre algorithme de machine learning. Les images de jour sont écartées directement pour limiter la complexité de notre algorithme et ne garder que des images nocturnes similaires dont le seul critère de différenciation est la présence ou non de nuages. Les images restantes sont ensuite converties en noir et blanc et un cache visant à supprimer les bords inexploitables de l'image leur est ajouté. Ces prétraitements accomplis, les images de la base de données sont ainsi prêtes pour servir de base d'entrainement à notre algorithme de machine learning.

Ce premier tri des images permet ainsi d'épargner le passage inutile de 50% des images dans l'algorithme de machine learning. Il répond également au premier critère de discrimination des images mis en place en réponse à la problématique principale. En outre, l'ajout du cache et la conversion des images en niveaux de gris permettra par la suite un meilleur traitement de celles-ci à travers l'algorithme présenté ci-dessous.

IV. Méthodes de machine learning catégorisant les images

Rédacteur : Iris – Matias / Relecteur : Eve – Iris – Julie - Matias

Le machine learning* est un enjeu majeur du projet, ainsi qu'une source de difficulté importante. En effet, cette notion était très peu connue voire inconnue par les membres du groupe au début du projet. De nombreuses méthodes sont utilisées dans ce domaine pour divers usages. Il fallait donc trouver une approche qui convenait à la problématique du projet, tout en étant suffisamment abordable pour être comprise et implémentée.

a. Les méthodes possibles

Rédacteur : Matias / Relecteur : Eve – Iris – Julie - Matias

Il a tout d'abord fallu s'intéresser aux différents algorithmes de machine learning existants. Il existe deux grandes familles de classification d'images : la classification supervisée et non supervisée. Dans le premier cas, les algorithmes sont pré-entraînés sur un ensemble d'images déjà étiquetées, c'est-à-dire des images qui sont déjà associées à des catégories. Par exemple pour un algorithme destiné à reconnaître des images de chats, une grande quantité de photographies associées à l'étiquette « chat » seront fournies pour entraîner le programme à reconnaître des chats. L'algorithme va ensuite extraire des caractéristiques à partir d'une nouvelle image, et les comparer à une liste de caractéristiques présélectionnées pour attribuer une étiquette. Dans la classification d'image non supervisée, les algorithmes de machine learning utilisés ne sont pas pré-entraînés. Autrement dit, ils apprennent par eux-mêmes à analyser et à regrouper les données brutes, en identifiant sans aide humaine les caractéristiques à identifier. Une base de données préétiquetée importante ayant été mis à notre disposition, la méthode de classification supervisée a été retenue.

Plusieurs méthodes de classification supervisées existent pour la catégorisation d'images, notamment les réseaux de neurones conventionnels* (CNN), les machines à vecteur de support* (SVM) et les k plus proches voisins* (k-NN). Suite aux conseils de Monsieur Frédéric Maussang, nous avons préféré éviter de creuser la piste des réseaux convolutifs de neurones, une méthode complexe nécessitant un long temps d'apprentissage et de découverte que la période limitée laissée pour le projet n'aurait pas permis. Parmi les deux méthodes restantes, nous avons opté pour la méthode k-NN, mieux documentée et plus simple à implémenter.

b. Méthode des K voisins

Rédacteur : Matias – Iris / Relecteur : Eve – Iris – Julie - Matias

i. Principe de fonctionnement

Comme évoquée précédemment, la méthode des K plus proches voisins est une méthode assez répandue et facile à comprendre. Pour illustrer son fonctionnement de façon simple, nous pouvons nous appuyer sur la figure 15 issue du site Open Classroom [3] :

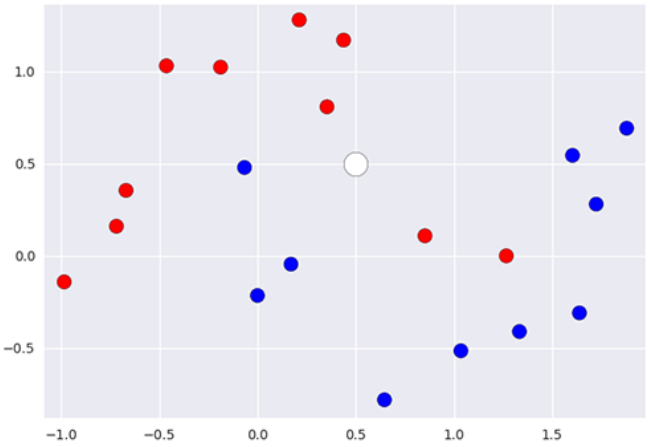


Figure 15 - Illustration de la méthode des K plus proches voisins

Dans cet exemple, les données sont réparties en seulement deux catégories : « rouge » et « bleu ». Lorsqu'une nouvelle donnée doit être triée (représentée par le point blanc sur l'image précédente), l'algorithme va chercher les autres données qui ressemblent le plus à celle qu'on lui demande de classifier. Le « K » du titre de la méthode indique combien d'autres données vont permettre de classifier la nouvelle image. On le qualifie d'hyperparamètre car il ne dépend pas des données à traiter, sa valeur devra être déterminée ultérieurement pour optimiser les performances de l'algorithme. Dans l'exemple précédent, si k vaut 5, alors l'algorithme va déterminer les 5 points les plus proches du nouveau point gris à classer, ce qui donne la figure 16.

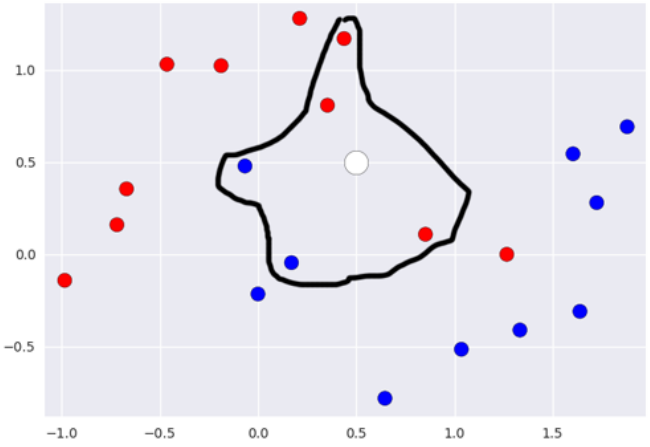


Figure 16 - Suite de l'illustration de la méthode des K plus proches voisins

On remarque sur la figure 16 que parmi les 5 points les plus proches du point blanc, 3 sont rouges et seulement 2 sont bleus. Le nouveau point sera donc logiquement classé dans la catégorie « rouge ». Dans le cadre de notre projet l'idée sera la même en remplaçant les points par des images.

L'un des principaux inconvénients de cette méthode est la nécessité de garder en mémoire toutes les données d'entraînement (celles qui sont déjà classifiées et sur lesquelles l'algorithme se base pour classer les nouvelles données). Ce type d'algorithme est donc difficilement utilisable sur des projets de très grande envergure.

ii. Implémentation de l'algorithme d'entrainement

Après s'être informés sur le principe de fonctionnement, il a été nécessaire d'implémenter notre méthode des k plus proches voisins (voir figures 27 à 29 en partie c. de l'annexe). Tout d'abord, il a fallu déterminer une distance de référence. En effet, l'algorithme des k voisins est basé sur la notion que les échantillons similaires ont tendance à se regrouper dans l'espace des caractéristiques*. Pour prédire l'étiquette d'un nouvel échantillon, l'algorithme cherche les k échantillons les plus proches dans l'espace des caractéristiques et attribue l'échantillon à la classe majoritaire parmi ces k voisins. Cette notion de proximité repose cependant sur celle de norme et de métrique. Il a donc fallu choisir une métrique adaptée à notre problème parmi celles proposées par le module Python « sklearn » que nous avons utilisé pour la partie machine learning.

