

Projet chef-d'oeuvre

Melanoma

Christophe GARREAU

SIMPLON
.CO



Sommaire

Introduction	3
Problématique	3
Analyse des besoins	3
Gestion de projet	4
Intelligence artificielle	5
Dataset	5
Chargement des données	5
Description des données	7
Méthodes	8
Datasets	8
Arbres de décision	9
Random Forest Classifier	9
Gradient Boosting Classifier	10
XGBoost Classifier	10
Méthodologie d'entraînement	10
Réseaux de Neurones	11
Convolutional Neural Network (CNN)	11
Transfer Learning après extraction des features avec InceptionV3	12
Transfer Learning avec InceptionResNetV2 comme extracteur de feature	12
Méthodologie d'entraînement	12
Résultat	12
Axes d'amélioration	14
Application	15
Base de donnée	15
Conception	15
Création	16
Insertion des données	18
Utilisation des données	19
Sauvegarde	20
Performance	20
Backend	21
Frontend	23
Qualité et Monitoring	25
Qualité	25
Débogage	25
Tests de l'application	26
Journalisation	27
Axes d'amélioration	28
Conclusion	28

Introduction

Le mélanome, tumeur maligne du système pigmentaire (mélanocytes), est la plus grave des trois formes de cancers cutanés, à cause de sa faculté à engendrer des métastases. Il peut apparaître sur une peau saine, dans 70 à 80% des cas, ou résulter de la transformation maligne d'un grain de beauté.

Si le pronostic de la maladie est généralement bon, avec un taux de guérison élevé si le mélanome est détecté tôt et traité, principalement par la seule chirurgie, il se dégrade fortement pour les formes évoluées, surtout métastatiques, en dépit de l'arrivée constante de nouveaux traitements.

Ainsi un diagnostic précoce du mélanome est indispensable pour limiter la survenue de forme avancée. Ce diagnostic passe par l'observation régulière de la peau par un dermatologue, et la caractérisation visuelle de chaque anomalie détectée.

Problématique

Pour pouvoir assurer le suivi des lésions cutanées de ses patients, et donc de pouvoir détecter le plus tôt possible tout changement pouvant survenir sur les lésions, un médecin a d'accéder à l'historique de toutes les lésions et leur diagnostic.

De plus, comme la détermination du caractère bénin ou malin d'une lésion est basée sur l'observation de son asymétrie, de la régularité de ses bords, de l'homogénéité de sa couleur, de son diamètre, et surtout de son évolution, celle-ci peut ne pas être évidente. Aussi une aide à la décision basée sur un algorithme d'Intelligence Artificielle peut être utile pour qu'un médecin établisse un diagnostic.

Analyse des besoins

L'objectif de ce projet est donc de délivrer une application permettant à plusieurs médecins de suivre leurs patients et leurs lésions.

Ainsi après une identification, un médecin aura un aperçu de la liste de ses patients. En sélectionnant un patient de la liste, apparaîtra alors la liste des lésions identifiées sur la peau du patient par le médecin. De même en sélectionnant une lésion, le médecin aura un aperçu de l'historique de cette lésion, et pourra sélectionner l'état de la lésion lors des précédentes visites du patient, et afficher l'image de cette lésion ainsi que le diagnostic bénin ou malin associé.

En plus de cette consultation, le praticien pourra compléter la base de données de l'application, en ajoutant à sa convenance des patients, des lésions ou de nouvelles images à l'historique d'une lésion. L'ajout d'une nouvelle image se fera en sélectionnant une image sur le terminal du médecin, et un modèle de Machine Learning prédira l'état bénin ou malin de la lésion, avant que cette image et sa prédiction ne soient entrées dans la base de données.

Gestion de projet

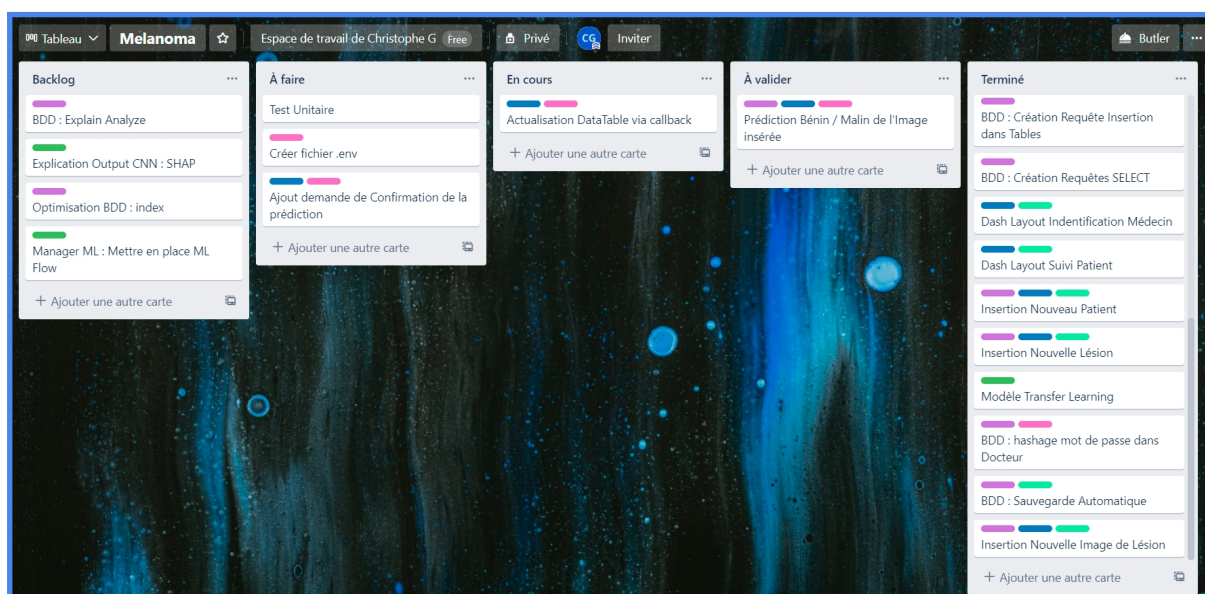
La méthode utilisée pour la gestion de ce projet est la méthode Kanban, car son système à flux tiré, guidé par la satisfaction du besoin du client, permet d'adapter les actions menées aux retours client, au fur et à mesure du développement. Ces retours doivent être fréquents pour que les fonctionnalités développées correspondent au mieux aux attentes du client, afin de ne pas en développer d'inutiles, et ainsi mieux tenir les coûts et les délais.

Le suivi visuel de l'avancement du projet est le tableau de l'outil TRELLO. Ainsi ce tableau est divisé en plusieurs colonnes décrivant l'état d'une fonctionnalité du produit développé ou d'une tâche particulière. Les différentes tâches glissent d'une colonne à l'autre au fil de leur développement. Les colonnes en question sont les suivantes :

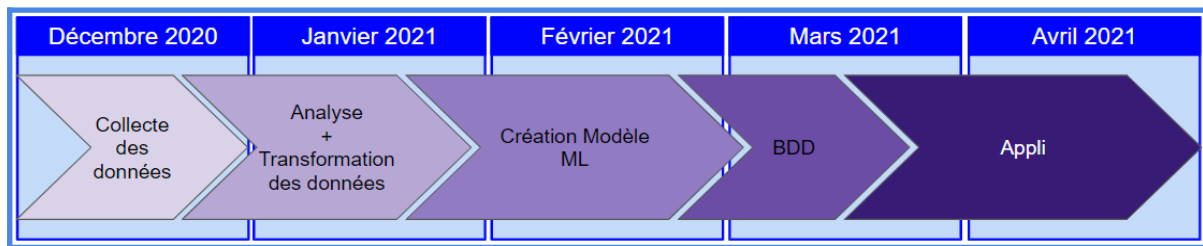
- Backlog : chaque élément de cette colonne représente une fonctionnalité à développer, fonctionnalité voulue par le client.
- A faire : une fonctionnalité du backlog est découpée en plusieurs tâches élémentaires nécessaires à la réalisation de la fonctionnalité. Dans l'attente de leur réalisation les tâches sont glissées dans cette colonne A faire.
- En cours : contient les tâches en cours de réalisation.
- A valider : le code des tâches de cette colonne doit être vérifié, testé et commenté par les autres développeurs de l'équipe projet, afin que la tâche en question soit conforme à la Definition of Done définie préalablement, avant le début du projet.
- Terminé : dans cette colonne se trouvent les tâches opérationnelles qui ont été validées, et qui sont conformes à la Definition of Done, condition indispensable pour qu'une tâche soit considérée comme terminée.

Ici, la Definition of Done d'une tâche est la suivante : "La tâche doit être opérationnelle en l'état, son code doit avoir été vérifié. Elle est considérée comme achevée totalement et est prête à être intégrée au projet développé livrable au client."

Début avril 2021, l'état de ce Kanban était le suivant :



Le projet s'est ainsi déroulé selon les étapes suivantes :



Intelligence artificielle

L'algorithme d'Intelligence Artificielle développé pour cette application permet de prédire le caractère bénin ou malin d'une lésion cutanée à partir d'une image en couleur de ladite lésion.

C'est un algorithme de classification binaire qui permet de classer la lésion concernée entre les classes BÉNIN et MALIN. Il est utilisé pour classer l'image chargée par le médecin au moment de son intégration dans la base de données.

En tant qu'algorithme de Machine Learning, il a besoin d'être entraîné et validé avec deux jeux de données différents et indépendants. Il apprend du premier jeu de données les caractéristiques communes aux images de chacune des deux classes. Le second jeu de données est utilisé pour réaliser des prédictions et vérifier la concordance entre les classes prédites et les vraies classes des lésions présentes sur les images de ce jeu de données.

