

Projet de Programmation

Visualisation et analyse des filtres dans les réseaux de neurones convolutionnels

Zoe Debaty, Mamadou Saliou Diallo, Mohamed Diallo,
Manuel Guevara Garban, Hugo Lecomte

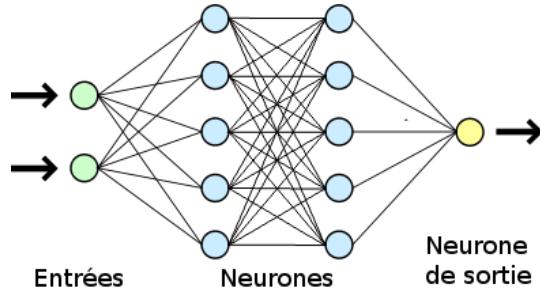
7 novembre 2021

Table des matières

1 Contexte	3
1.1 Réseaux des neurones	3
1.2 Réseau de neurones à convolution	3
1.3 Résumé	4
1.4 Analyse de l'existant	4
1.4.1 Le traitement de "feature map" de J.Yosinski	4
1.4.2 La recherche de la meilleure image d'activation de F.Chollet	5
1.4.3 tf-keras-vis	7
1.4.4 Différences avec notre projet	8
1.4.5 Algorithme au coeur du projet	8
2 Besoins	9
2.1 Besoins Fonctionnels	9
2.1.1 Gestion du modèle	9
2.1.2 Gestion du jeu de données	13
2.1.3 Interface graphique	14
2.1.4 Non prioritaire	20
2.2 Besoins non fonctionnels	20
3 Architecture et logiciel	21
3.1 Exemples de bon fonctionnement	21
3.2 Schémas	28
3.3 Bibliothèques externes	29
4 Analyse du fonctionnement et tests	29
4.1 Analyse du fonctionnement	29
4.2 Tests	30
4.2.1 Description des tests	30
4.2.2 Discussion des résultats	30
4.3 Problèmes, défauts, bugs	30
5 Conclusion	31
5.1 Améliorations	31

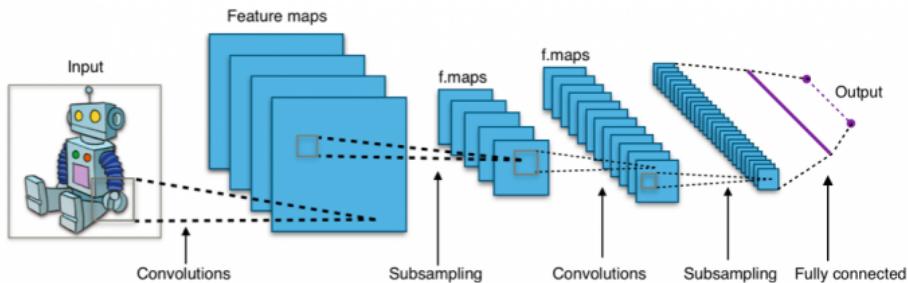
1 Contexte

1.1 Réseaux des neurones



Les réseaux de neurones sont une classe d’algorithmes propres au domaine de l’intelligence artificiel. En s’inspirant du fonctionnement du cerveau et des interactions entre les vrais neurones, ces algorithmes sont particulièrement efficaces dans le domaine de la classification par exemple.

1.2 Réseau de neurones à convolution



Les réseaux de neurones à convolutions sont des réseaux de neurones spécialisés dans la reconnaissance d’images. Par l’application de filtres reconnaissant des éléments de l’image de plus en plus précisément à mesure que l’on progresse dans les couches.

Les réseaux de neurones à convolution est un réseau de neurone avec des couches de convolution. Il va appliquer des opérations de filtrage par convolution et renvoyer des "feature map" sur lesquelles on appliquera des transformation (normalisation, redimensionnement, etc) afin d’en observer le contenu. Les différentes feature map misent dans un vecteur représentent la sortie du premier bloc, et l’entrée du second (voir schéma)

1.3 Résumé

Dans ce projet de programmation nous allons proposer un outil pour mieux comprendre le fonctionnement des réseaux de neurones convolutionnels à partir des informations contenues dans les couches impliquées dans la construction des filtres du réseau. Cet outil permettra de réaliser de petites expériences sur les réseaux de neurones à convolution, en particulier grâce à un générateur de formes géométriques simples et une interface de création et d'entraînement de réseau de neurones à convolution.