Par défaut, la métrique utilisée pour mesurer la distance entre les échantillons est la distance euclidienne. Il en existe d'autres comme la distance Manhattan, qui mesure les distances en termes de déplacements verticaux ou horizontaux, ou la distance de Chebyshev, mesurant la distance maximale entre des coordonnées des points. La façon dont sont calculées ces distances n'est pas au cœur du projet mais peut être retrouvée dans cet article d'IBM [4] explicitant les méthodes de calcul des différentes distances. Lors de la phase de test, les trois distances citées plus haut ont été implémentées et testées, mais seule la distance euclidienne s'est exécutée avec un temps relativement faible comparativement aux deux autres, ce qui en a fait la méthode principale utilisée dans la suite des tests.

De plus, le choix du paramètre k, soit le nombre de voisins, influence la performance du modèle. Un k trop petit peut entrainer une sensibilité excessive au bruit, tandis qu'un k trop grand peut entrainer une baisse de précision. Tout comme pour la métrique utilisée, le nombre de voisins a été déterminé par des tests sur notre base de données.

L'algorithme d'entrainement fonctionne donc de la manière suivante : il récupère les photographies triées de la base de données et les retraite avant de les placer dans des dossiers étiquetés sous forme de tableaux « numpy ». Il sépare ensuite aléatoirement en deux notre base de données : 80% des images servent à l'entrainement du modèle, 20% au test. Il crée ensuite notre classifieur k-NN en fonction du k choisi et de la métrique retenue. Ensuite, il prédit les étiquettes de nos images à tester et les compare avec leurs étiquettes réelles en renvoyant le taux de précision, sous forme de pourcentage.

c. Amélioration utilisant la transformée de Fourier des images

Rédacteur : Iris / Relecteur : Eve – Iris – Julie - Matias

La méthode des K voisins nécessite la recherche de paramètres communs. En effet, il est nécessaire de trouver des points de ressemblance et de différence entre les images, ici selon leur couverture nuageuse. Pour une image exploitable, donc catégorisée comme dégagée, l'algorithme doit détecter les points lumineux que sont les étoiles, absents et remplacés par des aplats de gris que sont les nuages pour les photographies couvertes. En partant de ce postulat, nous avons pensé à l'idée de ne pas appliquer notre algorithme directement sur nos images mais sur la transformée de Fourier* en 2D des photographies. Cette idée est venue suite à la SAR optique sur l'optique de Fourier, qui a fourni une grande partie de la documentation de ce projet en lien avec les transformées de Fourier [5]. Pour une image en noir et blanc, qui est un signal en deux dimensions, les surfaces homogènes et importantes sont converties en basses fréquences, visibles en bordure des images représentant les transformées de Fourier. Au contraire, les mouvements brusques d'intensité lumineuses, à l'instar des points lumineux, ou les contours d'objet, sont représentées dans les hautes fréquences en centre d'image.

On pourrait ainsi s'attendre à observer une différence entre les images de ciel dégagé, contenant des points lumineux visibles, et les images de ciel couvert, contenant de grandes surfaces grisées homogènes. Il a donc fallu soumettre nos photographies à un prétraitement, et entrainer l'algorithme sur leurs transformées de Fourier en espérant obtenir un meilleur taux de précision dans la classification.

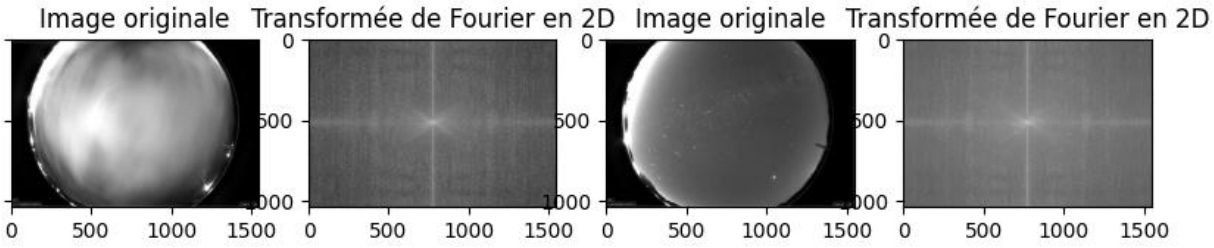


Figure 17 - Résultat de l'algorithme transformant deux images de notre base de données en leur transformée de Fourier

d. Algorithme de tri final

Rédacteur : Iris / Relecteur : Eve – Iris – Julie - Matias

Une fois notre algorithme de classification choisi comme étant celui avec le plus haut taux de précision parmi ceux développés, il faut implémenter l'algorithme permettant de prendre en entrée le répertoire du jour et permettant d'associer un tag aux photographies qu'il contient.

Pour cela, il s'est révélé indispensable de créer une fonction nommée predict_image_label qui prend en entrée une image et notre modèle k-NN. Cette fonction prétraite notre image (en effectuant les mêmes prétraitements que ceux ayant servis à entrainer notre algorithme), puis lui associe une

étiquette de prédiction. On aboutit alors au code présenté dans les figures 30 à 33 en partie d. de l'annexe.

L'algorithme de machine learning étant programmé, il reste à l'implémenter au sein de l'observatoire astronomique, dans le but de répondre pleinement à la demande du Client. Néanmoins, avant cela il faut encore tester le livrable avant son rendu au Client. Pour cela des diverses étapes de tests et de validation ont été effectuées.

V. Implémentation, tests et validation du livrable

a. Implémentation dans l'observatoire

Rédacteur : Julie / Relecteur : Eve – Iris – Julie – Matias

Suite à des concertations avec le Client, le besoin initial d'exploiter un jour ou un mois complet a été restreint à l'analyse d'une journée. En effet, la caméra prend environ 1000 photographies par jour et un volume d'un mois représenterait un temps de traitement non acceptable pour l'utilisateur, au vu de la complexité en temps de l'algorithme.

i. Input

Le client et le groupe ont défini qu'il faudrait rentrer en début du code le chemin des images à traiter. Ce chemin devra arriver directement sur le plus petit dossier contenant les photographies donc le dossier d'une journée. Une piste d'interaction avec l'utilisateur aurait pu être l'ouverture en début d'exécution du programme d'une bulle de commande permettant d'aller chercher le dossier concerné et de récupérer ainsi le chemin du dossier à traiter. Cependant ceci relève plus de l'interface homme machine et ne fait pas partie intégrante du projet.

En outre, les dossiers que l'algorithme analysera seront présents en propre dans l'ordinateur. Il n'utilisera pas la version drive des fichiers.

ii. Output

En sortie de l'algorithme, le Client souhaite avoir sous fichier « .csv » la liste des photographies, et associer à chacune d'entre elles sa catégorie d'appartenance sous forme d'un tag. Le fichier « .csv » final contient donc :

- Pour une image prise de jour : Nom_de_l'image ; jour et un retour à la ligne.
- Pour une image prise de nuit : Nom_de_l'image; nuit; catégorie_de_ciel et un retour à la ligne.