Dataset

Chargement des données

Le jeu d'images utilisé pour entraîner les modèles de Machine Learning testés lors de la mise en place de cette application, a été collecté sur le site de l'International Skin Imaging Collaboration (<https://www.isic-archive.com>) via l'API de ce site. Le projet Melanoma de l'ISIC est un partenariat académique et industriel qui a pour but de faciliter l'application de l'imagerie numérique de la peau pour aider à réduire la mortalité due au mélanome.

L'API de ISIC a été requêtée à partir d'un Notebook par l'intermédiaire du code suivant :

```

api = ISICApi()
savePath = 'D:/Data/ISICArchive/'

if not os.path.exists(savePath):
    os.makedirs(savePath)

imageList = api.getJson('image?limit=23906&offset=0&sort=name')

print('Downloading %s images' % len(imageList))
imageDetails = []#

for image in imageList:
    if not os.path.isfile(savePath + image['name'] + str('.jpg')):
        print(image['_id'])
        imageFileResp = api.get('image/%s/download' % image['_id'])
        imageFileResp.raise_for_status()
        imageFileOutputPath = os.path.join(savePath, '%s.jpg' % image['name'])

        with open(imageFileOutputPath, 'wb') as imageFileOutputStream:
            for chunk in imageFileResp:
                imageFileOutputStream.write(chunk)

```

Il est à noter que pour la simple collecte des images aucun identifiant n'est exigé par l'API.

En parallèle un jeu de données contenant des métadonnées associées aux images téléchargées ci-avant est collecté, sous forme d'un fichier CSV, également via l'API ISIC, grâce au code suivant :

```

api = ISICApi()
outputFileName = 'D:/Data/ISICArchive/metadata'

imageList = api.getJson('image?limit=23906&offset=0&sort=name')

print('Fetching metadata for %s images' % len(imageList))
imageDetails = []

for image in imageList:
    print(' ', image['name'])
    # Fetch the full image details
    imageDetail = api.getJson('image/%s' % image['_id'])
    imageDetails.append(imageDetail)

    # Determine the union of all image metadata fields
    metadataFields = set(
        field
        for imageDetail in imageDetails
        for field in imageDetail['meta']['clinical'].keys()
    )
    metadataFields = ['isic_id'] + sorted(metadataFields)

    # Write the metadata to a CSV
    print('Writing metadata to CSV: %s' % outputFileName+'.csv')
    with open(outputFileName+'.csv', 'w') as outputStream:
        csvWriter = csv.DictWriter(outputStream, metadataFields)
        csvWriter.writeheader()
        for imageDetail in imageDetails:
            rowDict = imageDetail['meta']['clinical'].copy()
            rowDict['isic_id'] = imageDetail['name']
            csvWriter.writerow(rowDict)

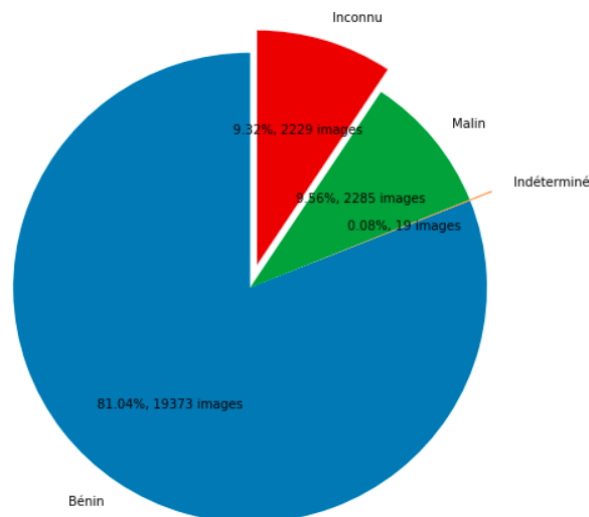
```

La collecte de ces données ne nécessite également pas d'identifiant spécifique.

Description des données

Les datasets, images et CSV de métadonnées, sont stockés sur un disque SSD externe pour des raisons de place, puisque les 23907 fichiers téléchargés depuis l'API ISIC ont une taille de 50Go.

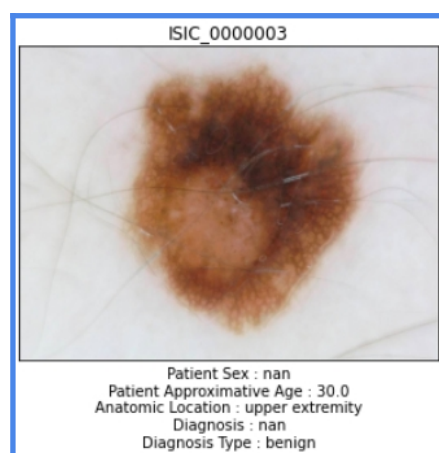
Les 23906 images téléchargées sont au format JPEG. Ce sont des photographies de lésions cutanées, au format 767 x 1022 pixel, en couleur. D'après le CSV de métadonnées, ces images sont réparties en plusieurs catégories, *Bénin*, *Malin*, *Inconnu* et *Indéterminé*, selon la répartition suivante :



Parmi ces catégories, seules les images des catégories *Bénin* et *Malin* sont gardées pour l'entraînement du modèle de Machine Learning. Les images gardées sont donc aux nombres suivants :

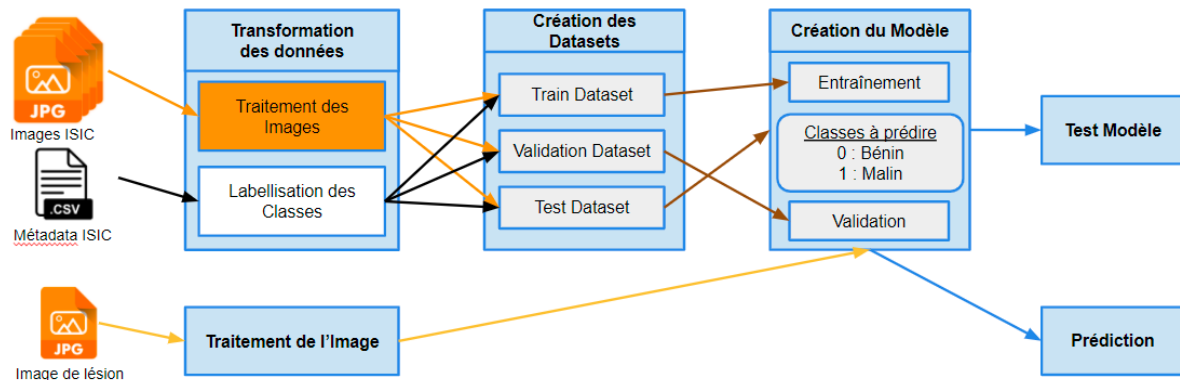
Classe : Bénin	Classe : Malin	Total
19373	2285	21658

Ci-après un exemple d'image avec ses métadonnées associées :



Méthodes

Le but de l'IA utilisée dans cette application est de classifier les images téléchargées par le médecin selon deux catégories : bénin ou malin. Pour ce faire la méthodologie est la suivante :



La transformation des données est un préalable indispensable à leur ingestion dans un modèle de Machine Learning, puisque ces modèles exigent notamment des données numériques. De plus, en fonction du type de modèle utilisé, les données des images doivent avoir un format particulier, rendant obligatoire un traitement spécifique.

Les modèles utilisés sont les suivants :

- XGBoostClassifier
- GradientBoostingClassifier
- RandomForestClassifier
- CNN
- Transfer Learning après extraction des features avec InceptionV3
- Transfer Learning avec InceptionResNetV2 comme extracteur de feature

Les trois premiers modèles utilisés ici sont des modèles de classification basés sur des arbres de décision. Les arbres de décision permettent la classification dans les différentes classes à l'aide de règles successives.

Les autres sont des modèles de réseaux de neurones, soit un réseau de neurones convolutifs (CNN), soit un réseau de neurones pré-entraîné, utilisé pour extraire les features des images, couplé à des couches de neurones connectés.

Datasets

Afin d'éviter un déséquilibre des classes *Bénin* et *Malin* dans le dataset, et ainsi a priori se placer dans de bonnes conditions pour entraîner un modèle de Machine Learning, le même nombre d'images de chaque classe est intégré dans chaque dataset.

Ces datasets sont au nombre de 3 : un dataset d'entraînement, un dataset de validation nécessaire pour vérifier la généralisation du modèle entraîné avec des données non

connues du modèle, et un dataset de test. Ce dernier dataset est utilisé pour réaliser des prédictions et mesurer les performances du modèle.

Lors de la construction des datasets chaque image d'un dataset n'est présente que dans ce dataset et en un seul exemplaire. Pour chaque dataset, un jeu d'image de chaque classe est sélectionné parmi toutes les images ISIC téléchargées, puis les deux jeux d'images sont mélangés aléatoirement.

Les datasets obtenus contiennent donc les nombres d'images suivants :

	Classe : Bénin	Classe : Malin	Total
Dataset d' Entraînement	1500	1500	3000
Dataset de Validation	500	500	1000
Dataset de Test	200	200	400

Chaque dataset se compose d'une partie image et d'une partie label, qui contient les classes des images.

Dans la partie image de chaque dataset, les images sont chargées, redimensionnées en 150 x 150 x 3 dimensions, et transformées en array de 150 arrays de 150 arrays de 3 valeurs. Ces valeurs sont ensuite standardisées en les divisant par 255, qui est la valeur maximale d'un pixel.

Dans la partie label de chaque dataset, ces labels sont transformés en 0 pour *Bénin* et 1 pour *Malin* à l'aide d'un LabelEncoder.

Les 3 datasets sont ainsi prêts pour être utilisés pour l'entraînement des modèles.