Notre objectif sera ainsi d'afficher l'évolution, couche par couche, des filtres d'un réseau à convolution auquel on a passé une image spécifique en entrée mais aussi les images qui activent le plus les filtres des couches du réseau.

Une interface graphique sera mise en place pour lier le tout.

1.4 Analyse de l'existant

1.4.1 Le traitement de "feature map" de J.Yosinski

Dans cet article [1], J.Yosinski cherche à comprendre les réseaux de neurones au travers de la visualisation de ce qu'il se passe dans les différentes couches. La partie qui nous concerne le plus est la "3. Visualizing via Regularized Optimization", elle correspond à ce que nous cherchons à faire dans ce projet de programmation.

Ici, J.Yosinski cherche à optimiser l'image qu'il reçoit des "feature map" en appliquant plusieurs modifications sur celle-ci, chose que nous devrons aussi implémenter dans notre programme.

La première étape est de faire une régularisation des données, couplée à une montée de gradient :

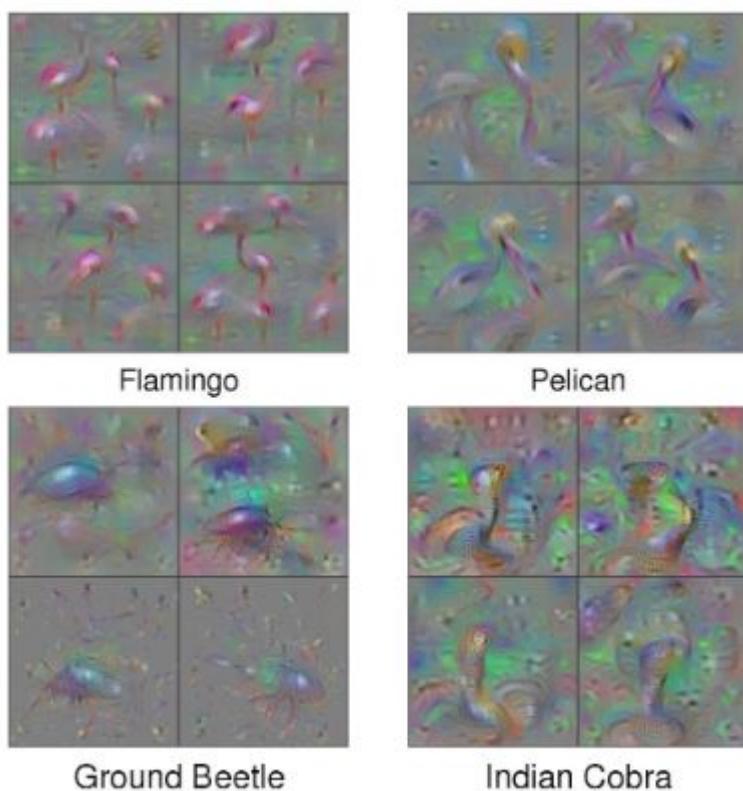
$$x = r_\theta(x + \eta \frac{\delta a_i}{\delta x})$$

Il applique ensuite ce qu'il appelle un L_2 decay, une régularisation qui pénalise les valeurs trop larges, ce qui permet d'éviter qu'un petit nombre de pixels aux valeurs extrêmes prennent le dessus sur le reste de l'image.

Un filtre gaussien est ensuite appliqué. Il explique qu'avec la montée de gradient, les images produites ont des fréquences trop haute, et qu'elles ne sont donc pas réalistes et interprétables. Le flou permet donc de lisser l'image.

Il explique que, bien que les deux dernières étapes suppriment les hautes amplitudes ainsi que les hautes fréquences, ils laissent des pixels avec des normes très faible, ne laissant pas paraître de manière précise le sujet principal de l'image. Il va donc prendre les pixels avec des normes faibles (par rapport à la moyenne des normes des pixels de l'image) et mettre leur norme à 0.