Ce format a été choisi car il permet de voir les photographies triées sur n'importe quel système d'exploitation, et se révèle particulièrement visuel lorsque l'on utilise des tableurs. On peut ainsi

facilement vérifier si l'algorithme n'a pas fait d'erreur et faire soi-même les suppressions et sauvegardes avec le tableur ou l'outil de gestion des dossiers.

b. Tests et validation

Rédacteur : Eve – Iris / Relecteur : Eve – Iris – Julie – Matias

Finalement, dans l'optique de vérifier les diverses parties du livrable des tests ont été effectués. Toutes les parties ont tout d'abord été testées indépendamment les unes des autres. Enfin, un dernier test a été accompli avant le rendu du livrable afin de vérifier que ce dernier fonctionne dans son entièreté.

i. Tri jour/nuit

Dans un premier temps, dans le cas du tri jour/nuit chaque phase a été testée individuellement. La première étape n'a rencontré aucun problème particulier. En effet, elle retournait bien la liste de tous les fichiers contenus dans un dossier dont le chemin lui avait été précédemment spécifié. La phase suivante a, quant à elle, nécessité d'être reprise une fois. En effet, la fonction changeant le format ne fonctionnait pas, cela était dû à la partie « .jpg » du nom du fichier, qui n'était pas prise en compte auparavant. Une simple modification et l'ajout d'un « split(".") » a permis de régler le problème, comme le montre la figure 17.

```
def splitDate(date_string):

    parts3 = date_string.split(".") # Partie corrigée

    parts = parts3[0].split("__")
    date = parts[0]
    heure = parts[1]

    parts1 = date.split("_")
    year = int(parts1[0])
    month = int(parts1[1])
    day = int(parts1[2])

    parts2 = heure.split("_")
    hour = int(parts2[0])
    minutes = int(parts2[1])
    seconds = int(parts2[2])

    return datetime(year, month, day, hour, minutes, seconds)
```

Figure 18 - Présentation de la fonction corrigée permettant de changer le nom des fichiers au format "datetime"

L'étape suivante n'a pas eu besoin d'être reprise. Pour une liste donnée de fichiers au format « datetime », la fonction retournait bien deux listes distinctes. La seule modification apportée n'a pas été directement dans cette fonction mais dans celle qu'elle utilise. En effet, comme expliqué précédemment dans la partie prétraitement au niveau de la sous-partie du tri jour/nuit, la fonction permettant de calculer l'heure de ce que nous appelons ici « nuit » a subi une modification. La nuit astronomique commençait et finissait quand le Soleil se situait à 18° en-dessous de l'horizon. Néanmoins, cela ne représentait pas les heures limites déterminées avec le Client. C'est pourquoi, afin de modifier cet algorithme de nombreux tests ont été effectués. Nous avons fait tourner cet algorithme avec une date et une heure précise à laquelle nous voulions que l'algorithme considère comme la limite jour/nuit, le 1er octobre à 20h40. A chaque test, le paramètre établissant la nuit

astronomique était modifié afin d'atteindre les attentes du Client. Après une vingtaine d'essais, l'optimisation du paramètre a permis de l'établir à 26°.

Enfin, l'étape finale à tester était la copie des fichiers dans les bons dossiers indiqués, « Jour » et « Nuit ». Aucun problème n'a été détecté après avoir indiqué le chemin vers les dossiers voulus.

Ces tests ont permis de valider le critère de tri jour/nuit. A l'issu des différentes phases de test les photographies ont bien été classées dans deux dossiers différents et respectent le critère d'une heure et demie après le coucher du Soleil et avant le lever de ce dernier.

ii. Cache et conversion en niveaux de gris

Les étapes de l'application du cache et de la conversion en niveaux de gris ont nécessité quelques reprises en raison de leur impact sur la modification de l'image. Une modification que l'on ne souhaite que temporaire pour ne pas porter atteinte à l'intégrité de notre base de données d'images qui ne nous appartient pas, et est la propriété de l'observatoire. Ainsi, il a fallu commencer par la création d'une copie de l'image, et c'est sur cette copie que sont effectués les traitements nécessaires au tri.

Le cache sur mesure de forme elliptique a été implémenté manuellement, en modifiant les paramètres sur plusieurs images en vue d'éliminer le moins possible de partie étoilée mais de tout de même supprimer l'entièreté des lumières parasites. De plus, il a été requis de prendre garde au fait que le masque ainsi créé n'est qu'un cache, et ne fait pas partie intégrante de notre image. C'est pourquoi, il a été nécessaire de créer encore une nouvelle image fusionnant directement avec le cache avant de passer à la partie suivante de l'implémentation. La conversion en niveaux de gris n'a, quant à elle, pas posé de souci lors de la réalisation ni de ses tests.

iii. Machine learning

De nombreuses modifications et améliorations ont été effectuées sur l'algorithme de machine learning, fonctionnant dans un processus itératif dans le but de trouver les meilleurs paramètres permettant d'obtenir une classification la plus juste possible. Des premiers tests ont été effectués sur une base de données restreinte à moins de 4000 images et contenant des erreurs dans leur classification, les taux de précision ne dépassaient alors pas les 70%. Lors de l'entrainement de l'algorithme sur la base de données complète, nous avons séparé en deux jeux de données notre base de données initiale : 80% des images ont servi à entrainer l'algorithme (soit 12800 photographies), et 20% ont servi au test afin de vérifier sa précision (soit 3200 photographies). Pour mesurer le taux de précision de l'algorithme, il a été requis de calculer le pourcentage de ressemblance dans deux listes, l'une contenant les étiquettes donc les véritables catégories de nos images tests, et l'autre contenant les étiquettes prédites par le modèle.

En utilisant la méthode des K voisins avec la distance euclidienne sur les images, sans le prétraitement des photographies en noir et blanc et masquées, on peut alors comparer les performances de notre modèle pour différentes valeurs de k. On a fait tourner le programme cinq fois pour différentes valeurs et compilé les moyennes des résultats dans le tableau Figure 19. Il ressort un taux de précision au mieux de 82,58%.

Cela signifie qu'une nouvelle image pourra être classée dans la bonne catégorie avec plus de 82% de fiabilité en moyenne.

Nombre de voisins k	1	5	6	7	10	20	50	100
Précision de l'algorithme pour la distance euclidienne	79,73	80,13	81,67	80,22	81,91	82,58	79,43	78,65

Figure 19 - Tableau synthétisant les résultats de notre algorithme des k voisins pour différents paramètres k et pour la métrique euclidienne. Présente les pourcentages de précision de l'algorithme pour une moyenne sur 5 tests.

Nous avons également tenté de faire tourner notre algorithme avec la distance de Manhattan mais les temps d'exécution étaient tels que le programme n'a pas pu se terminer.

Il a également été possible de développer un code utilisant les réseaux convolutifs de neurones pour classifier les images de l'observatoire (voir partie E de l'annexe). Ses taux de précisions sont de l'ordre de 80% également, néanmoins la méthode retenue n'est pas celle-ci car elle n'a pu être exploitée que lors d'une courte durée.

Ainsi, l'algorithme retenu est celui utilisant la méthode des k voisins, avec pour paramètres la distance euclidienne et 20 voisins. Ce programme conçu répond en grande partie aux demandes du client en triant statistiquement correctement 82% des images. Cela reste cependant améliorable, notamment en utilisant des méthodes de réseaux convolutifs plus poussés.

