

# Rapport de Projet : Neural Network And Learning

---

**Sujet :**

Construction d'un classificateur de photo de scènes  
naturelles

Yann MARTIN D'ESCRIGNE  
Yohann TOGNETTI

« Année universitaire 2020 - 2021 »

# Table des matières

<b>1</b>	<b>Présentation générale du projet</b>	<b>2</b>
<b>2</b>	<b>Descriptif du sujet et ses finalités</b>	<b>2</b>
2.1	Descriptif du sujet . . . . .	2
2.2	Les catégories . . . . .	2
<b>3</b>	<b>Description des données</b>	<b>3</b>
3.1	Le jeu de données . . . . .	3
3.2	Des données mal classifiées . . . . .	3
<b>4</b>	<b>Implémentation de l'algorithme</b>	<b>4</b>
4.1	preprocessing . . . . .	4
4.2	trainModel . . . . .	5
4.2.1	Description du modèle . . . . .	5
4.2.2	Evolution du modèle . . . . .	5
4.3	predict et méthodes graphiques . . . . .	6
4.4	Temps d'apprentissage du modèle . . . . .	6
<b>5</b>	<b>Résultat obtenus</b>	<b>7</b>
5.1	Résultat général . . . . .	7
5.2	Analyse détaillé des classes . . . . .	8
<b>6</b>	<b>Les améliorations possibles</b>	<b>10</b>
6.1	Travail sur le jeu de données . . . . .	10
6.2	Travail sur le modèle . . . . .	10
<b>7</b>	<b>Conclusion</b>	<b>11</b>
<b>8</b>	<b>Membres du groupe</b>	<b>11</b>
8.1	Yann MARTIN D'ESCRIBENNE . . . . .	11
8.2	Yohann TOGNETTI . . . . .	11
<b>9</b>	<b>Références</b>	<b>12</b>

# 1 Présentation générale du projet

Dans le cadre de notre cours de Neural Network and Learning, il nous a été demandé d'effectuer un projet impliquant une intelligence artificielle travaillant sur des images. Celle-ci a pour but de construire un classificateur de photo de scènes naturelles comme ceux qu'on pourrait intégrer dans un appareil photo intelligent qui à chaque prise de photo va taguer l'image avec le nom d'une catégorie qui y correspond.

## 2 Descriptif du sujet et ses finalités

### 2.1 Descriptif du sujet

Ainsi, pour entrer dans les détails le but qui est de "taguer l'image" correspond au fait de la catégoriser grâce à un classificateur entraîné possédant les catégories de scènes naturelles désirées. Pour cela avons dû utiliser un réseau neuronal convolutif étant donné que nous traitons des images. Une fois la réponse du classificateur reçue, elle correspond initialement à un entier, nous avons donc le devoir de le convertir en tag correspondant au nom de la catégorie trouvée.

L'affichage du résultat doit se faire à l'aide d'une méthode *predict* qui prend en entrée le chemin de fichier d'une image et une chaîne de caractères 'mode' qui renvoie le nom de la catégorie correspondant à l'image si **mode='category'** où un vecteur de probabilités si **mode='probabilities'** ou chaque élément du vecteur indique la probabilité que l'image appartienne à la catégorie correspondante.

### 2.2 Les catégories

Celles-ci sont au nombre de 6, chacune correspond à une catégorie de paysage ou de scène naturelle de la liste suivante :

- buildings
- forest
- glacier
- mountain
- sea
- street

*Building* correspond ainsi à des bâtiments comme une maison ou un immeuble qui se trouve généralement dans une ville.

*Forest* comme son nom l'indique correspond à un paysage forestier avec la présence ou non d'animaux.

*Glacier* quant à lui représente une montagne enneigée ou bien un glacier ou morceaux de glace de la banquise, cette catégorie est extrêmement corrélée avec *Mountain* car celle-ci correspond à une montagne pouvant parfois aussi être enneigé. Le découpage n'est pas toujours pertinent et encore plus ici, nous en parleront dans la section *Description des données*.

*Sea* représente les décors marins en général, que ce soit une plage, la mer avec ou sans terre ferme, ou bien même un décor sous-marin, tout cela va dans cette catégorie.