Arbres de décision

Un arbre de décision est une méthode de classification logique grâce au calcul du choix le plus viable. La racine de cet arbre est le classement à effectuer, ici le classement en *Bénin* ou *Malin*. Les branches de l'arbre sont les choix à faire pour cette classification, chaque choix débouchant sur d'autres choix, ce qui augmente la profondeur de l'arbre. L'algorithme choisit les critères de prise de décision et les évalue en un score qu'il cherche à maximiser pour aboutir à la classification désirée la plus fiable possible.

Random Forest Classifier

Ce modèle est un groupe d'arbres de décision entraînés sur des sous-ensembles de données différentes aléatoirement choisies. Le modèle réalise des prédictions avec tous les arbres et choisit la meilleure. Il offre de meilleures performances que l'arbre de décision seul grâce à un grand nombre d'arbres, qui limitent l'overfitting que pourrait engendrer un seul

arbre. Cependant il est beaucoup plus lent qu'un arbre de décision seul, et difficile à interpréter, l'interprétabilité étant la force des arbres de décision.

Gradient Boosting Classifier

Ce modèle est un ensemble construit en ajoutant un à un des arbres de décision dont l'objectif est de minimiser les erreurs commises par l'arbre précédent. Il est optimisé lors de l'entraînement en minimisant une fonction de loss par la technique de descente de gradient.

XGBoost Classifier

Ce modèle d'Extrême Gradient Boosting Classifier se différencie des autres modèles de gradient boosting par sa rapidité accrue de calcul et ses meilleures performances.

Méthodologie d'entraînement

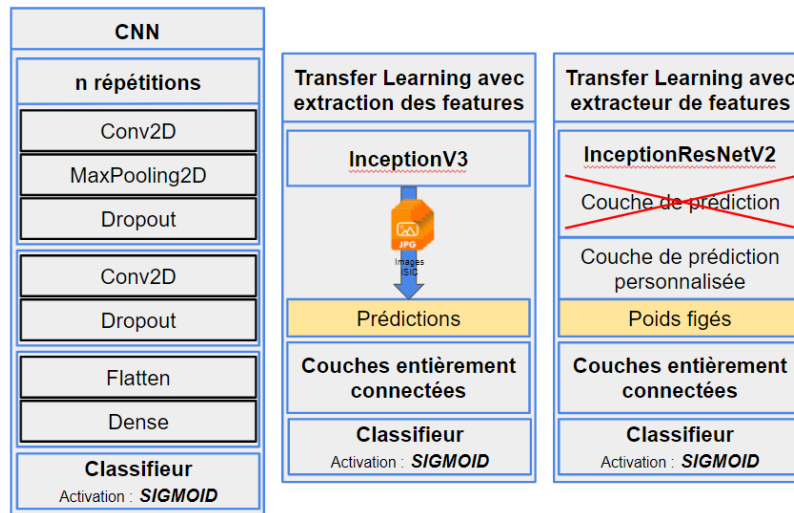
Ces modèles sont entraînés avec les datasets d'images et de labels d'entraînement, en utilisant leurs hyperparamètres par défaut.

Random Forest Classifier	Gradient Boosting Classifier	XGBoost Classifier
<pre>{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': 42, 'verbose': 0, 'warm_start': False}</pre>	<pre>{'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0.1, 'loss': 'deviance', 'max_depth': 3, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_iter_no_change': None, 'presort': 'deprecated', 'random_state': None, 'subsample': 1.0, 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': 0, 'warm_start': False}</pre>	<pre>{'base_score': 0.5, 'booster': 'gbtree', 'colsample_bylevel': 1, 'colsample_bynode': 1, 'colsample_bytree': 1, 'gamma': 0.5, 'gpu_id': -1, 'importance_type': 'gain', 'interaction_constraints': '', 'learning_rate': 0.01, 'max_delta_step': 0.1, 'max_depth': 4, 'min_child_weight': 0.2, 'missing': nan, 'monotone_constraints': '()', 'n_estimators': 10, 'n_jobs': 4, 'nthread': 4, 'num_class': 2, 'num_parallel_tree': 1, 'objective': 'multi:softmax', 'random_state': 0, 'reg_alpha': 0.5, 'reg_lambda': 0.8, 'scale_pos_weight': 1, 'silent': False, 'subsample': 0.8, 'tree_method': 'exact', 'validate_parameters': 1, 'verbosity': None}</pre>

Ces modèles sont cross validés : le dataset d'entraînement est divisé, ici en 2, aléatoirement, et le modèle est entraîné sur un des sous datasets et testé sur l'autre, et inversement. Chacun de ces sous datasets est utilisé une fois pour l'entraînement et une fois pour le test.

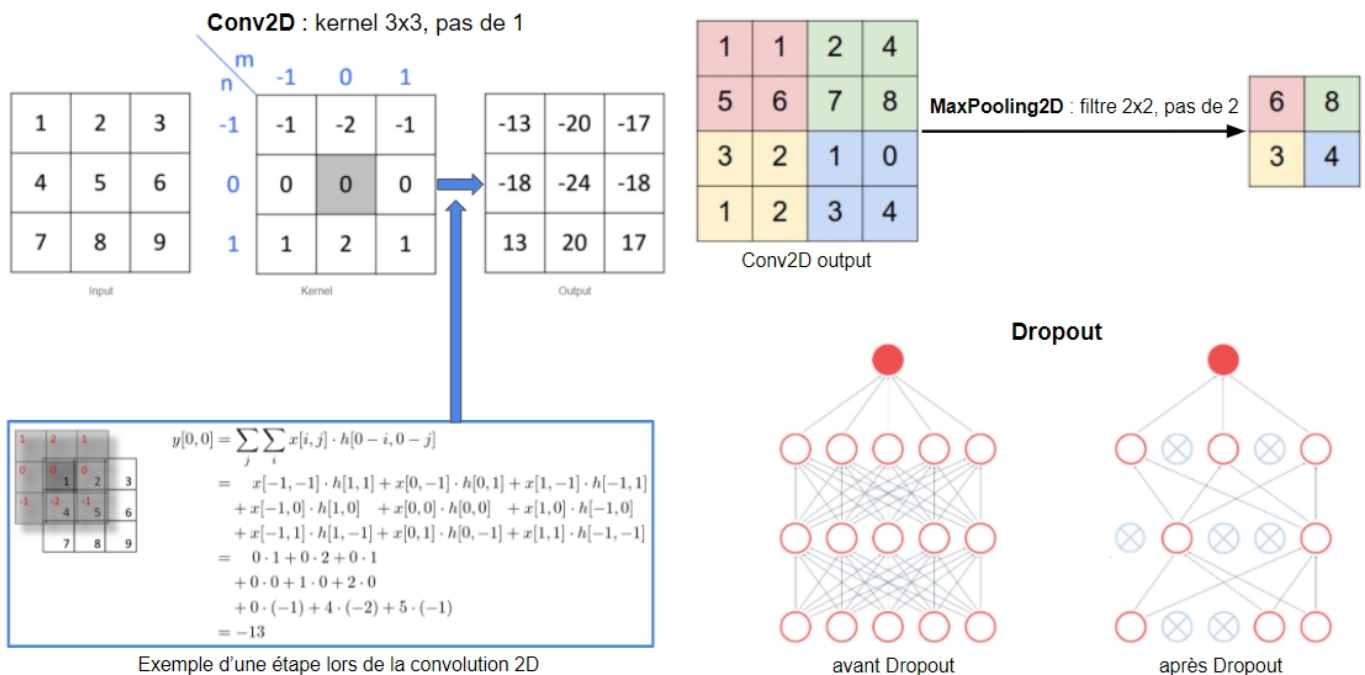
Réseaux de Neurones

Ces modèles utilisés ici ont les architectures suivantes :



Convolutional Neural Network (CNN)

Un CNN se compose de couches de Conv2D, qui appliquent un produit matriciel de chaque pixel avec un kernel, de couches de MaxPooling2D, qui sélectionnent seulement la valeur maximale d'un groupe de pixel, et des couches de Dropout, qui déconnectent un pourcentage choisi des neurones générés précédemment. Le Dropout est utilisé comme régularisation afin d'éviter l'overfitting.



Ce modèle se termine par une couche Flatten, destinée modifier la dimension des outputs du réseau ci-avant, suivie d'une couche Dense activée par une fonction Sigmoid, destinée à réaliser la classification proprement dite.

Transfer Learning après extraction des features avec InceptionV3

Ce modèle est un réseau de neurones dans lequel les données injectées sont les prédictions faites par le modèle InceptionV3 de Keras à partir des images ISIC préalablement transformées. Ce modèle est un modèle séquentiel composé de couches Dense et Dropout, via une couche GlobalAveragePooling2D, qui récupère les features extraites par le modèle InceptionV3. Le modèle est terminé par une couche Dense, activée par une fonction Sigmoid.

Transfer Learning avec InceptionResNetV2 comme extracteur de feature

Ce modèle est construit à partir du modèle pré-entraîné InceptionResNetV2 de Keras dont la dernière couche de prédiction a été enlevée et remplacée par une couche personnalisée. Les poids de ce modèle ainsi modifié sont figés et récupérés pour servir d'extracteur de features des inputs du futur modèle final. Des couches entièrement connectées sont ensuite ajoutées, suivies d'un classifieur (couche Dense) utilisant une fonction d'activation sigmoid.

Méthodologie d'entraînement

Après construction, ces modèles neuronaux sont d'abord compilés, avec comme optimiseur *RMSprop*, et comme loss *binary_crossentropy*. L'optimiseur et la loss sont deux paramètres de compilation. Le premier permet d'adapter les poids affectés à chaque neurone à chaque époque (cet optimiseur possède un taux d'apprentissage qui influe sur la vitesse de variation de cet optimiseur), et le second est une fonction que le modèle cherche à diminuer lors de l'entraînement.

Le modèle est ensuite entraîné avec les datasets d'images et de labels d'entraînement, pendant un nombre d'époque suffisant pour que la loss se stabilise, et avec un batch size à ajuster afin d'optimiser les performances. Parallèlement à l'entraînement, le modèle utilise les datasets d'images et de labels de validation pour ajuster les poids du modèle afin d'éviter l'overfitting.