Le livrable a ainsi pu être validé dans sa globalité grâce aux tests précédents. Ils permettent donc la validation globale du livrable, bien que quelques sources d'améliorations subsistent. Les critères de tri choisis lors du cahier des charges avec le Client sont respectés, le projet a donc abouti.

VI. Conclusion

Rédacteur : Eve / Relecteur : Eve – Iris – Julie – Matias

Les principaux objectifs du projet ont été atteints. En effet, la partie concernant le tri jour/nuit est fonctionnelle. Chaque photographie se voit bien attribuer une catégorie et les demandes du Client sont respectées : la nuit ne concerne que les photographies prises à partir d'une heure et demie après le coucher du Soleil jusqu'à une heure et demie avant le lever du Soleil. Le cache utilisé pour améliorer le machine learning est également abouti. Les photographies traitées ne possèdent plus le contour lumineux dû aux lumières du RAK (Restaurant Associatif de Kernévent) et à la ville de Brest. Les photographies ont par la suite été converties en niveaux de gris. Cette étape est également opérationnelle comme l'a montré la figure 14. Enfin, la partie concernant le machine learning est efficace. En effet, l'algorithme final permet d'atteindre 80% de réussite sur les images prétraitées, c'est-à-dire que 80% des photographies sont classées dans les bonnes catégories. Ainsi, les résultats

permettent bien de répondre à la problématique exposée dans l'introduction. Le livrable final permet bien de trier les images stockées à l'observatoire. De plus, les catégories qui ont été choisies ont bien permis de discriminer certaines photographies en fonction des conditions atmosphériques. Le projet offre alors la possibilité de remplacer un traitement manuel des images prises par la caméra Allsky, par une classification automatique de ces mêmes images au sein d'un classeur Excel.

Néanmoins, bien que les objectifs aient été atteints, certaines pistes d'améliorations subsistent à court ou moyen terme. Le taux de réussite pourrait notamment être amélioré en augmentant la base de données ou en changeant de méthode de machine learning. 80% de précision est un résultat acceptable et encourageant, mais améliorer ce score est pertinent pour limiter les erreurs ainsi que les corrections manuelles qui peuvent s'avérer pénibles à réaliser. En augmentant la taille de la base de données, une amélioration avec la méthode utilisée a été remarquée. En effet, pendant les tests intermédiaires, sur moins de 4000 images la précision ne dépassait pas 70%. D'un autre côté, sur la base de données avec les images d'un mois complet (16 000 images de nuit environ) la précision est montée à plus de 80% comme expliqué précédemment. Une piste d'exploration serait donc de posséder une plus grande base de données étiquetées. Un autre point d'amélioration serait de chercher à utiliser une autre méthode de machine learning peut-être plus complexe, dans le cas où la méthode des k plus proches voisins, utilisée dans le cadre de ce projet se révélait insuffisante malgré les optimisations apportées. Cependant, ces perspectives n'étaient pas envisageables dans le cadre du projet car la base de données utilisées a été triée manuellement et les méthodes de machine learning demandent un temps d'apprentissage conséquent. Enfin, il serait préférable d'augmenter le volume d'images traitées en un temps limité. Pour le moment, ce projet ne permet de traiter que quelques mois de données dans un délai raisonnable. Il faudrait donc diminuer la complexité en temps de l'algorithme afin d'améliorer la capacité de traitement de l'algorithme.

Enfin, le projet global tel qu'il est aujourd'hui a pour vocation d'être poursuivi dans les années qui suivent. Comme évoqué dans l'introduction, le projet détaillé dans ce rapport technique s'intègre dans une étude plus large améliorant l'expérience utilisateur au sein de l'observatoire de la Pointe du Diable. L'implémentation de cette partie permettra de résoudre le problème de stockage de photographies sur le long terme. Néanmoins, de nouvelles implémentations restent à être développées. Par exemple, il reste à créer un algorithme distinguant et nommant les planètes. Celui-ci avait été commencé l'année dernière mais n'avait pas abouti. Enfin, une information en temps réelle sur le type de nuages pourrait être proposée en vue d'une prochaine étude en lien avec l'observatoire.

VII. Bibliographie

[1] LI Xiaotong, WANG Baozhu, QIU Bo, WU Chao. « An all-sky camera image classification method using cloud cover features », *Atmospheric Measurement Techniques,* Vol. 15, 16 juin 2022, p.3629–3639.

[2] OPENCLASSROOMS, *Soleil Calcul du lever et coucher du Soleil* [en ligne]. Disponible sur : <https://openclassrooms.com/forum/sujet/soleil-2> (consulté le 05.04.2023).

[3] OPENCLASSROOMS, *Entraînez votre premier k-NN* [en ligne]. Disponible sur : <Entraînez votre premier k-NN - Initiez-vous au Machine Learning - OpenClassrooms> (consulté le 30.03.2023).

[4] IMB, *Qu'est-ce que l'algorithme des k plus proches voisins ?* [en ligne]. Disponible sur : < https://www.ibm.com/fr-fr/topics/knn > (consulté le 22.05.2023)

[5] IMT Atlantique, Cours de Sar Optique de Fourier, polycopié 2023 PolySAR_B_CoursTP2023.pdf (consulté le 13.04.2023 et le 20.05.2023) .

VIII. Glossaire

Caméra plein ciel ou caméra Allsky : Caméra permettant d'avoir une vue du ciel (champ de vision 180°x130°) via une projection sphérique.

Cloud : Le terme « cloud » désigne les serveurs accessibles sur Internet, ainsi que les logiciels et bases de données qui fonctionnent sur ces serveurs. Les serveurs situés dans le cloud sont hébergés au sein de datacenters répartis dans le monde entier. L'utilisation du cloud computing (informatique cloud) permet aux utilisateurs et aux entreprises de s'affranchir de la nécessité de gérer des serveurs physiques eux-mêmes ou d'exécuter des applications logicielles sur leurs propres équipements.

Ephémérides : Table donnant de jour en jour, ou pour d'autres intervalles de temps, les valeurs calculées de diverses grandeurs astronomiques variables.

Espace des caractéristiques : Dans le contexte de la classification d'images, cela fait référence à l'ensemble des variables ou attributs utilisés pour représenter chaque image. Ces caractéristiques peuvent inclure des informations telles que la valeur des pixels, les histogrammes de couleurs, les textures, les formes...

Etiquetage des données (ou data labeling en anglais) : consiste à attribuer des informations à divers points de données afin que les algorithmes de machine learning puissent mieux en comprendre la signification.

Machines à vecteur de support (SVM) : technique d'apprentissage supervisé pour résoudre des problèmes de régression et de discrimination qui établit une frontière de séparation entre les échantillons.

Machine learning : L'apprentissage automatique (machine learning en anglais) est un champ d'étude de l'intelligence artificielle qui vise à donner aux machines la capacité d'« apprendre » à partir de données, via des modèles mathématiques. Plus précisément, il s'agit du procédé par lequel les informations pertinentes sont tirées d'un ensemble de données d'entraînement.

Méthode des K plus porches Voisins (k-NN) : technique d'apprentissage supervisé calculant la distance entre des échantillons selon une norme établie.