Enfin la catégorie *Street* met en avant une rue, qui peut donc facilement se rapprocher de *building* étant donnée la présence quasi-permanente de bâtiments dans une rue, on y trouve souvent des passant et des voitures.

## 3 Description des données

### 3.1 Le jeu de données

Les données utilisées pour pouvoir réaliser cet apprentissage viennent du « Intel Image Classification » de Google. Ce jeu de données est composé d'un total de 24.347 images, toutes de taille 150x150 pixels dans un format RGB. Le dossier principal est réparti en trois sous-dossiers, `seg_train`, `seg_test` et `seg_pred`.

Les dossiers de train et de test sont tous deux repartis en 6 sous-dossiers qui correspondent à nos classes (street, buildings, ...) et qui possèdent chacun les images correspondantes.

Le répertoire train possède en tout 14.046 images réparties avec des proportions plutôt équivalente comme nous pouvons le voir :

- Buildings 2.191 images
- Forest 2.271 images
- Glacier 2.416 images
- Mountain 2.512 images
- Sea 2.274 images
- Street 2.382 images

Le dossier de test possède lui un total de 3000 images, qui sont également réparties de façon équivalente. Évidemment il n'y a pas autant d'image que dans le set de train car il ne s'agit pas ici d'entraîner le modèle mais de valider ses résultats.

- Buildings 437 images
- Forest 474 images
- Glacier 553 images
- Mountain 525 images
- Sea 510 images
- Street 501 images

Comme nous avons pu le voir, les deux jeux de données possèdent un nombre important d'images pour chaque classe. Cela est très favorable pour pouvoir faire une classification correcte et un apprentissage sans under-fitting.

Le dernier répertoire est celui du test, il contient 7.301 images de toutes classes

confondues, il ne nous a donc pas été possible de tester réellement notre modèle ou bien même d'y faire des résultats graphiques. Il reste néanmoins une bonne source de test manuel avec des images que l'IA n'a jamais vues auparavant.

### 3.2 Des données mal classifiées

Au cours de la construction et l'apprentissage de notre modèle, nous nous sommes rendu compte que de nombreuses images ne se trouvaient pas forcément au bon endroit. En effet certaine image n'était pour nous, soit dans la mauvaise classe, soit complètement hors sujet et ne correspondait à aucune de nos catégories. Voici certains exemples qui illustrent nos propos :



FIGURE 1 – Image 20662.jpg, catégorie Mountain du jeu de test.

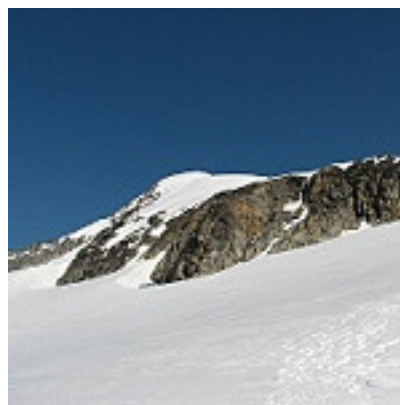


FIGURE 2 – Image 23589.jpg, catégorie Glacier du jeu de test.

Ces deux images n'ont donc pas la même catégorie et pourtant, elles se ressemblent fortement car ce sont des montagnes enneigées. Les catégories Mountain et Glacier sont toutes deux correctes, même si cela correspondrait peut-être plus à Mountain si on en suit le reste des images de Glacier. Mais pour l'IA qui un coup a appris à classer cette image en Mountain puis une similaire en Glacier, le travail devient extrêmement difficile et elle aura tendance à mettre l'image un peu au hasard dans une des deux catégories.

Un autre type de problème est les images hors contexte donc difficilement classifiables comme celle-ci :



FIGURE 3 – Image 1512.jpg, Catégorie Glacier du jeu de train.

Même pour nous humains, il n'est pas possible d'attribuer la catégorie Glacier à cette image et c'est le cas pour pas mal de données dans les catégories. Elles sont juste hors contexte. Certaines images vont donc automatiquement faire descendre la précision car l'IA ne sait pas quel patronne similaire aux images précédentes elle doit retrouver ici. Néanmoins, La plupart des autres images sont correctes et vont nous permettre de faire une bonne classification malgré ces quelques tâches dans le jeu de données.