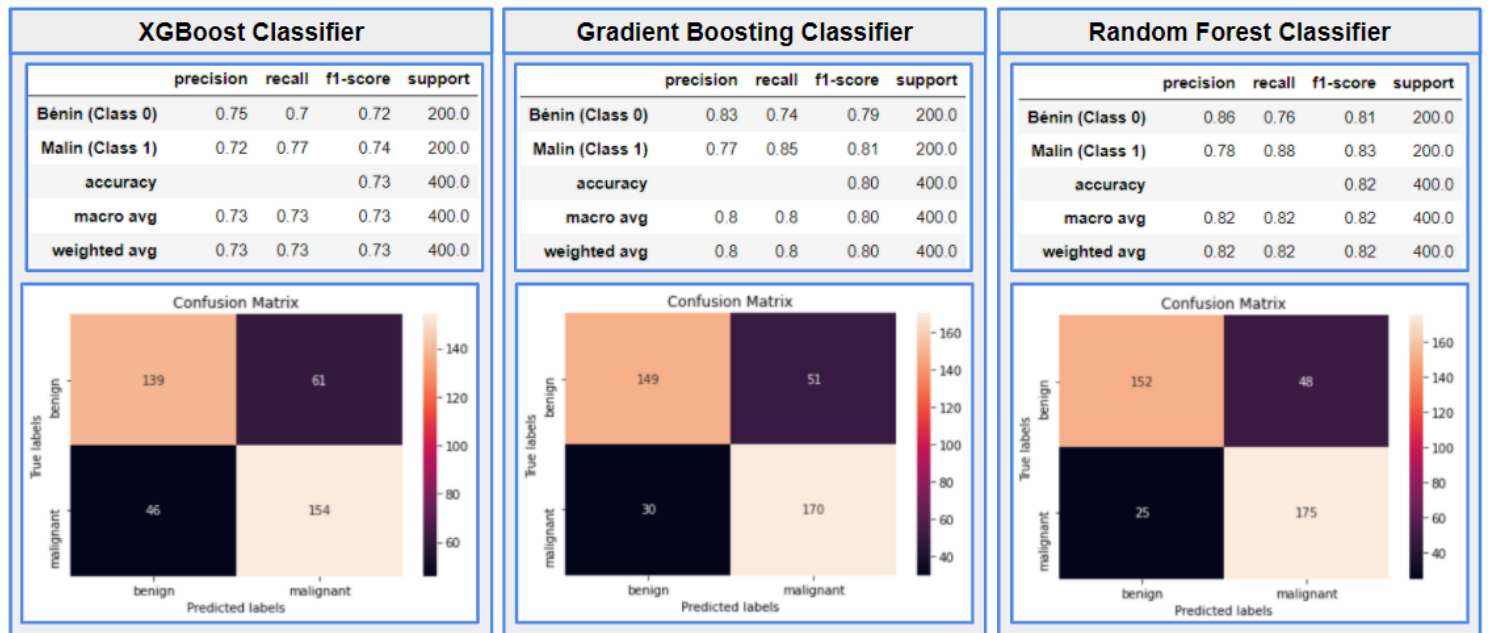
Au cours de cet entraînement les métriques choisies sont enregistrées. Ces métriques sont, pour ces modèles de classification, l'accuracy, la précision, le recall, et le f1 score. Elles permettent d'analyser les performances du modèle au cours de l'entraînement, de vérifier si cet entraînement est optimal, si les courbes stagnent, et de contrôler l'overfitting. Un modèle bien entraîné est caractérisé par des métriques de validation proches de celles d'entraînement et légèrement inférieures, faute de quoi le modèle aurait un overfitting.

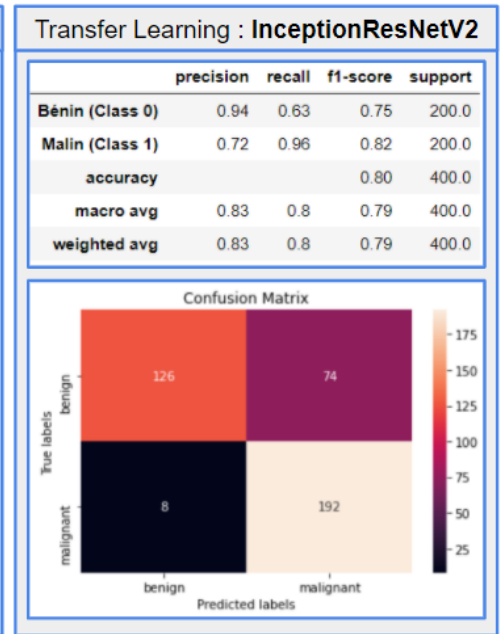
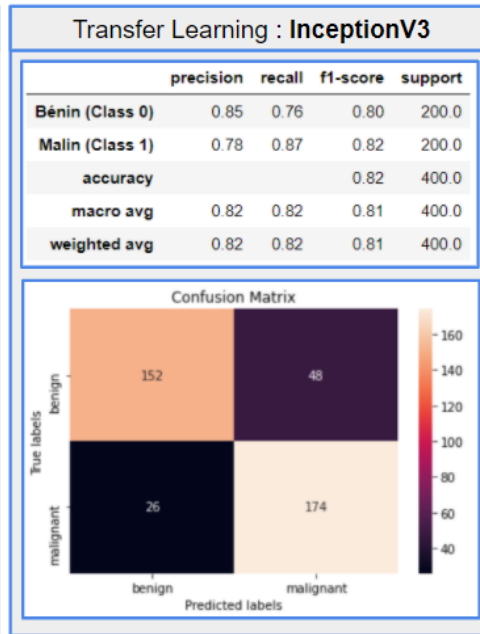
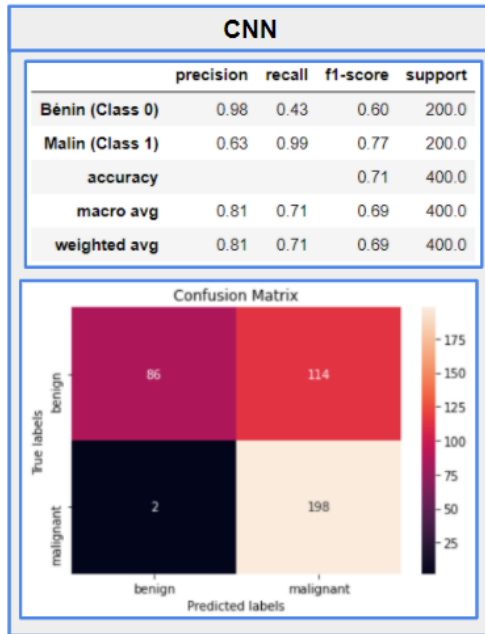
Résultat

Après entraînement des différents modèles, les métriques (accuracy, precision, recall et f1-score) sont récupérées et synthétisées dans le tableau suivant :

	Accuracy Train	Accuracy Val	Accuracy Test	Precision Classe 0 "Bénin"	Precision Classe 1 "Malin"	Recall Classe 0 "Bénin"	Recall Classe 1 "Malin"	F1-score Classe 0 "Bénin"	F1-score Classe 1 "Malin"
XGBoost Classifier	80%	70%	73%	75%	72%	70%	77%	72%	74%
Gradient Boosting Classifier	94%	75%	80%	83%	77%	74%	85%	79%	81%
Random Forest Classifier	100%	77%	82%	86%	78%	76%	88%	81%	83%
CNN	79%	68%	71%	98%	63%	43%	99%	60%	77%
Transfer Learning : InceptionV3	88%	80%	82%	85%	78%	76%	87%	80%	82%
Transfer Learning : InceptionResNetV2	83%	80%	80%	94%	72%	63%	96%	75%	82%

De plus pour chacun des modèles, après prédiction sur le dataset de test, la matrice de confusion de chacun des modèles est établie. Une matrice de confusion est un tableau qui représente, dans l'ordre de gauche à droite et de haut en bas, le nombre de Vrai Positif, de Faux Négatif, de Faux Positif, et de Vrai Négatif. Ces matrices, ainsi que les rapports de classification des modèles, sont présentés ci-après :





Ici, l'objectif est de classer des images de lésion cutanée en *Bénin* ou *Malin*. Or une mauvaise classification d'une image de lésion maligne peut s'avérer dramatique pour le patient, puisque l'essentiel dans le traitement du mélanome est que le diagnostic soit le plus précoce possible. Ainsi, pour être efficace dans ce contexte, le meilleur modèle est celui qui minimise le nombre de Faux Positifs.

Il apparaît donc que le meilleur des six modèles étudiés ci-avant est le modèle *CNN*, qui a le plus petit nombre de Faux Positifs. Cependant, avec seulement 3% de Faux Positifs en plus, mais 20% de Faux Négatifs en moins, le modèle de transfer learning basé sur *InceptionResNetV2* est plus performant. Ce choix est confirmé par un f1-score de la classe *Malin* de ce modèle supérieur.

Il est à noter que le f1-score du modèle *RandomForestClassifier* est supérieur, mais ce modèle possède plus de Faux Positifs, et une accuracy d'entraînement bien supérieure à celles de validation et de test qui atteint les 100%, ce qui traduit un overfitting, qui est une trop forte adaptation aux données d'entraînement qui empêche le modèle de s'adapter à de nouvelles données.