Il obtient donc à la fin :



Exemple de rendu de "feature map" par J. Yosinski

Cet article décrit les étapes d'amélioration de l'image que nous allons devoir implémenter, il est donc la base de notre projet de programmation.

1.4.2 La recherche de la meilleure image d'activation de F.Chollet

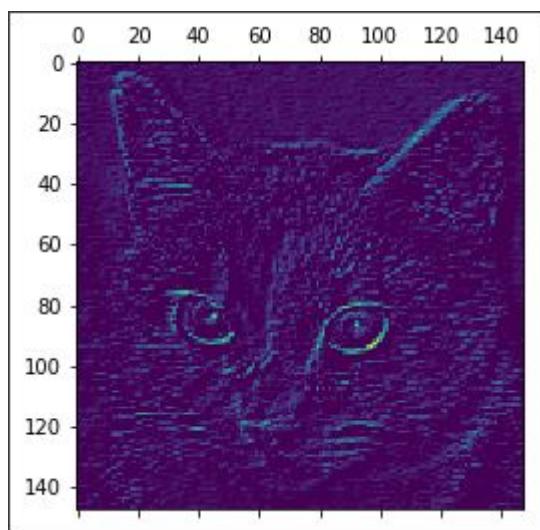
F.Chollet, dans son livre "Deep Learning with Python" [2] cherche à comprendre comment les réseaux de neurones traitent les images en entrée. Pour notre projet, nous nous intéresserons à la partie "Visualizing what convnets learn" [3] (une version google collab est accessible depuis le site) .

Visualiser les activations intermédiaires

La première chose qu'il cherche à faire, est d'afficher les "feature map" en sortie des couches de convolution et de pooling.

Après avoir chargé un modèle entraîné, il va créer un modèle d'activation (qui prend des images en entrée, et donne l'activation des couches de convolution et de "pooling" en sortie) afin d'avoir une sortie par couche (chose qu'il ne pouvait pas faire avec son modèle de base).

En récupérant le tableau de retour de la prédiction, il peut le parcourir et afficher les images qui y sont stockées.



Exemple de "feature map" pour la couche 3 de son réseau

Il conclut avec quelques remarques sur l'analyse de l'image par le réseau :

- La première couche correspond à une détection d'angle.
- Plus les couches sont profondes, plus les images paraissent abstraites.
- Plus on va profondément, moins les filtres sont activés, cela signifie que les images ressemblent de moins en moins à celles d'entraînement.

Cela nous permettra de comparer avec nos résultats à la fin du projet.

Visualiser les filtres

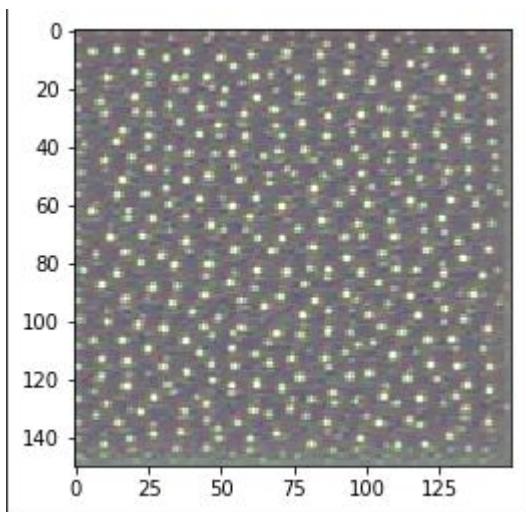
Une autre manière de comprendre le fonctionnement des réseaux de neurone est de chercher l'image qui active le plus les couches du réseau.

La première étape est de récupérer une fonction de perte qui maximise la valeur du filtre donné dans une couche de convolution donnée.

Il va faire une descente de gradient pour ajuster les valeurs de l'image de sortie afin de maximiser la valeur d'activation. Il précise que pour avoir une descente de gradient fluide, il régularise la valeur du gradient en le divisant par sa norme L2 :

$$\text{NormeL2} = \sqrt{\frac{1}{n} \sum_{k=1}^n x_k^2}$$

En faisant la descente de gradient de manière stochastique dans une boucle et en ramenant le résultat dans l'intervalle [0,255], il obtient des images traitables.