## 4 Implémentation de l'algorithme

Notre algorithme est développé en python. Il se trouve dans le fichier *Program.py* et possède 3 méthodes majeures qui ont permis de répondre aux attentes du sujet. Nous allons donc présenter chacune de ces méthodes avec un peu plus de détails.

La réussite de ce sujet est grandement due au livre **L'APPRENTISSAGE PROFOND AVEC PYTHON** de *François CHOLLET*. La lecture de son travail nous a permis un apprentissage rapide du fonctionnement des réseaux neuronal convolutif ainsi que leur mise en application pour notre IA.

### 4.1 preprocessing

Cette méthode prend comme argument le directory qui contient les images avec la structure décrite dans la partie *Description des données* afin d'effectuer le pré-traitement.

Dans un premier temps cette méthode va donc créer un répertoire pour les données d'entraînement, puis un autre pour les données de validation en se basant sur le répertoire donné en argument.

Ensuite, deux *ImageDataGenerator* vont être créés avec une mise à l'échelle de 1/255. En effet il est beaucoup plus simple pour l'IA de travailler avec des données entre 0 et 1 que 0 et 255 concernant le RGB de chaque pixels. Le premier *ImageDataGenerator* va servir pour les données d'entraînement et le deuxième pour la validation.

Finalement deux générateurs vont être créés pour les deux parties avec comme taille voulue 150x150 pixel étant donnée que cela correspond à la taille des images dans le jeu de données. Le *batch\_size* est de **32** et le *class\_mode* est en **binary** étant donné que nous avons comme résultat un vecteur

constitué de 0 et de 1.

Les générateurs d'entraînement et validation ainsi créés vont être retournés par la méthode afin de les utiliser dans les autres.

## 4.2 trainModel

Cette méthode a pour but de créer et entraîner le modèle. Elle prend en argument les deux générateurs créés à la partie précédente, ainsi que des paramètres propres à l'entraînement de l'ia comme le nombre d'epoch, le nombre d'étapes par epoch ainsi que le chemin d'enregistrement du modèle qui sera créé.

### 4.2.1 Description du modèle

La construction de notre modèle actuel est comme ci-dessous :

conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
dropout (Dropout)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
dropout_1 (Dropout)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
dropout_2 (Dropout)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	0
dropout_3 (Dropout)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dropout_4 (Dropout)	(None, 6272)	0
dense (Dense)	(None, 512)	3211776
dropout_5 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 6)	3078

FIGURE 4 – Résumé du model

La partie principale de notre modèle se constitue donc de plusieurs blocs composés de la manière suivante : un **2D convolution layer** suivi d'un **Max pooling operation for 2D spatial data** et se terminant par un **dropout layer**. À la fin de ces paternes

un **flatten layer** est mis en place suivi d'un nouveau **dropout**. Notre modèle se termine enfin par deux **dense layer** avec un autre **dropout** entre eux

Les convolutions layers utilisent tous de strides de taille 3x3 et ont pour fonction d'activation *Rectified Linear Unit*. La première couche utilise comme input\_shape des images RGB de taille 150x150 afin de s'adapter à nos données.

Les maxpooling layers ont quant à eux un strides de taille 2x2.

Tous les dropout mis en place dans ce modèle ont comme paramètre 20% des données abandonnées. Ceci afin de limiter l'overfitting de nos données.

Le premier dense layer prend comme paramètre 512 dimensions avec comme fonction d'activation *Rectified Linear Unit* et utilise également un *kernel\_regularizer*. Le deuxième quant à lui possède 6 dimensions, étant donné que l'on souhaite 6 classes et utilise *softmax* comme fonction d'activation.

La compilation du modèle se fait avec un optimiser *adam* et une fonction de perte *categorical\_crossentropy* et se focalise sur la précision du modèle.

Finalement nous entraînons le modèle avec les paramètres donnés précédemment et le stockons dans history afin d'afficher sur un graphe l'évolution de la précision au cours de l'entraînement. Nous sauvegardons au passage le modèle dans le chemin donné en argument.

### 4.2.2 Evolution du modèle

Grâce aux nombreux exemples du livre qui nous a servis de base, notre premier modèle possédait déjà une précision aux alentours de 80% pour les données de validation et 98% pour celle d'entraînement. Ceci était déjà un très bon résultat mais notre modèle souffrait donc d'**overfitting**.