Axes d'amélioration

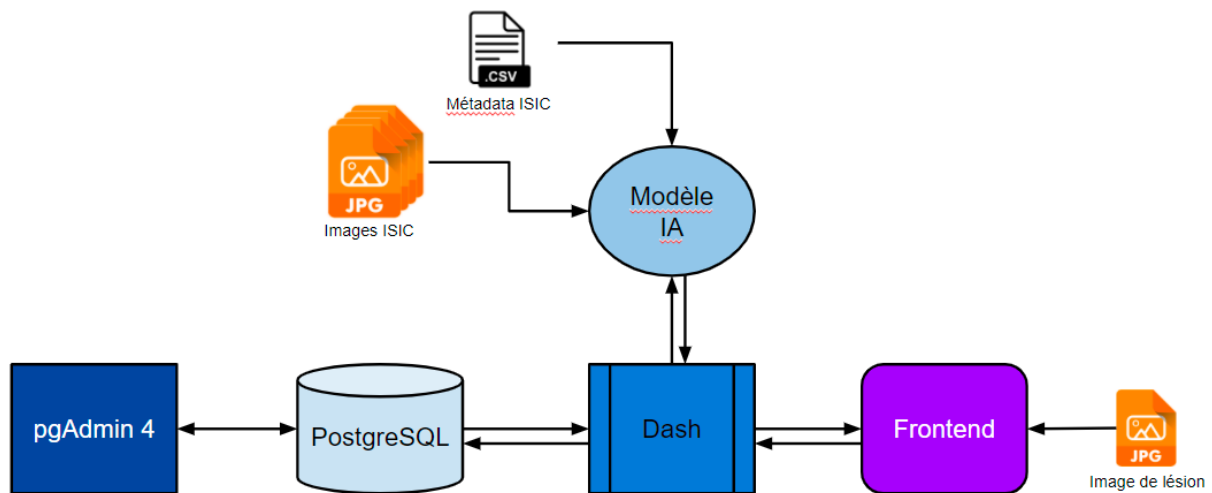
Outre une optimisation plus approfondie du modèle offrant les meilleures performances, un axe d'amélioration possible serait de concaténer les informations du csv de metadata non utilisées aux images du dataset d'entraînement, et d'entraîner un réseau de neurones avec ce dataset, vraisemblablement un modèle de transfer learning.

Application

L'application *Melanoma* est un outil de suivi des lésions cutanées des patients de plusieurs médecins, couplé à une aide au diagnostic du caractère bénin ou malin d'une lésion

cutanée.

Elle est composée de plusieurs composants qui interagissent entre eux comme suit :



Plus en détail, par le biais du Frontend, le médecin utilisateur s'identifie tout d'abord. Le Backend Dash requête alors la base de données PostgreSQL avec les identifiants saisis, et affiche ensuite le tableau de suivi avec les données propres au médecin qui s'est identifié. Puis selon les choix du médecin sur le Frontend, le Backend requête la base de données pour afficher les historiques voulus. Le médecin peut également ajouter un nouveau patient ou une nouvelle lésion pour un patient voulu, via le Frontend, le Backend se chargeant après validation de transmettre la requête d'insertion à la base de données. Enfin, le médecin peut ajouter une image d'une lésion directement depuis son ordinateur, toujours via le Frontend. Là, le Backend exécute la prédiction du caractère bénin ou malin de la lésion contenue sur l'image choisie, grâce au modèle préalablement entraîné avec les images et les métadonnées ISIC. Une fois la prédiction validée via le Frontend par le médecin, le Backend envoie une requête d'insertion à la base de données pour y insérer le chemin de l'image et la prédiction.

Parallèlement, la base de données peut être administrée par un développeur via pgAdmin.

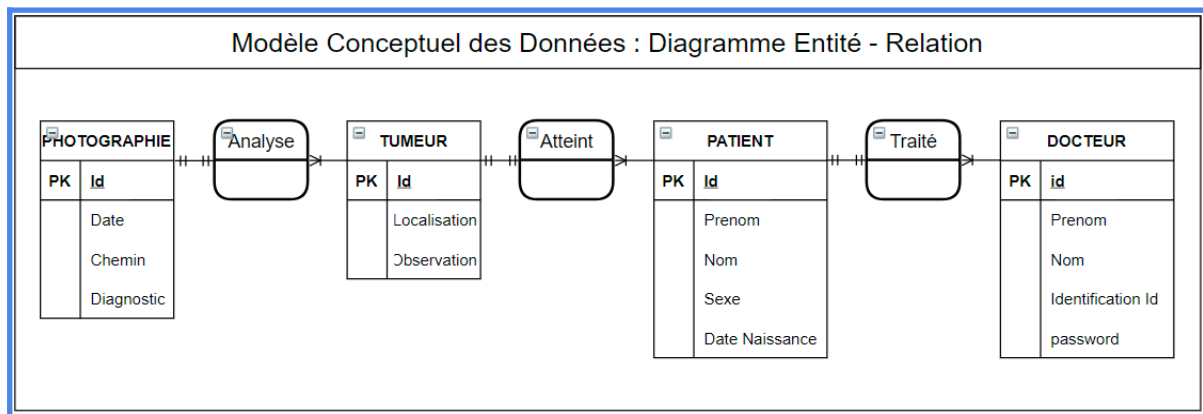
Base de donnée

Conception

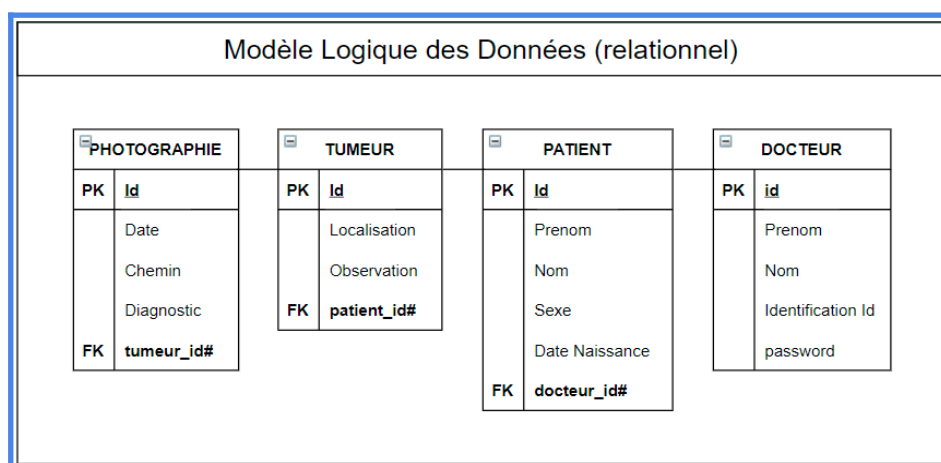
Afin d'exploiter pleinement les relations entre les différentes entités permettant de gérer et de suivre l'historique des consultations des patients, une base de données relationnelle a été choisie.

Pour son élaboration, dans une première étape le Diagramme Entité - Relation suivant a été établi par le listage des 4 entités impliquées dans le suivi des lésions d'un patient. Ces Entités sont liées linéairement deux à deux par une Relation spécifique, qui traduit l'action de l'une des Entités sur l'autre. Ces Relations sont caractérisées par le nombre d'Entités possiblement reliées entre elles, dans un sens et dans l'autre de la Relation. Ici ces

cardinalités sont dans une direction 1-1 et dans la direction opposée 1-n.



A partir de ce Diagramme Entité - Relation, un Modèle Logique des Données est établi en transformant toutes les Entités et les Relations en Tables reliées entre elles. Dans le cas de cette base de données les Relations ne portant pas en elles d'information spécifique supplémentaire, elles ne sont pas transformées en Table. Les Relations entre les Tables sont donc retranscrites dans une Table par une Clé Étrangère qui reprend un des attributs de la Table à laquelle elle est liée.



Création

Le système de base de données relationnelle utilisé pour ce projet est PostgreSQL, avec son interface pgAdmin. Les requêtes nécessaires à la création et à la manipulation de cette base de données via Python sont exécutées via l'adaptateur psycopg2.

A partir d'un Notebook, la base de donnée est créée à l'aide du code suivant :

CREATION DE LA BASE DE DONNEES

```
# Connect to PostgreSQL DBMS
conn = psycopg2.connect(host='localhost', user='postgres', password='[REDACTED]');
conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT);

# Obtain a DB Cursor
cursor = conn.cursor();
name_Database = "final_project";

# Create table statement
sqlCreateDatabase = "create database "+name_Database+";"

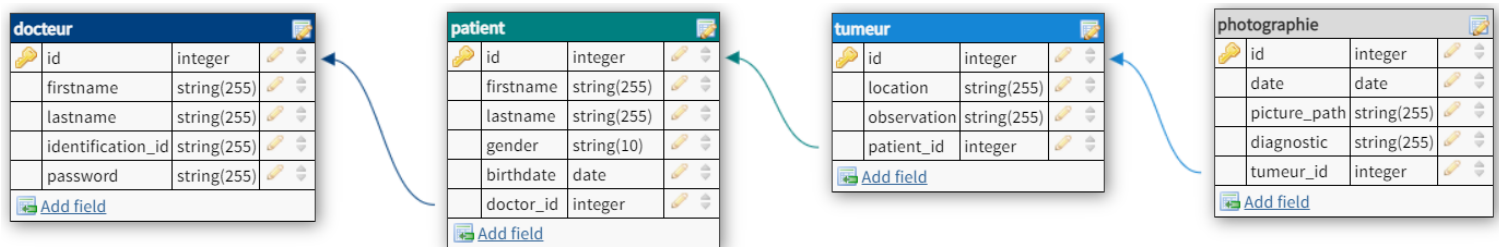
# Create a table in PostgreSQL database
cursor.execute(sqlCreateDatabase);
```

Une fois la base de données créée, les différentes Tables sont créées à l'aide de sections de code semblables à celle ci-après :

CREATION DE LA TABLE DOCTEUR

```
conn = psycopg2.connect(host='localhost', database="final_project", user='postgres', password='[REDACTED]');
cur = conn.cursor()
command = (
    """
    CREATE TABLE Docteur (
        id SERIAL PRIMARY KEY,
        firstname VARCHAR(255) NOT NULL,
        lastname VARCHAR(255) NOT NULL,
        identification_id VARCHAR(255) NOT NULL,
        password VARCHAR(255) NOT NULL
    )
    """
)
cur.execute(command)
conn.commit()
cur.close()
```