Exemple d'image obtenue par F.Chollet

Ce programme est adaptable afin de voir les images de chaque filtre de chaque couche du réseau. En observant toutes les images des différents filtres, il se rend compte que plus les couches sont profondes, plus les images récupérées sont complexes.

Ce document, ainsi que le google collab associé, seront très pratique dans la compréhension mais surtout le codage de notre programme. En effet F.Chollet code en Python, langage que nous utiliserons aussi, et il partage son code qui lui a permis d'arriver aux résultats montrés.

1.4.3 tf-keras-vis

tf-keras-vis est un outil utilisé pour étudier les réseaux de neurones et permet, entre autres, la visualisation les filtres à convolution, les couches

denses, les cartes de chaleur, etc... Il s'agirait dans notre projet de proposer une implémentation s'inspirant en partie de cet outil, mais sous une forme bien moins ambitieuse en portant une attention particulière sur la méthode de montée du gradient pour visualiser les filtres des couches à convolution.

1.4.4 Différences avec notre projet

- Dans les deux existant (et dans la majorité des articles trouvables sur internet), ils utilisent des réseaux de neurones qui reconnaissent des images complexes (chat, chien, voiture, etc). Hors, pour notre projet de programmation, nous allons nous limiter à un réseau de neurone qui reconnaît des formes simples (carré, cercle, triangle, etc).
- Une interface graphique sera faite de notre côté, ce qui permettra de regrouper en un endroit toutes les fonctionnalités de notre programme.

1.4.5 Algorithme au coeur du projet

Nous allons implémenter la technique de montée de gradient utilisée par Jason Yosinski et décrite dans son article [1]. Cette technique nous permettra de générer l'image qui maximise la fonction d'activation d'un filtre donné dans le réseau de neurones à convolution.

Tout d'abord, on considère une image $x \in \mathbb{R}^{CxHxW}$ où $C = 1$ canal de couleur (car, dans ce projet, on va s'intéresser à l'analyse des images monochromes), la hauteur (H) et la largeur (W). Quand cette image est analysée par le réseau de neurones, cela va causer une activation $a_i(x)$, pour une unité i qui sera l'index d'une boucle qui parcourra la totalité des couches du réseau. Ensuite, comme dans l'article [1], nous allons définir une fonction de régularisation paramétrique (notée $R_\theta(x)$) qui va pénaliser l'image dans plusieurs aspects. (cf [Analyse de l'existant](#)).

On cherche alors à trouver l'image x^* qui maximise la fonction d'activation d'un filtre cible, définie comme :

$$x^* = \arg \max(a_i(x) - R_\theta(x)) \quad [1]$$

Pour trouver cette image on va implémenter une montée de gradient en alternant entre faire un pas vers le gradient de $a_i(x)$ et un pas dans la direction donnée par r_θ (avec un pas de gradient de taille η).

$$x = r_\theta(x + \eta \frac{\delta a_i}{\delta x}) \quad [1]$$

Ensuite, comme on a vu dans l'[l'expérience réalisée par François Chollet](#), nous allons appliquer une étape de centrage sur l'image.

$$\begin{aligned}\mu &= \left(\frac{1}{n} \sum_{i=1}^n x_i\right) \\ \sigma &= \sqrt{\frac{\sum(x-\mu)^2}{n-1}} \\ x' &= x - \mu \\ x'' &= \frac{x'}{\sigma}\end{aligned}$$

L'algorithme prendra donc en entrée un réseau des neurones convolutionnel et donnera en sortie un ensemble d'images qui maximiseront au plus la fonction d'activation de chaque filtre du réseau des neurones.

2 Besoins

2.1 Besoins Fonctionnels

Afin d'identifier au mieux nos besoins fonctionnels nous avons décidé de découper ces derniers en 3 parties essentielles.