Nous avons remédié au problèmes en utilisant des dropout layers afin que notre mo-

dèle ne se focalise pas trop sur ses données d'entraînement uniquement. Nous avons choisi d'abandonner 20% de nos résultats après chaque paternes conv2D/maxpooling. On peut ainsi constater la différence de précision des données de validation dans les graphiques suivants :

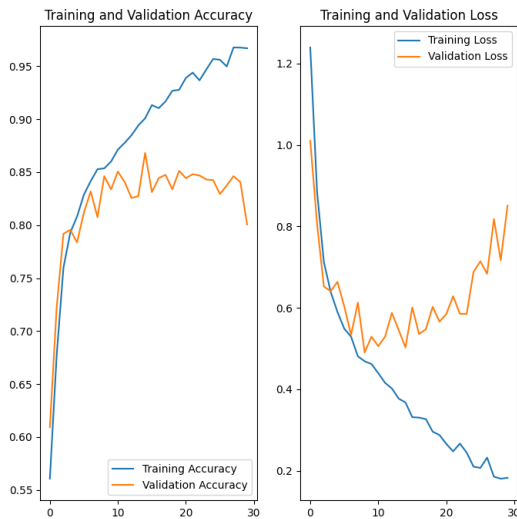


FIGURE 5 – Précision sans dropout

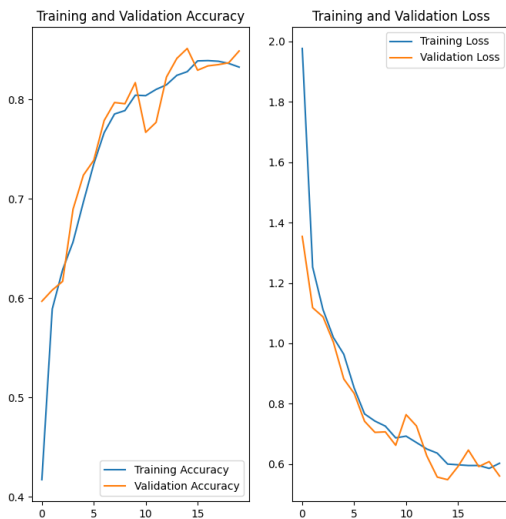


FIGURE 6 – Précision avec dropout

On voit alors très clairement que la précision de validation est beaucoup plus proche

de celle d'entraînement même si celle-ci a donc tendance à être plus basse, s'expliquant par l'abandon de 20% des résultats obtenus.

### 4.3 predict et méthodes graphiques

Cette méthode correspond aux attentes du sujet pour afficher les prédictions de notre modèle avec le mode attendu qu'il prend donc en argument. Il prend également pour paramètre le modèle entraîné ainsi que le chemin de fichier de l'image à prédire.

Enfin, pour mettre sous forme de diagramme ou de graphiques les résultats obtenus qui seront présentés dans la section suivante, nous nous sommes appuyé sur la librairie matplotlib. Nous avons donc mis en place différentes méthodes pour ceci, ainsi qu'un autre programme appelé *report.py* pour l'affichage graphique de la matrice de confusion.

### 4.4 Temps d'apprentissage du modèle

Le temps d'apprentissage du modèle dépend bien évidemment du nombre d'epoch ainsi que du nombre d'étapes par epoch choisi. Notre modèle final se base sur 60 epochs de 430 étapes chacune. Bien évidemment lors de nos tests d'améliorations, nous ne mettions que 30 epochs de 120 étapes afin de voir rapidement les résultats.

Sur le pc le plus performant du groupe, le modèle final prend environ 15-20 min pour s'entraîner et lors de nos tests environ 2-3 minutes. Cela vient du fait de l'utilisation de l'installation des dll utilisant la carte graphique et cœurs cuda pour l'apprentissage de l'IA. Néanmoins ce temps dépend bien sur fortement du matériel utilisé et de sa configuration.