Ainsi, après l'exécution des requêtes précédemment évoquées, les Tables suivantes ont été créées :



Insertion des données

L'application ayant pour objet le suivi des patients et de leurs lésions par un médecin, la seule table qui a été remplie via un Notebook, et non directement depuis l'application, est la Table Docteur. En effet puisque cette Table stocke les identifiants de connexion pour l'utilisateur médecin, elle est remplie préalablement, par le biais de la requête suivante :

```
INSERTION NOUVEAU DOCTEUR

conn = psycopg2.connect(host='localhost', database="final_project", user='postgres', password='[REDACTED]');
cur = conn.cursor()
command = (
    """
    INSERT INTO Docteur (
        firstname,
        lastname,
        identification_id,
        password
    )
    VALUES (%s, %s, %s, %s)
    RETURNING id;
    """
)

record_to_insert = ('Christophe', 'Garreau', 'christophe', blake2s(b'Mad33+').hexdigest())

cur.execute(command, record_to_insert)
conn.commit()

# get the generated id back
client_id = cur.fetchone()[0]

cur.close()
```

Il est à noter que les mots de passe d'identification rentrés lors de la création d'un nouveau médecin sont hashés au moment de leur insertion dans la Table Docteur, pour des raisons de sécurité.

Les insertions dans les autres Tables sont réalisées directement depuis l'application en cliquant sur les boutons dédiés. Une fois le bouton "Valider" cliqué après saisie des informations, ces informations sont insérées dans la table correspondante, via une requête analogue à celle ci-après :

INSERTION D'UN NOUVEAU PATIENT

```
def new_patient(new_firstname, new_lastname, new_gender, new_birthdate, doc_id):
    conn = psycopg2.connect(host='localhost', database="final_project", user='postgres', password='[REDACTED]')

    cur = conn.cursor()
    command = (
        """
        INSERT INTO Patient (
            firstname,
            lastname,
            gender,
            birthdate,
            doctor_id
        )
        VALUES (%s, %s, %s, %s, (SELECT id from Docteur WHERE id=%s))
        RETURNING id;
        """
    )

    cur.execute(command, (new_firstname, new_lastname, new_gender, new_birthdate, str(doc_id)))
    conn.commit()

    # get the generated id back
    patient_id = cur.fetchone()[0]

    cur.close()

    return print(f"Nouveau Patient créé avec l'id : {patient_id}")
```

Utilisation des données

Chacun des tableaux affichés dans l'application est le résultat d'une requête à la Table correspondante, effectuée grâce à un code semblable à celui ci-après.

RECUPERATION DE LA LISTE DES LESIONS D'UN PATIENT

```
def tumeurs_display(pat_id):
    conn = psycopg2.connect(host='localhost', database="final_project", user='postgres', password='[REDACTED]')

    cur = conn.cursor()

    command = (
        """
        SELECT id, location, observation FROM Tumeur WHERE patient_id = %s
        """
    )

    cur.execute(command, str(pat_id))
    Tumeurs = cur.fetchall()
    cur.close()

    # Extract the column names
    col_names = list(map(lambda x: x[0], cur.description))

    # Create the dataframe, passing in the list of col_names extracted from the description
    df = pd.DataFrame(Tumeurs, columns=col_names)

    return df
```

Sauvegarde

Une sauvegarde automatique de la base de données a été mise en place via l'exécution périodique par le Planificateur de Tâches de Windows du fichier .bat suivant :

SAUVEGARDE AUTOMATIQUE DE LA BASE DE DONNEES

```
auto_database_backup.bat
1  @echo Backup database %PG_PATH%%PG_FILENAME%
2  @echo off
3  SET PG_BIN="C:\Program Files\PostgreSQL\13\bin\pg_dump.exe"
4  SET PG_HOST=localhost
5  SET PG_PORT=5432
6  SET PG_DATABASE=final_project
7  SET PG_USER=postgres
8  SET PGPASSWORD=[REDACTED]
9  SET PG_PATH="C:\Users\Christophe GARREAU\Desktop\final_project\database_backup"
10
11  rem date
12  SET DJOUR=%DATE:~0,2%
13  SET DMOIS=%DATE:~3,2%
14  SET DANNEE=%DATE:~6,10%
15
16  rem heure
17  SET HEURE=%TIME:~0,2%
18  SET MINUTE=%TIME:~3,2%
19
20  SET SEP=_
21
22  SET PG_FILENAME=%PG_PATH%\%PG_DATABASE%%SEP%%DJOUR%%SEP%%DMOIS%%SEP%%DANNEE%%SEP%%HEURE%%SEP%%MINUTE%.sql
23
24
25  %PG_BIN% -h %PG_HOST% -p %PG_PORT% -U %PG_USER% %PG_DATABASE% > %PG_FILENAME%
26
27  @echo Backup Taken Complete %PG_PATH%%PG_FILENAME%
```

Cette sauvegarde automatique permet de stocker dans le dossier *database_backup* les fichiers .sql périodiquement générés. Ces fichiers permettent, le cas échéant, de restaurer la base de données dans l'état où elle était à la date de la sauvegarde choisie.

Performance

La base de données étant très restreinte actuellement, les requêtes s'exécutent rapidement. Pour suivre cette vitesse d'exécution il est possible de demander lors de la requête SQL les informations concernant le temps d'exécution des différentes parties de la requête, en précédant la requête par *EXPLAIN ANALYZE*, comme dans l'exemple suivant :

MESURE DU TEMPS D'EXECUTION D'UNE REQUETE

```
conn = psycopg2.connect(host='localhost', database="final_project", user='postgres', password='');  
cur = conn.cursor()  
  
command = (  
    """  
    EXPLAIN ANALYZE SELECT Patient.firstname, Patient.lastname FROM Patient  
    INNER JOIN Docteur ON Docteur.id = Patient.docteur_id  
    WHERE docteur_id = %s  
    """  
)  
  
cur.execute(command, str(1))  
Explanation = cur.fetchall()  
cur.close()
```

Ce qui est récupéré en retour regroupe les informations suivantes :

```
['Nested Loop (cost=0.14..19.04 rows=1 width=1032) (actual time=0.080..0.082 '  
'rows=1 loops=1)',  
' -> Seq Scan on patient (cost=0.00..10.88 rows=1 width=1036) (actual '  
'time=0.014..0.015 rows=1 loops=1)',  
'   Filter: (docteur_id = 1)',  
' -> Index Only Scan using docteur_pkey on docteur (cost=0.14..8.15 rows=1 '  
'width=4) (actual time=0.064..0.065 rows=1 loops=1)',  
'   Index Cond: (id = 1)',  
'   Heap Fetches: 1',  
'Planning Time: 1.760 ms',  
'Execution Time: 0.156 ms']
```

Lorsque la taille de cette base de données augmentera fortement avec le nombre de patients et donc d'images de leurs lésions, il sera utile d'ajouter des index pour accélérer les requêtes liées à cet index. Ainsi la requête SQL de cette création d'index serait par exemple de la forme suivante :

CREATE INDEX patient-index ON patient (lastname varchar);

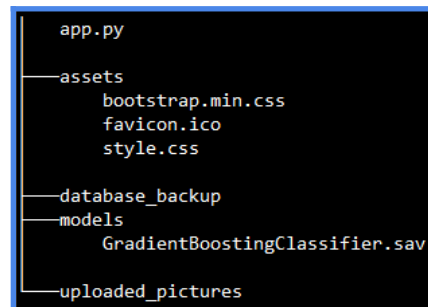
Ainsi lors d'une requête sur lastname de la table patient contenant une condition WHERE de type varchar, un appel direct à cet index sera effectué sans passer toute la table au crible, ce qui accélèrera grandement la durée de la requête.

Backend

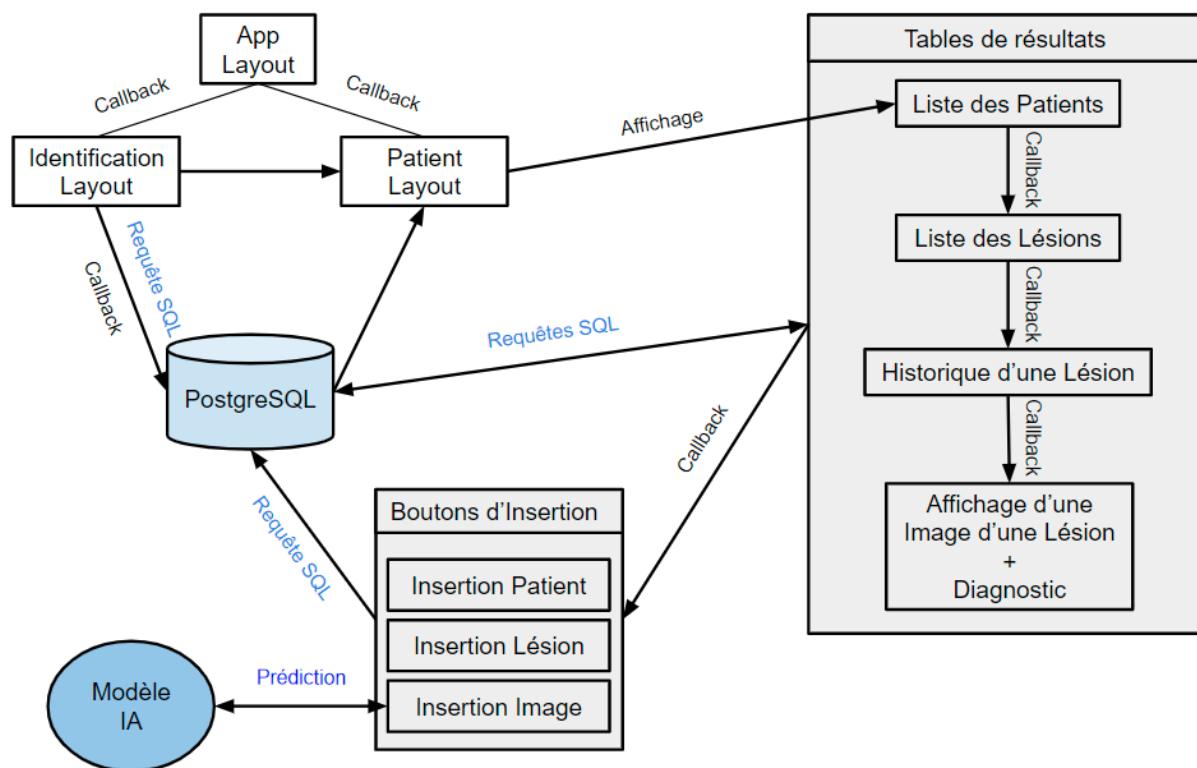
Le Backend de l'application Melanoma a été codé en utilisant le framework Plotly Dash, basé sur Flask, Plotly.js, et React.js. C'est une librairie open source sous licence MIT, qui permet de créer en Python des applications web de visualisation de données hautement personnalisables.