NAS : L'acronyme NAS, pour Network Attached Storage, désigne un périphérique de stockage utilisé pour le stockage et le partage de fichiers via un réseau (Ethernet ou de type WAN la plupart du temps). Il s'agit d'un serveur de fichiers capable de fonctionner de façon autonome. On le résume parfois à un disque dur relié à un réseau (privé, professionnel, etc.). Il peut être traduit en français par serveur de stockage en réseau, ou stockage raccordé en réseau. Documents, images ou vidéos peuvent être servis depuis un NAS sur les terminaux connectés à son réseau. Des NAS peuvent également être utilisés comme un serveur web.

Nuit astronomique : Pour définir la nuit astronomique, on utilise la référence de 18° sous l'horizon car ce n'est qu'en dessous de ce seuil que les étoiles de faible magnitude apparente peuvent être observées.

Réseaux de neurones convolutif (CNN) : méthode d'apprentissage probabiliste dont la conception est inspirée des neurones du cerveau humain.

Transformée de Fourier : C'est une opération qui transforme une fonction intégrable en une autre fonction, décrivant le spectre en fréquences de *f*. Dans le cas de la transformée de Fourier en 2D, une image est un signal 2D présentant des variations d'intensité lumineuse dans l'espace. On calcule sa transformée de Fourier avec les coordonnées cartésiennes en échantillonnant l'image. De ce fait, les basses fréquences traduisent les surfaces grandes et homogènes, tandis que les hautes fréquences représentent les contours, variations brusques d'intensité ainsi que le bruit de l'image.

IX. Annexes

a. Comparaison entre le planning réel et le planning anticipé

Rédacteur : Matias / relecteur : Eve – Julie – Iris – Matias

Planning prévisionnel :

Codev Planning

9 mars 2023

Diagramme de Gantt

5

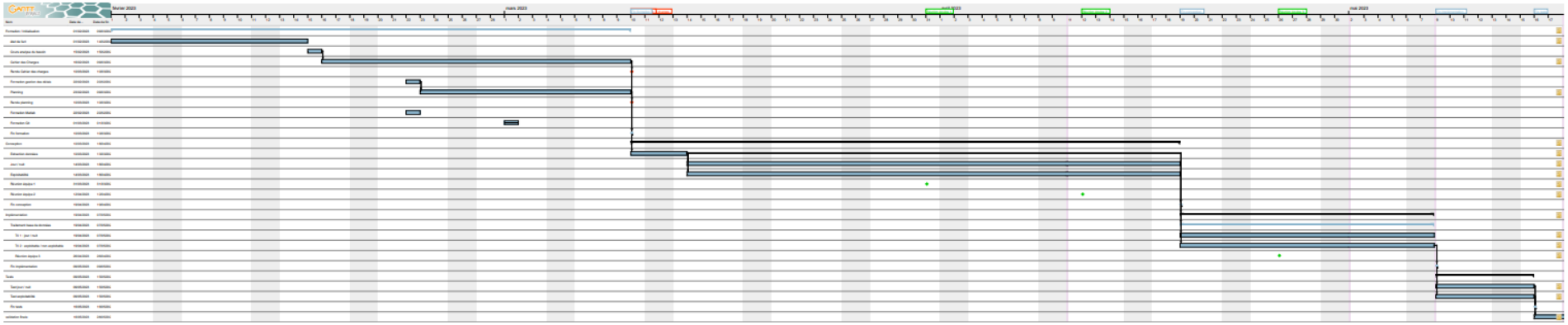
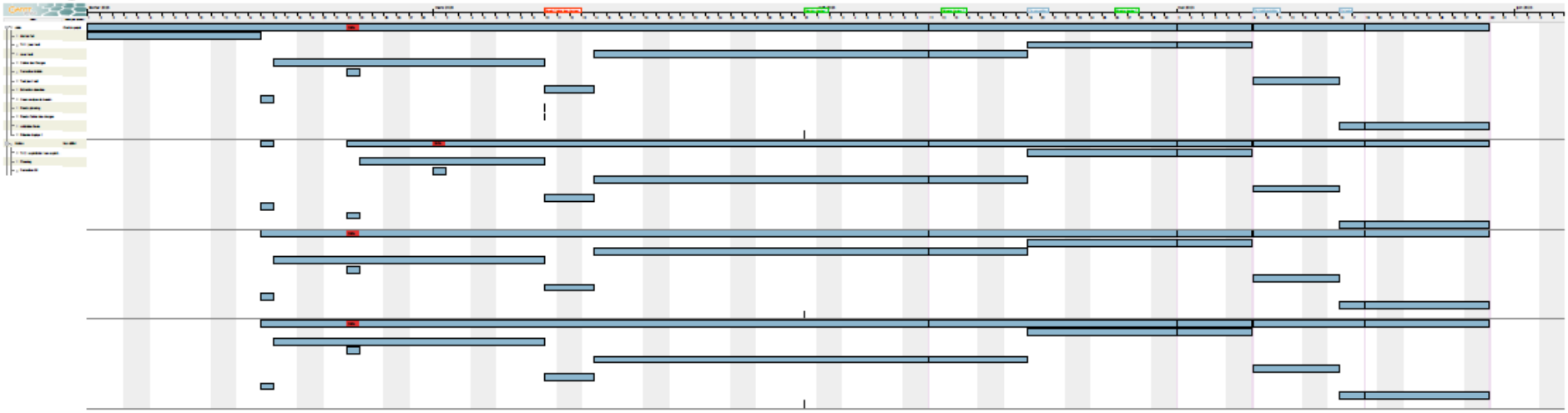
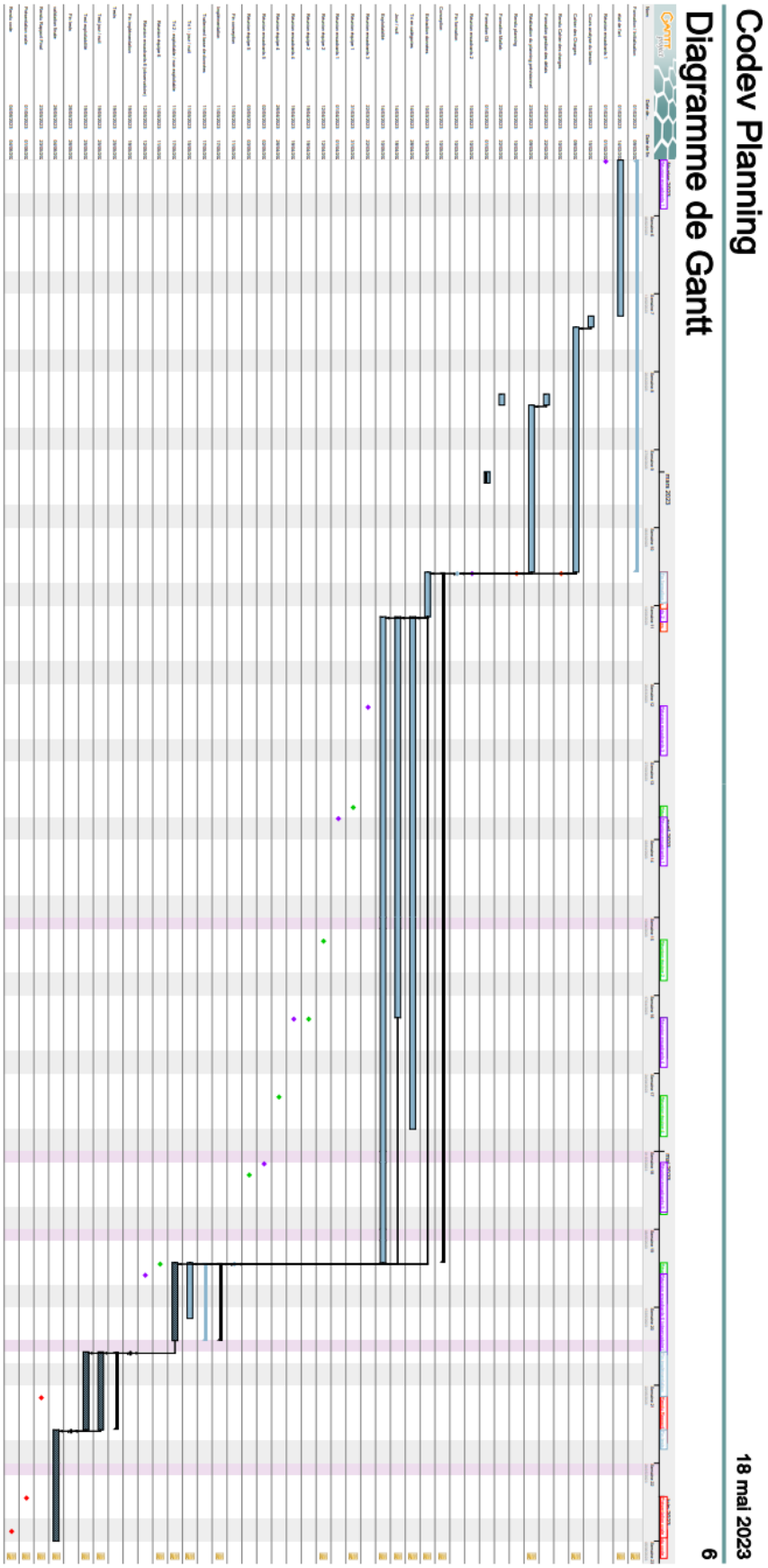