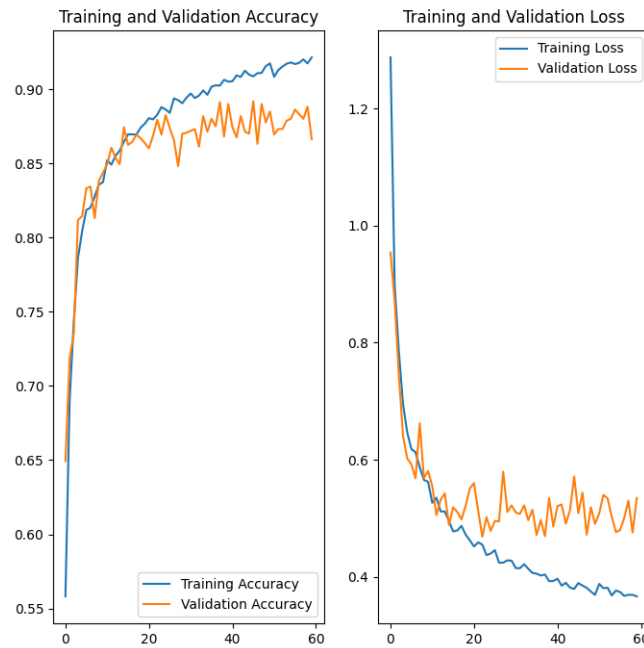


FIGURE 7 – Précision et perte pour les données de validation et training

## 5 Résultat obtenus

Il est à noter que les résultats peuvent être légèrement différents à chaque entraînement du programme ainsi les graphiques qui vont suivre ne sont pas forcément absolus.

pour les données de validation. La précision de l'entraînement elle continue d'augmenter entraînant donc un overfitting.

### 5.1 Résultat général

Dans un premier temps nous allons analyser les résultats de notre modèle en général. On constate facilement grâce à la figure 7 que notre modèle possède une précision d'entraînement de 95% tandis que la précision des données de validation est plus proche de 87%. On constate donc un léger overfitting dans notre modèle que même les dropout n'ont pas pu améliorer.

Ces résultats sont tout de même satisfaisants, ce qui n'est pas étonnant en vue du nombre de données pour l'entraînement. On pourra également constater qu'après 30 epochs, le modèle commence à se stabiliser et ne monte que très légèrement en précision