Cette application se compose d'un script Python contenant l'intégralité du code de l'application, d'un dossier assets contenant notamment le CSS de Dash, d'un dossier models contenant les sauvegardes des modèles d'IA entraînés avec le dataset ISIC, d'un dossier uploaded_pictures contenant la copie des images téléchargées par le médecin, et

d'un dossier *database_backup* contenant les sauvegardes périodiques de la base de données. L'arborescence de tous ces fichiers et ces dossiers est la suivante :



Ce Backend a le fonctionnement suivant :



Ainsi ce Backend gère deux pages différentes. Généralement un script Python est créé par page affichée par l'application, or ici le choix a été fait de tout inclure dans le même script afin que la page *identification* puisse partager les variables qu'elle génère avec la page *patient*.

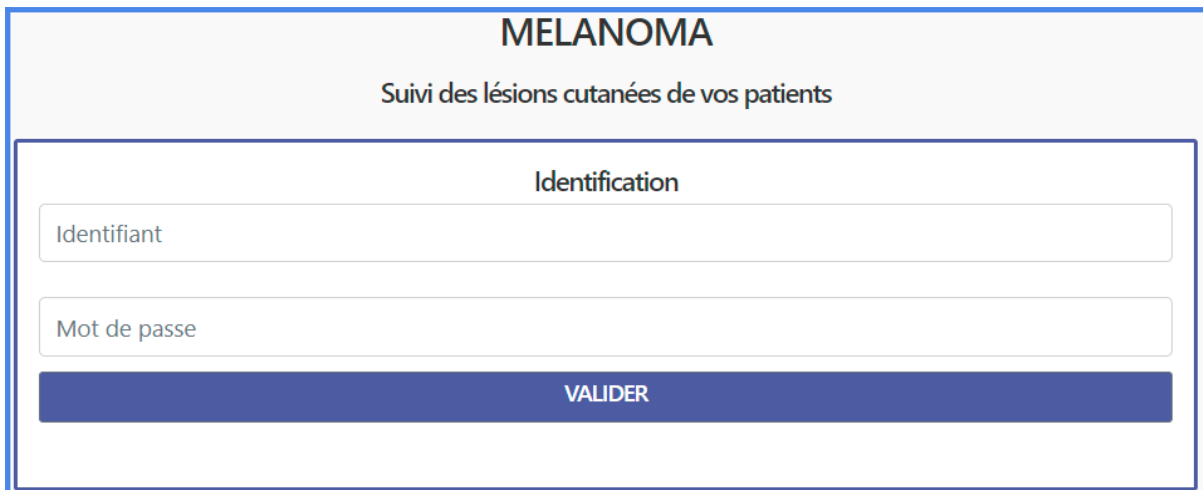
Chacune des pages communique avec la base de données. La page *identification* requête les identifiants de la Table docteur. La page *patient* quant à elle, récupère l'id du médecin qui s'est identifié, et requête la base de données pour récupérer les informations patient, lésion et historique à afficher dans les différents tableaux qui seront présentés par le Frontend. Tous les tableaux communiquent entre eux par le biais de callbacks, qui sont des fonctions Python de Dash automatiquement appelées quand les propriétés d'un composant input changent. Ces composants input sont les éléments Dash qui permettent de construire le Frontend.

Enfin un dernier groupe de callbacks gère la création de nouveaux patients, de nouvelles lésions, et de nouvelles images, et l'insertion de ces nouvelles entités dans la base de données. Ces callbacks communiquent avec les tableaux précédents pour y récupérer les identifiants nécessaires à l'insertion du nouvel élément.

Notamment, le callback gérant l'insertion d'une nouvelle image d'une lésion d'un patient, récupère les identifiants du patient et de la lésion, récupère l'image à insérer ajoutée par le médecin, et, par le biais d'une fonction ad hoc, la copie dans le répertoire *uploaded_pictures*, et la transforme afin de l'utiliser pour réaliser une prédiction de son caractère bénin ou malin, en utilisant le modèle d'IA. Il insère ensuite le lien de cette image et la prédiction dans la Table *photographie*.

Frontend

Le Frontend de l'application Melanoma a été également codé avec le framework Plotly Dash. Une fois démarrée, l'application s'affiche sur un navigateur web à l'adresse *localhost:8051*. Une fois l'application lancée, l'utilisateur médecin arrive, via la layout *identification_layout*, sur la page d'identification suivante où il doit s'identifier :



The screenshot shows the 'MELANOMA' application interface. At the top, the title 'MELANOMA' is displayed in a large, bold, black font, with the subtitle 'Suivi des lésions cutanées de vos patients' below it in a smaller, regular black font. The main content area is a white box with a blue border, titled 'Identification'. It contains two input fields: 'Identifiant' and 'Mot de passe', both with light gray placeholder text. Below these fields is a prominent blue button with the white text 'VALIDER'.

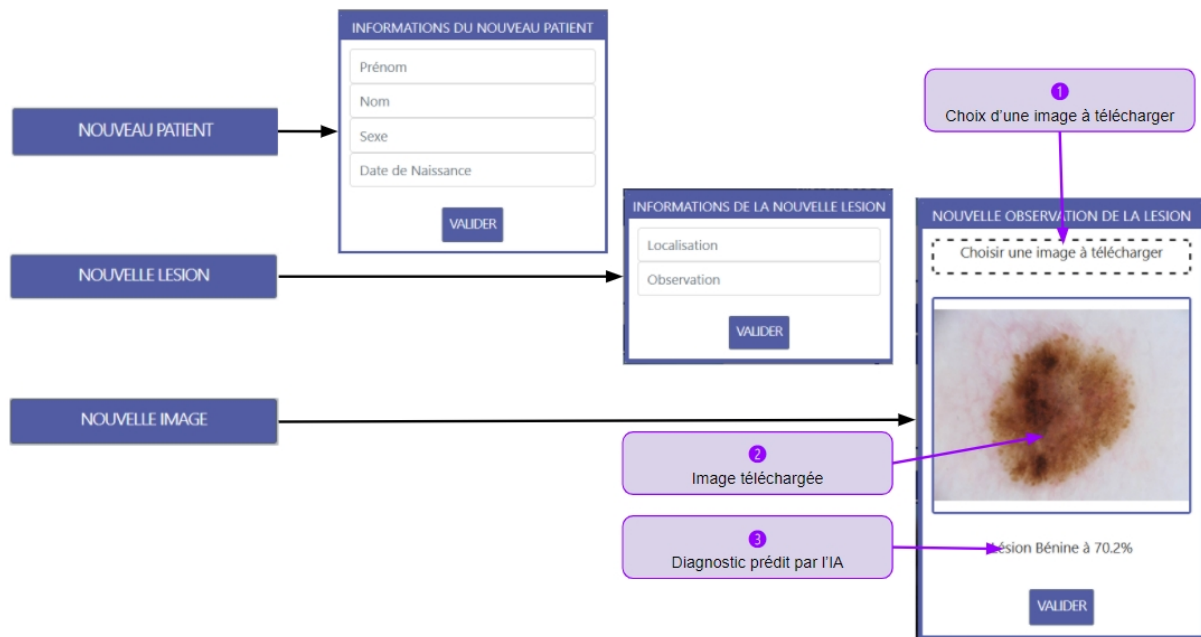
Une fois identifié, le médecin arrive sur un écran (layout *patient_layout*) où sont affichés ses patients :

GESTION DES PATIENTS	
<div>LISTE DES PATIENTS</div> <div>Christophe Garreau</div>	<div>LESIONS DU PATIENT</div> <div>Aucun Patient sélectionné</div>
	<div>HISTORIQUE DE LA LESION</div> <div>Aucun historique disponible</div>
<div>NOUVEAU PATIENT</div> <div>NOUVELLE LESION</div> <div>NOUVELLE IMAGE</div>	<div>DETAIL DE LA LESION</div> <div>Aucune lésion sélectionnée</div> <div>Aucune lésion sélectionnée</div>

En cliquant sur un patient, le médecin affiche la listes des lésions du patient, puis en cliquant sur une lésion l'historique des observations de la lésion, pour finir en cliquant sur cette observation le médecin voit l'image de la lésion à la date choisie, ainsi que le diagnostic de cette lésion :