Diagramme des Ressources



Planning réel effectivement suivi :

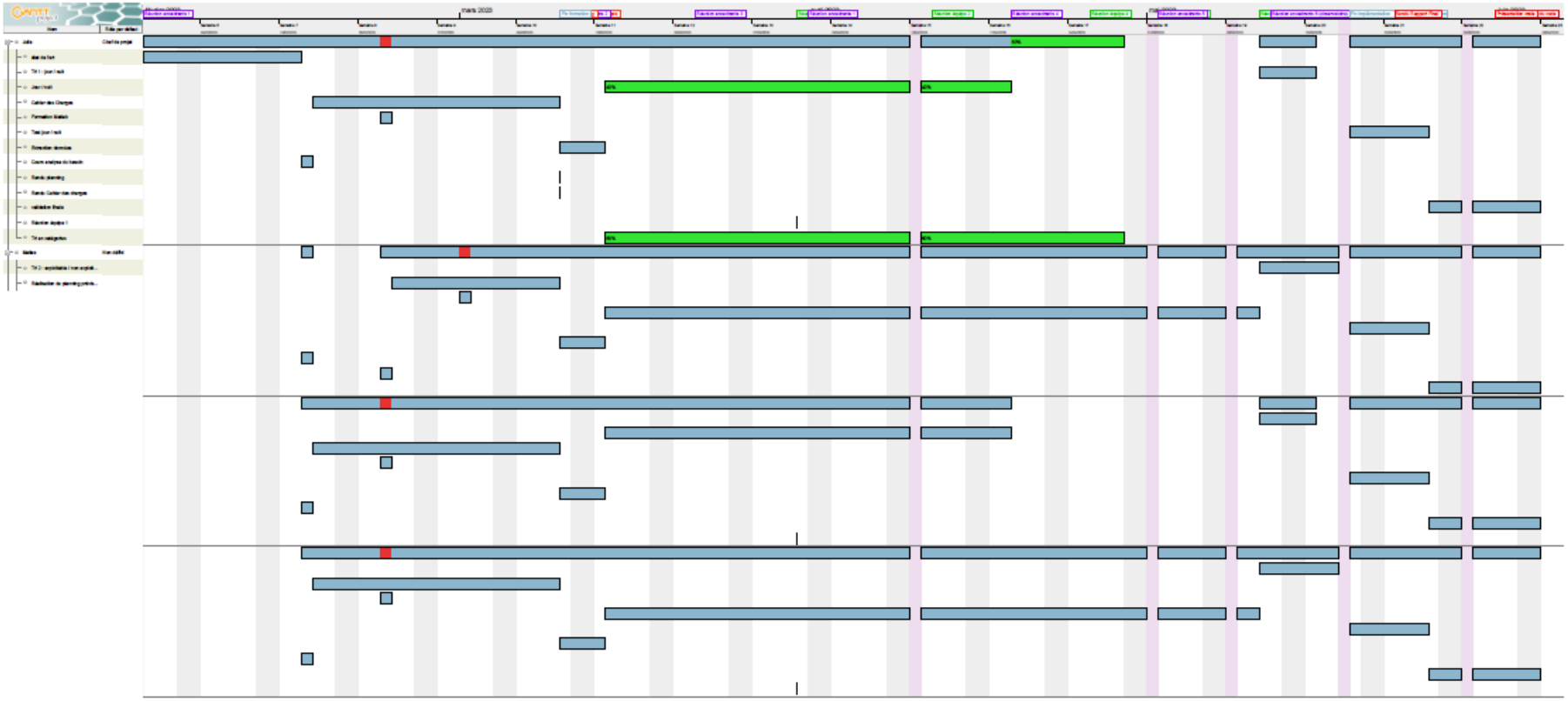


Codev Planning

18 mai 2023

Diagramme des Ressources

7



En comparant le planning initialement proposé avec celui effectivement suivi, on remarque tout d'abord que plus de réunions d'équipe ont été effectuées que dans la prévision. En effet, le groupe s'est réuni quasiment tous les mercredis sur les créneaux réservés au CODEVSI. De plus, les réunions avec les encadrants ont été ajoutées sur la version finale. Elles avaient été omises lors de la première version car les dates ont été décidées au fil de la réalisation du projet.

Par la suite, nous avons constaté une période de conception plus longue qu'anticipée, notamment concernant la partie machine learning qui a nécessité des recherches et une période d'apprentissage conséquentes. En outre, la phase d'implémentation s'est avérée plus courte car une bonne partie s'est faite en même temps que les recherches, permettant ainsi de déterminer si une méthode était utilisable ou non. Une tâche qui n'avait pas été suffisamment prise en compte au début du projet et au moment de la conception du premier planning, a été la longue période de traitement des données en catégories. Ce traitement était nécessaire afin que la base de données, désormais étiquetées selon les catégories définies dans le cahier des charges, puisse servir de base d'entraînement aux algorithmes de machine learning. Ce travail demandait de trier manuellement 33 448 images en diverses catégories, ce qui a pris une grande partie du temps.

En plus de cela, nous avons été confrontés à un problème technique qui a entraîné la perte temporaire du travail de tri réalisé. Ce contretemps n'a pas eu énormément d'impact sur les premières expérimentations car nous avions la possibilité de travailler à plus petite échelle sur une base de données moins grande. Cependant les conséquences ont été plus grandes concernant le début des premières phases de tests finaux qui ont dû être retardées. En effet, nous avions besoin d'une base de données suffisamment fournie pour espérer avoir des résultats fiables.