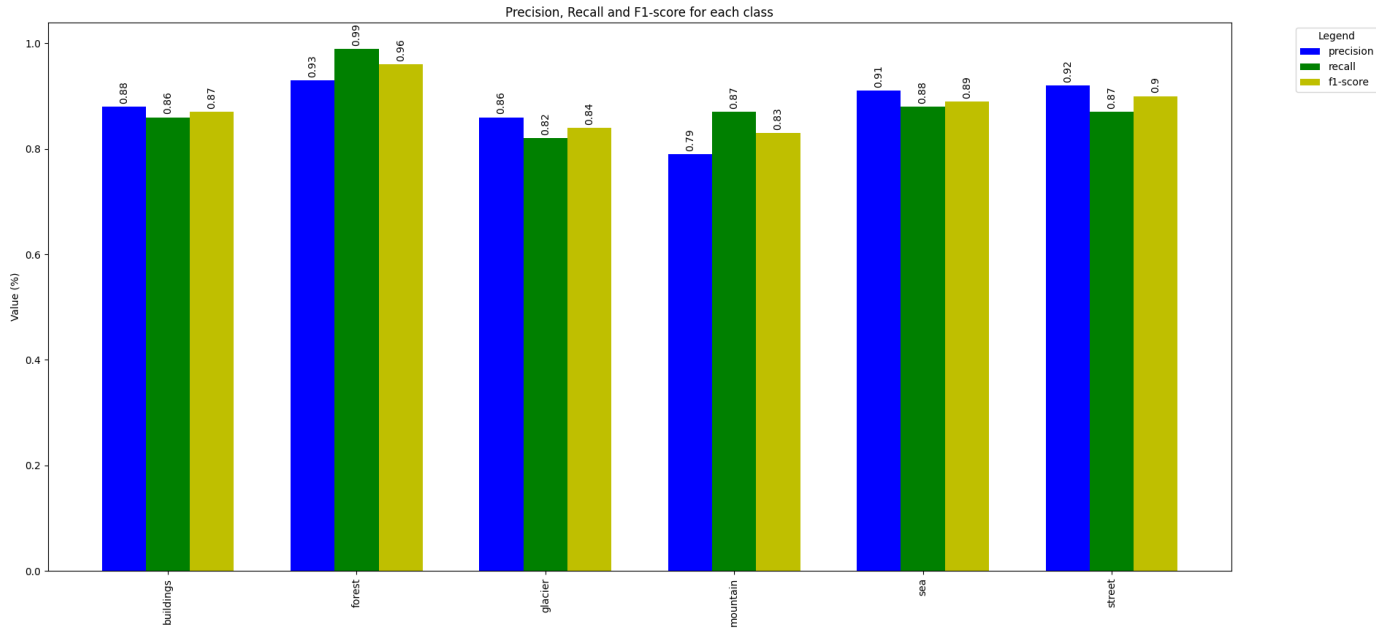


FIGURE 8 – Statistique général de chaque classe

## 5.2 Analyse détaillé des classes

Nous avons ensuite décidé de regarder les scores des différentes classes séparément. Au niveau de la « precision », le « recall » et « f1-score », nous avons une moyenne de 88% ce qui est un score plutôt satisfaisant.

Une fois décomposé, voici un graphique reprenant les différentes catégories. (figure 8)

Ce graphique nous permet pour chaque classe de voir comment l'IA a réussi à les classifier. Nous avons ainsi choisi 3 score : le recall, la precision et le f1-score.

Le f1-score est un mixte du recall et de la précision et est donc élément le plus favorable à regarder pour un esprit de synthèse. Tout d'abord, nous pouvons voir que la classe « forest » est la plus facile à repérer pour l'IA avec un f1-score de 96% ce qui est plus qu'acceptable. De plus, il arrive rarement que l'on assigne cette classe à une image qui n'est pas une forêt (« recall » à 99%). La classe « mountain » est la classe qui a le moins bon f1-score mais qui reste plus qu'acceptable étant donné qu'il est à 83%. Cela s'explique

par la similarité de certaines images entre glacier et montagne, on constate d'ailleurs que glacier possède des scores similaires ce qui justifie nos propos.

Maintenant, nous allons voir plus précisément les erreurs de notre IA. Pour cela nous utiliserons le graphique en camembert suivant (figure 9) montrant la matrice de confusion sous forme graphique pour chaque catégorie.

Si nous regardons la classe buildings, nous pouvons voir que les plus grandes erreurs viennent principalement de la classe Street avec 8.01% des éléments qui ont été classés en street alors qu'ils devaient être classés en building. Nous retrouvons aussi l'inverse avec 8.18% en buildings alors qu'ils devaient être en street. Cela peut s'expliquer facilement. En effet, les deux classes représentent généralement des villes, sur une même photo il peut y avoir une rue et un bâtiment. Dans ce cas-là il peut être difficile à l'IA de pouvoir dire si la photo veut montrer principalement une rue ou bien les bâtiments qui l'entourent. La deuxième comparaison que nous pouvons

faire est celle entre « glacier » et « mountain ». Pour ces deux classes, il y a un pourcentage de classification en montagne de 13.02% alors que ce sont des glaciers et de 9.33% pour l'inverse. Cela est dû à notre jeu de données. Comme vu précédemment (cf. 3.2), il y a beaucoup d'images sans glace qui sont considérées comme des glaciers et d'images de montagnes gelées qui peuvent être trouvées dans le jeu de données de ces deux catégories.

En ce qui concerne la classification « sea »,

nous pouvons voir que certaines photos où l'on voit de l'eau comme des lacs, la mer ou encore des rivières. Il est probable que la probabilité de 6% classifier comme une montagne est due à ce type d'image mais il doit être possible d'améliorer celles-ci.

Pour finir, la classe « forest » qui avait des résultats vraiment satisfaisant se remarque facilement sur la catégorie qui s'y rapporte car la bonne classification s'effectue à 98.52% avec donc un très faible pourcentage des autres classes.