GESTION DES PATIENTS							
<div>LISTE DES PATIENTS</div> <div>Christophe Garreau</div>	<div>LESIONS DU PATIENT</div> <table border="1"> <thead> <tr> <th>Localisation</th> <th>Observation</th> </tr> </thead> <tbody> <tr> <td>bras gauche</td> <td>première observation</td> </tr> </tbody> </table>	Localisation	Observation	bras gauche	première observation		
Localisation	Observation						
bras gauche	première observation						
	<div>HISTORIQUE DE LA LESION</div> <table border="1"> <thead> <tr> <th>Date</th> <th>Chemin de l'image</th> <th>Diagnostic</th> </tr> </thead> <tbody> <tr> <td>2021-03-25</td> <td>./uploaded_pictures/ISIC_0000000.jpg</td> <td>Lésion Bénine</td> </tr> </tbody> </table>	Date	Chemin de l'image	Diagnostic	2021-03-25	./uploaded_pictures/ISIC_0000000.jpg	Lésion Bénine
Date	Chemin de l'image	Diagnostic					
2021-03-25	./uploaded_pictures/ISIC_0000000.jpg	Lésion Bénine					
<div>NOUVEAU PATIENT</div> <div>NOUVELLE LESION</div> <div>NOUVELLE IMAGE</div>	<div>DETAIL DE LA LESION</div> <div>  </div> <div>Lésion Bénine</div>						

D'autre part, en cliquant sur les boutons en bas à gauche de la page, le médecin peut à sa convenance ajouter un nouveau patient à sa liste, ajouter une lésion à un patient, ou ajouter une image à une lésion :



Pour un nouveau patient ou une nouvelle lésion, une fois le formulaire rempli, cliquer sur valider permet d'inclure les données du formulaire dans la base de données. Pour la nouvelle image en revanche, le médecin télécharge l'image de son choix via le bouton dédié, puis l'image s'affiche, et en-dessous de cette dernière la prédiction du modèle d'IA s'affiche. En cliquant sur *valider* le médecin copie l'image dans le dossier *./uploaded_pictures*, sauvegarde le chemin de cette image, ainsi que le diagnostic prédit, dans la base de données.

Qualité et Monitoring

Qualité

Débogage

Lors du développement de cette application un débogage continu a été réalisé, puisque le framework Dash le permet par le biais de la fonction suivante :

```
if __name__ == '__main__':
    app.run_server(host='0.0.0.0', debug=True, port=8051)
```

Toute modification du code est immédiatement visible sur le *localhost:8051* via un navigateur web après toute sauvegarde. De plus, en cas de problème, un message indique l'erreur et sa localisation. Ce débogage continu permet de vérifier directement le bon fonctionnement d'un nouveau bout de code et de le corriger immédiatement le cas échéant.

Tests de l'application

Néanmoins, en plus de ce débogage instantané, il est intéressant de tester plus globalement la robustesse du code produit. En effet, une fonction notamment peut fonctionner correctement avec les entrées utilisées mais ne pas être assez robuste pour fonctionner avec toutes les entrées qu'il serait possible de lui fournir. Ainsi parmi les différentes catégories de test, les tests unitaires permettent de tester individuellement chaque composant de l'application.

Pour mener à bien ces tests unitaires plusieurs librairies existent. Celle utilisée lors du développement de l'application Melanoma est *pytest*. Cette librairie permet de tester une fonctionnalité en créant une fonction qui appelle la fonction à tester, et en vérifiant si la sortie produite par la fonction est conforme à la valeur attendue.

Ainsi, par exemple, la fonction *transform_picture_to_predict* prend le chemin d'une image en argument, charge l'image, la transforme, et utilise le modèle d'IA pour prédire le caractère bénin ou malin de la lésion présente sur l'image. Elle retourne deux outputs : la classe *bénin* ou *malin*, et la probabilité de la prédiction. La fonction de test utilisée pour tester cette fonction est codée comme suit :

```
def test_transform_picture_to_predict():
    prediction_test_diag, prediction_test_proba = transform_picture_to_predict('./uploaded_pictures/ISIC_0000008.jpg')
    assert prediction_test_diag == "Lésion Bénine" and prediction_test_proba == 70.2
```

Le test est ensuite exécuté par la commande *pytest*. Le résultat du test se présente sous la forme suivante :

```
(project-env) PS C:\Users\Christophe GARREAU\Desktop\final_project> pytest
===== test session starts =====
platform win32 -- Python 3.7.4, pytest-6.2.2, py-1.10.0, pluggy-0.13.1
rootdir: C:\Users\Christophe GARREAU\Desktop\final_project
plugins: dash-1.19.0
collected 1 item

test_app.py .

===== warnings summary =====
test_app.py::test_transform_picture_to_predict
test_app.py::test_transform_picture_to_predict
test_app.py::test_transform_picture_to_predict
test_app.py::test_transform_picture_to_predict
test_app.py::test_transform_picture_to_predict
test_app.py::test_transform_picture_to_predict
C:\Users\Christophe GARREAU\AppData\Local\Programs\Python\Python37\lib\importlib\_bootstrap.py:219: RuntimeWarning: numpy.ufunc size
changed, may indicate binary incompatibility. Expected 192 from C header, got 216 from PyObject
    return f(*args, **kwargs)

-- Docs: https://docs.pytest.org/en/stable/warnings.html
===== 1 passed, 6 warnings in 1.22s =====
```

D'autres tests existent pour vérifier la qualité du code d'une application :

- tests d'intégration : vérifient l'intégration des différents modules entre eux
- tests fonctionnels : vérifient la conformité de l'application avec le cahier des charges
- tests de non régression : vérifient que des modifications n'ont pas affecté le fonctionnement de l'application
- tests IHM : vérifient que la charte graphique est respectée
- tests de configuration : vérifient l'impact des environnements d'exploitation sur le fonctionnement de l'application
- tests de performance : vérifient la capacité des serveurs et des réseaux à supporter

la charge

- tests d'installation : vérifient la documentation et la procédure d'installation

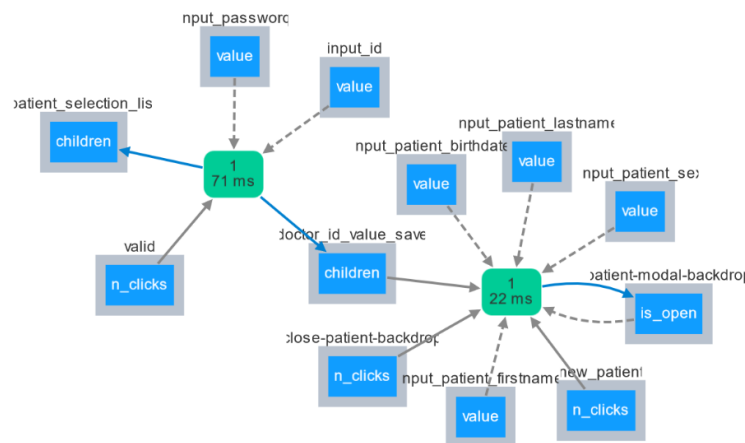
Journalisation

Le journalisation est une méthode de suivi des événements ayant lieu lors du fonctionnement d'une application. Des appels à l'outil de journalisation utilisé sont effectués dans le code pour indiquer qu'un événement a eu lieu.

Contrairement à la méthode la plus simple de journalisation qui est la fonction *print()* de Python, des méthodes plus élaborées, comme le module *logging* de Python, permettent de gérer des niveaux pour les différents événements, et même de sauvegarder dans un fichier la liste de tous les journaux (logs) générés.

Alors que la journalisation est utilisée pour la gestion des événements survenant lors de l'exécution d'une application, le monitoring assure une fonction similaire, si ce n'est qu'il permet de suivre de façon continue la performance des différentes fonctions de cette application.

Le framework Dash fournit graphiquement un outil de mesure des performances des différentes fonctions de l'application. Le rendu de ces performances se présente comme dans l'exemple ci-dessous :



Plus spécifiquement, un outil de journalisation spécifique au Machine Learning peut-être utilisé pour notamment suivre les performances et les métriques des différents modèles entraînés, et de déployer la version voulue du modèle le plus performant. Un outil qui peut être utilisé à cette fin est *MLflow*.

Axes d'amélioration

Les principaux axes d'amélioration de cette application seraient non exhaustivement les suivants :

- Améliorer le modèle d'Intelligence Artificielle
- Insérer une demande de validation de la prédiction par l'utilisateur
- Mettre en place une journalisation des événements
- Mettre en place une journalisation des modèles de Machine Learning via MLflow
- Généraliser les tests unitaires
- Optimiser l'affichage des informations recueillies de la base de données
- Mettre en place un Docker pour faciliter le déploiement de l'application web

Conclusion

Ce projet a été une occasion de mettre en pratique une grande partie des notions abordées au cours de cette formation Simplon en les intégrant dans un projet à visée médicale mettant en jeu une Intelligence Artificielle comme une aide à la décision.

En effet, de la collecte des données via une api, à la mise en place d'un visuel adapté, en passant par l'exploration et la sélection des données, la sélection des features à intégrer au modèle de Machine Learning, la sélection et la mise au point des modèles, la mise en place et l'exploitation d'une base de données, l'intégration d'une base de données à un Backend, et la connection de ce Backend à un Frontend, ce projet m'a permis d'utiliser et de manipuler concrètement les composants principaux d'une application web embarquant un modèle de Machine Learning, et ce d'un point de vue technique et pratique, ce qui ancre dans la réalité les notions essentielles, y compris celles parfois abordées superficiellement ou de façon trop abstraite.

De plus, confortant ainsi l'aspect du besoin pratique du client, ressenti lors de l'alternance en entreprise, la mise en œuvre de ce projet m'a permis d'envisager un problème côté client, en imaginant ses attentes pour y répondre au mieux, sans fonctionnalité inutile.

Pour conclure, même si cette application comporte encore des défauts et des lacunes, j'en suis fier, puisque je n'avais aucune compétence en matière de développement d'application web ni d'Intelligence Artificielle avant le début de cette formation, et que cette application est fonctionnelle.