Enfin, nous avons adapté la gestion du temps sur la fin du projet en nous adaptant au décalage entre la date de rendu du rapport final et la date de rendu du code fixée par le client. En effet, cette différence nous permet de finaliser les tests et d'apporter quelques améliorations supplémentaires au code après le rendu de ce rapport.

b. Algorithme de tri jour/nuit

```
from datetime import datetime
import math


## Première étape : récupération du nom des fichiers

import os

directory = "C:/Users/33624/OneDrive/Bureau/IMT/CODEVSI" #
Spécifiez le chemin d'accès au répertoire contenant les fichiers
attention à bien mettre / et pas \

files = os.listdir(directory) # Utilisez la méthode listdir() de
la bibliothèque os pour récupérer les noms des fichiers dans le
répertoire

# Parcourez la liste des fichiers et affichez les noms de fichier,
création d'une liste contenant le nom des dossiers

f=[]
def NomFichiers(files):
    for file in files:
        f.append(file)
    return f
```

Figure 20 - Etape 1 du tri jour/nuit

```
## Deuxième étape : Transformation des noms pour un nouveau format
datetime

def splitDate(date_string):

    parts3 = date_string.split(".") # Partie corrigée

    parts = parts3[0].split("__")
    date = parts[0]
    heure = parts[1]

    parts1 = date.split("_")
    year = int(parts1[0])
    month = int(parts1[1])
    day = int(parts1[2])

    parts2 = heure.split("_")
    hour = int(parts2[0])
    minutes = int(parts2[1])
    seconds = int(parts2[2])

    return datetime(year, month, day, hour, minutes, seconds)


LWhen=[]
def listWhen(f):
    for file in f :
        LWhen.append(splitDate(file))
    return LWhen
```

Figure 21 - Etape 2 du tri jour/nuit

```
## Troisième étape : tri des images en deux listes distinctes,
récupère le nom de l'image


Jour=[]
Nuit=[]
def tri(LWhen):
    i=0
    for j in LWhen :
        if is_daytime(latitude, longitude, j, twilight)==True:
            Jour.append(f[i])
        if is_daytime(latitude, longitude, j, twilight)==False:
            Nuit.append(f[i])
        i+=1
    return Jour, Nuit
```

Figure 22 - Etape 3 du tri jour/nuit

```
## Quatrième étape : Utilisation des heures de capture d'image
pour savoir s'il fait jour (True) ou nuit (False), en prenant en
compte la nuit comme environ 1h30 après le coucher du Soleil

latitude = 48.3600473
longitude = -4.5705254
twilight = 'astronomical'
#when = splitDate("2022_10_05__20_23_40") #when=datetime(2022, 10,
5, 20, 23, 40)

def is_daytime(latitude, longitude, when, twilight):

    day = when.toordinal() - 693594
    t = when.time()
    time = (t.hour + t.minute / 60.0 + t.second / 3600.0) / 24.0

    # calculate julian day and century:
    jd = day + 2415018.5 + time
    jc = (jd - 2451545.0) / 36525.0

    # siderial time at greenwich
    gstime = (280.46061837 + 360.98564736629 * (jd - 2451545.0) +
(0.0003879331 - jc / 38710000) * jc ** 2) % 360.0

    # geometric mean longitude sun (deg)
    l0 = (280.46646 + jc * (36000.76983 + jc * 0.0003032)) % 360

    # geometric mean anomaly sun (radians)
    m = math.radians(357.52911 + jc * (35999.05029 - 0.0001537 *
jc))
```

Figure 23 - Etape 4 du tri jour/nuit 1/3

```
    # calculate elements used in multiple places below:
    omega = math.radians(125.04 - 1934.136 * jc)
    latitude = math.radians(latitude)

    # mean obliquity of ecliptic, corrected (radians)
    seconds = 21.448 - jc * (46.815 + jc * (0.00059 - jc *
0.001813))
    e0 = 23.0 + (26.0 + seconds / 60.0) / 60.0
    e = math.radians(e0 + 0.00256 * math.cos(omega))

    # sun true longitude (deg)
    o = l0 + c

    # sun apparent longitude (radians)
    lambda_ = math.radians(o - 0.00569 - 0.00478 *
math.sin(omega))

    # sun declination (radians)
    declination = math.asin(math.sin(e) * math.sin(lambda_))

    # sun right ascension (deg)
    rightasc = math.degrees(math.atan2(math.cos(e) *
math.sin(lambda_), math.cos(lambda_)))

    elevation = math.degrees(math.asin(
        math.sin(latitude) * math.sin(declination) +
        math.cos(latitude) * math.cos(declination) *
        math.cos(math.radians(gstime + longitude - rightasc))))
```

Figure 24 - Etape 4 du tri jour/nuit 2/3

```
    # - Solar diameter gives 0.833 degrees - rim of sun appears
before centre of disc.
    # - For civil twilight, allow 6 degrees.
    # - For nautical twilight, allow 12 degrees.
    # - For astronomical twilight, allow 18 degrees.
    if twilight is None:
        limit = -0.8333
    elif twilight == 'civil':
        limit = -6.0
    elif twilight == 'nautical':
        limit = -12.0
    elif twilight == 'astronomical':
        limit = -26.0 #Correspond à peu près à 1h30 après le
coucher du Soleil et avant le lever du Soleil
    else:
        raise ValueError('is_day() twilight argument must be one
of: civil, nautical, astronomical or None.')

    #def is_daytime(elevation, limit):
    return bool(elevation > limit)
    #true = day, false = night

print(is_daytime(latitude,longitude, when, twilight))
```

Figure 25 - Etape 4 du tri jour/nuit 3/3

```
## Cinquième étape : ranger les fichiers dans un dossier sur
l'ordinateur

import shutil

# Créer le dossier où vous souhaitez ranger les fichiers
dossier_destination_Jour = "C:/Users/jurol/Documents/CODEV/
test_jn/jour"
dossier_destination_Nuit = "C:/Users/jurol/Documents/CODEV/
test_jn/nuit"

if not os.path.exists(dossier_destination_Jour):
    os.makedirs(dossier_destination_Jour)
if not os.path.exists(dossier_destination_Nuit):
    os.makedirs(dossier_destination_Nuit)

# Boucle à travers chaque fichier et les déplacer vers le dossier
de destination
for fichier_jour in Jour:
    chemin_fichier_jour = directory+"/"+fichier_jour
    print(chemin_fichier_jour)
    shutil.move(chemin_fichier_jour, dossier_destination_Jour)

for fichier_nuit in Nuit:
    chemin_fichier_nuit = directory+"/"+fichier_nuit
    print(chemin_fichier_nuit)
    shutil.move(chemin_fichier_nuit, dossier_destination_Nuit)
```

Figure 26 - Etape 5 du tri jour/nuit

c. Aperçu du code K-NN

```
1
2   import os
3   import cv2
4   import numpy as np
5   from sklearn.model_selection import train_test_split
6   from sklearn.neighbors import KNeighborsClassifier
7   from sklearn.metrics import accuracy_score
8
9
10  # Définition des chemins vers les dossiers contenant les images
11  degage2_dir = 'bdd/degage2'
12  couverture_partielle2_dir = 'bdd/couverture_partielle2'
13  couverture_totale2_dir = 'bdd/couverture_totale2'
14
15  # Chargement des images et création des étiquettes correspondantes
16  images = []
17  labels = []
```