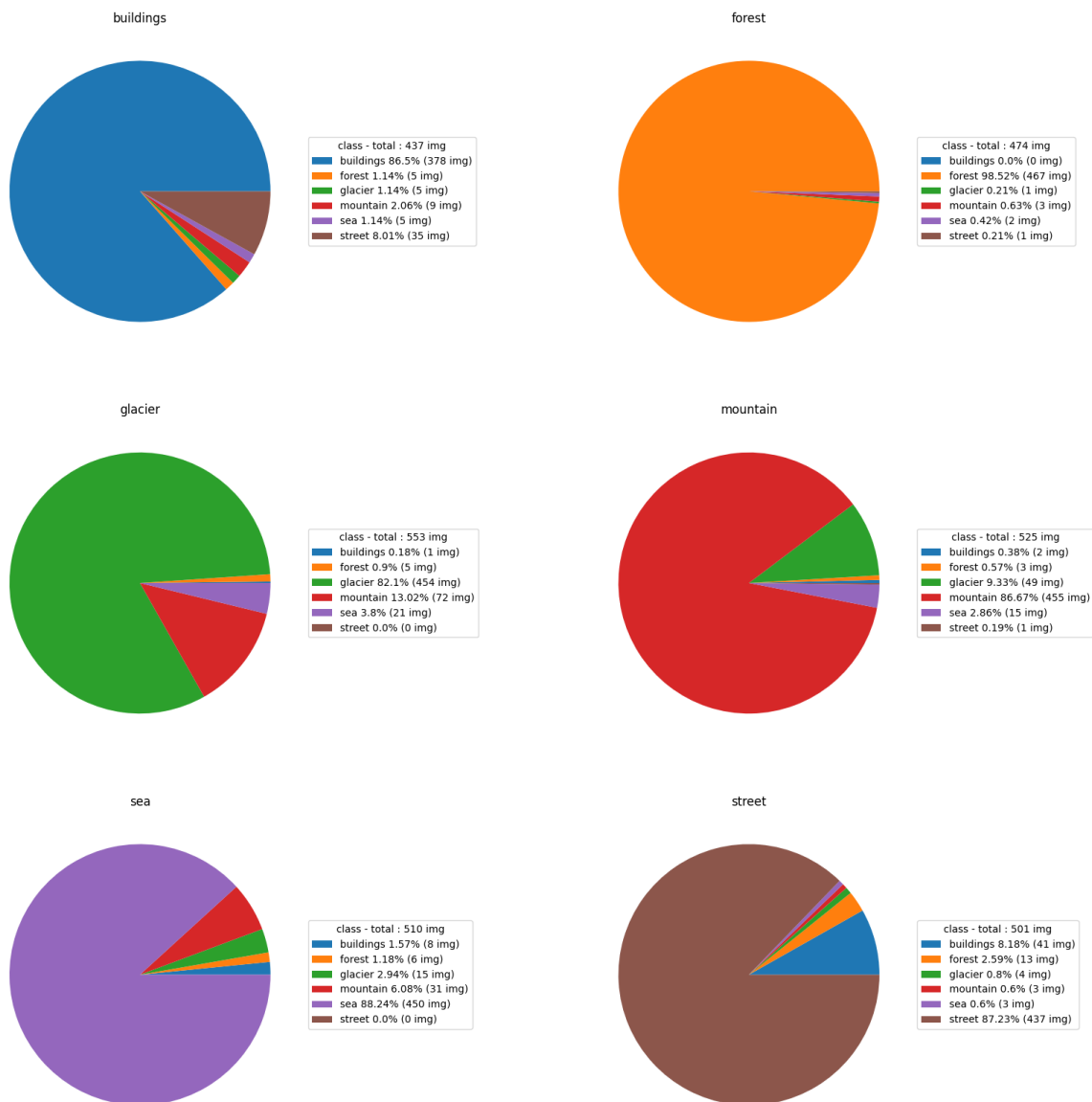


FIGURE 9 – matrice de confusion de chaque classe

## 6 Les améliorations possibles

Globalement, pour nous ce projet n'a pas tant d'amélioration à faire que cela. Les nombres de données pour construire le modèle sont largement suffisants, il n'y a donc pas eu à recourir à des transformations d'images pour lutter contre un manque de données. Elles sont plutôt diverses et équilibrées, il ne faut donc pas mettre en place de système de pondération pour les classes.

### 6.1 Travail sur le jeu de données

Comme expliqué précédemment, le jeu de données met certaines images dans des

classes qui n'ont pas toujours un réel rapport ou alors en met d'autres quasiment identiques dans des classes différentes. Résoudre ce problème en supprimant ou déplaçant toutes ces images problématiques dans les bonnes catégories permettrait alors à l'IA de mieux savoir différencier ces différentes images et ainsi obtenir une meilleure précision.

### 6.2 Travail sur le modèle

Bien évidemment, notre modèle est loin d'être parfait. Par manque de temps, nous n'avons pas expérimenté toutes les possibilités et d'autres méthodes d'apprentissage de classifications d'images existent. Notre précision ( 87%) reste néanmoins plus que correcte, mais il aurait peut-être été possible d'atteindre 90%.

## 7 Conclusion

Pour conclure sur notre travail sur le sujet donné, l'apprentissage et la construction du modèle de fût pas le plus difficile à mettre en place et obtenir rapidement une bonne précision.

Cela s'explique par le très bon livre donné en référence de François CHOLLET, celui-ci contenait toutes les informations nécessaires à la réussite du sujet mais aussi à l'apprentissage profond neuronal en général. Certaines solutions pour lutter contre l'overfitting y étaient même présentes. Il nous a ainsi fallu compléter toutes ces informations par quelques recherches supplémentaires sur internet afin d'obtenir notre résultat final.

La classification d'image étant quelques choses que nous n'avions encore jamais abordées, nous avons donc pu constater que ce n'est pas excessivement compliqué. À présent, nous avons le savoir minimal nécessaire afin d'étendre la classification d'image à n'importe quels problèmes pouvant être assimilé à ce projet, la seule différence étant le jeu de données et les catégories différentes.

## 8 Membres du groupe

### 8.1 Yann MARTIN D'ESCRIGNE

Ce projet m'a beaucoup apporté. Il est pour moi une bonne finalisation de tous les TPs que nous avons pu faire en Neural Network and Learning. Le confort du bon jeu de données m'a permis de mieux m'investir dans le projet, malgré quelques surprises dans certaines catégories.

Nous avons travaillé en groupe quant à l'implémentation du modèle mais également dans tout le projet en général, avançant ensemble de façon synchrone sur le projet. J'ai majoritairement travaillé sur la phase de preprocessing et de prédiction. J'ai également recherché de nombreuses informations dans le livre de François CHOLLET afin de pouvoir répondre à toutes nos questions et attentes du sujet. Je me suis occupé de rédiger les parties 1,2,4,6 et 7 du rapport.

### 8.2 Yohann TOGNETTI

Ce projet m'a beaucoup appris sur le fonctionnement des réseaux neuronaux ainsi que de leur implémentation. Grâce à ce projet, ainsi que les TP réalisés, je suis maintenant capable de pouvoir classifier les images. Au niveau des tâches, le travail a été séparé de manière plutôt équitable, cela nous a permis à tous les deux de pouvoir apprendre les diverses étapes à faire pour la classification.

L'une des tâches où j'ai principalement travaillé a été la prédiction ainsi que la partie de génération de traitement des résultats pour avoir une meilleure représentation de nos évolutions. Au niveau du rapport final, j'ai principalement rédigé les parties 3 et 5 qui traitent sur les données ainsi que les résultats.

## 9 Références

- Deep Learning with Python by François Chollet
- TensorFlow Conv2D Layers : A Practical Guide :  
<https://missinglink.ai/guides/tensorflow/tensorflow-conv2d-layers-practical-guide/>
- Keras documentation :  
<https://keras.io/api/layers/>
- Overfit and underfit | TensorFlow Core :  
[https://www.tensorflow.org/tutorials/keras/overfit\\_and\\_underfit#strategies\\_to\\_prevent\\_overfitting](https://www.tensorflow.org/tutorials/keras/overfit_and_underfit#strategies_to_prevent_overfitting)
- Image classification | TensorFlow Core :  
<https://www.tensorflow.org/tutorials/images/classification>
- WIKIPEDIA FR :  
[https://fr.wikipedia.org/wiki/Wikip%C3%A9dia:Accueil\\_principal](https://fr.wikipedia.org/wiki/Wikip%C3%A9dia:Accueil_principal)