Figure 27 - Code algorithme K-NN 1/3

```python
# Chargement des images de 'degage2'
for filename in os.listdir(degage2_dir):
    img = cv2.imread(os.path.join(degage2_dir, filename))
    img = cv2.resize(img, (150, 150))  # Redimensionner les images si nécessaire
    images.append(img)
    labels.append('degage')

# Chargement des images de 'couverture_partielle2'
for filename in os.listdir(couverture_partielle2_dir):
    img = cv2.imread(os.path.join(couverture_partielle2_dir, filename))
    img = cv2.resize(img, (150, 150))  # Redimensionner les images si nécessaire
    images.append(img)
    labels.append('couverture_partielle')

# Chargement des images de 'couverture_totale2'
for filename in os.listdir(couverture_totale2_dir):
    img = cv2.imread(os.path.join(couverture_totale2_dir, filename))
    img = cv2.resize(img, (150, 150))  # Redimensionner les images si nécessaire
    images.append(img)
    labels.append('couverture_totale')

# Conversion des listes en tableaux numpy
images = np.array(images)
labels = np.array(labels)
```

Figure 28 - Code algorithme K-NN 2/3

```python
# Séparation des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)

# Aplatir les images en vecteurs
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

# Création du classifieur k-NN et ajustement sur les données d'entraînement
k = 10  # Nombre de voisins
knn = KNeighborsClassifier(n_neighbors=k, metric='chebyshev')
knn.fit(X_train, y_train)

# Prédiction sur les données de test
y_pred = knn.predict(X_test)

# Calcul de l'exactitude (accuracy) du modèle
accuracy = accuracy_score(y_test, y_pred)
print("Exactitude : {:.2f}%".format(accuracy * 100))

```

Figure 29 - Code algorithme K-NN 3/3

d. K-NN avec les prétraitements (cache, noir et blanc…)

```python
import os
import cv2
import numpy as np
from PIL import Image, ImageDraw, ImageOps
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Définition des chemins vers les dossiers contenant les images
degage2_dir = 'bdd/degage2'
couverture_partielle2_dir = 'bdd/couverture_partielle2'
couverture_totale2_dir = 'bdd/couverture_totale2'

# Chargement des images et création des étiquettes correspondantes
images = []
labels = []
```

Figure 30 - Code algorithme K-NN amélioré 1/4

```python
# Fonction d'application du masque
def apply_circular_mask(image):
    # Récupérer la taille de l'image
    width, height = image.size

    # Calculer le rayon du cercle à dessiner
    radius = min(width, height) // 2

    # Créer une nouvelle image blanche avec un canal alpha
    img_mask = Image.new('RGBA', (width, height), (255, 255, 255, 0))

    # Dessiner un cercle blanc à l'intérieur de l'image blanche
    draw = ImageDraw.Draw(img_mask)
    draw.ellipse((width/2 - radius, height/2 - radius, width/2 + radius, height/2 + radius), fill=(255, 255, 255, 255))

    # Appliquer le masque à l'image originale
    img_mask.paste(image, (0, 0), mask=img_mask)

    return img_mask
```

Figure 31 - Code algorithme K-NN amélioré 2/4

```
# Chargement des images de 'degage2'
for filename in os.listdir(degage2_dir):
    img = Image.open(os.path.join(degage2_dir, filename))
    img = img.convert('RGBA')
    img = apply_circular_mask(img)  # Appliquer le cache circulaire
    img = img.convert('L')  # Conversion en niveau de gris
    img = np.array(img)
    images.append(img)
    labels.append('degage')

# Chargement des images de 'couverture_partielle2'
for filename in os.listdir(couverture_partielle2_dir):
    img = Image.open(os.path.join(couverture_partielle2_dir, filename))
    img = img.convert('RGBA')
    img = apply_circular_mask(img)  # Appliquer le cache circulaire
    img = img.convert('L')  # Conversion en niveau de gris
    img = np.array(img)
    images.append(img)
    labels.append('couverture_partielle')

# Chargement des images de 'couverture_totale2'
for filename in os.listdir(couverture_totale2_dir):
    img = Image.open(os.path.join(couverture_totale2_dir, filename))
    img = img.convert('RGBA')
    img = apply_circular_mask(img)  # Appliquer le cache circulaire
    img = img.convert('L')  # Conversion en niveau de gris
    img = np.array(img)
    images.append(img)
    labels.append('couverture_totale')
```

Figure 32 - Code algorithme K-NN amélioré 3/4

```
# Conversion des listes en tableaux numpy
images = np.array(images)
labels = np.array(labels)

# Séparation des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2)

# Aplatir les images en vecteurs
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

# Création du classifieur k-NN et ajustement sur les données d'entraînement
k = 5  # Nombre de voisins
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Prédiction sur les données de test
y_pred = knn.predict(X_test)

# Calcul de l'exactitude (accuracy) du modèle
accuracy = accuracy_score(y_test, y_pred)
print("Exactitude : {:.2f}%".format(accuracy * 100))
```

Figure 33 - Code algorithme K-NN amélioré 4/4

e. Méthode utilisant les réseaux convolutifs

Code effectué à l'aide de ChatGPT pour tenter de générer un algorithme de classification d'images utilisant les réseaux convolutifs. Il donne une précision de 80% en moyenne.

Il utilise la bibliothèque « tensorflow » de Python, qui est la référence quand on souhaite débuter les neurones convolutifs. Le fonctionnement global de l'algorithme est ensuite similaire à celui de la méthode k-NN, même si n'on n'avons pas cherché à creuser chaque point.

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense


# Définition des chemins vers les dossiers contenant les images
train_dir = '/chemin/vers/degage2'
val_dir = '/chemin/vers/couverture_partielle2'
```

```
test_dir = '/chemin/vers/couverture_totale2'

# Définition des paramètres du modèle
batch_size = 32
epochs = 10
img_height, img_width = 150, 150
num_classes = 3  # Nombre de classes (dégagé, partiellement couvert, totalement nuageux)

# Création des générateurs de données pour la formation, la validation et les tests
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical')

# Création du modèle CNN
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_width, 3)))
```

```
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compilation du modèle
model.compile(optimizer='adam',
        loss='categorical_crossentropy',
        metrics=['accuracy'])

# Entraînement du modèle
model.fit(train_generator, epochs=epochs, validation_data=val_generator)

# Évaluation du modèle sur les données de test
loss, accuracy = model.evaluate(test_generator)
print("Loss:", loss)
print("Accuracy:", accuracy)
```

f. Remerciements

Nous souhaitions remercier en premier lieu nos encadrants sur ce Projet, Monsieur Alain Peden et Monsieur Frédéric Maussang, qui nous ont conseillés et aidés durant toute la période de conception, de réalisation, de vérification ainsi que pour la rédaction du rapport.

Nous tenions également remercier Monsieur Dominique Fabre qui nous a fait confiance pour l'élaboration d'un code et d'un projet important pour lui et pour l'observatoire, mais également pour la base de données triée qu'il nous a transmise.

Enfin, nous voulions remercier le copil de l'UE CODEVSI ainsi que Julien Mallet qui nous a indiqué et conseillé sur le choix du sujet.