Tout d'abord [la gestion du modèle d'apprentissage](#) (Réseau de neurones à convolution), correspondant au chargement, à la sauvegarde, la création et au paramétrage du modèle. Ensuite nous abordons une partie de [gestion du jeu des données](#), où nous allons détailler les besoins requis pour réaliser l'[expérience de base](#).

2.1.1 Gestion du modèle

Sur cette partie nous avons regroupé les besoins liés au modèle d'apprentissage, qui est dans notre cas un **Réseau des neurones à convolution**.

1. Charger modèle pré-entraîné

- 1.1. Vérifier les tailles des images en entrée ainsi que des filtres pour assurer la compatibilité du modèle 2D.
- 1.2. Lire et charger en mémoire un modèle écrit dans un fichier de type .h5, .keras, .pb ou .model. [4] [5]

Test : Vérifier le type du modèle Keras, l'existence du fichier et l'intégrité de modèle (modèle non corrompu et donc utilisable pleinement par l'application).

- 1.3. Charger un modèle depuis le code via un import Keras, qui sera encapsulé par notre framework.

Test : Assurer de définir les champs concernant l'objet modèle comme des champs privés. (notation `__` en python)

- 1.4. Générer un modèle de visualisation avec Keras à partir des couches du modèle entraîné permettant d'afficher les "features maps" et les filtres contenus dans le réseau.

Modèle de visualisation : Pour extraire les "features maps" à examiner, nous devons créer un modèle Keras, qui prend des lots d'images ("batches") en entrée et génère en sortie l'activation de toutes les couches de convolution et de max "pooling". Pour ce faire nous devons instancier un modèle de visualisation avec deux arguments : un "tenseur" (ou une liste de "tenseurs") d'entrée et un tenseur de **sor-tie**, le résultat sera un modèle Keras classique avec la différence que ce dernier, permet d'avoir plusieurs sorties, contrairement au modèle "Sequential" de Keras. [4]

Test : Vérifier que la structure du modèle soit valide en fonction du modèle de visualisation, vérifier de prendre les bonnes sorties à chaque fois.

2. Charger l'image à faire passer dans le réseau à convolution

- 2.1. Lire et sauvegarder des fichiers images encodés en format .jpg, .png, .jpeg

Test : Vérifier que l'image existe dans le disque dans le bon type, vérifier la conversion.

- 2.2. Charger depuis la mémoire en tant que ndarray.

Test : Vérifier que l'image est bien de type ndarray et qu'elle contienne les bonnes dimensions 2D prises en compte par le logiciel.

- 2.3. Re-dimensionner l'image en entrée dans le réseau en faisant un réduction de l'échelle de l'image afin de la rendre compatible avec le réseau (exemple : une image de taille 1920 x 1080 a besoin d'un réduction de l'échelle vers 250x250 si le réseau prend des images en 250x250).

Test : Vérifier qu'en faisant cette réduction on ne perd pas des informations importantes de l'image.

3. Extraire les "features maps" pour représenter les couches de convolution uniquement (les couches "fully connected/flatten" ne seront pas prises en compte). [1]

- 3.1. Trouver l'image qui génère l'activation la plus pertinente pour chaque filtre en appliquant une montée de gradient et en maximisant la réponse d'un filtre spécifique.
- 3.1.1. Développer l'algorithme de montée de gradient pour trouver les images qui activent le plus les filtres. Passer par l'interface Keras pour exécuter les fonctions dans l'environnement de Keras et pouvoir bénéficier de l'exécution parallèle de ce framework. [3]
- 3.1.2. Créer une image grise avec un peu de bruit aléatoire qui sera utilisée par la montée de gradient, le résultat sera l'image en entrée qui devrait maximiser la réponse d'un filtre en particulier. [3].
- 3.1.2. Régulariser l'image résultante de la montée de gradient afin de la rendre compréhensible. [1]
- 3.1.3. Régulariser l'image en divisant le gradient par sa norme L2 :

$$\text{NormeL2} = \sqrt{\frac{1}{n} \sum_{k=1}^n x_k^2}$$

Cela garantit que l'ordre de grandeur des mises à jour effectuées sur l'image d'entrée est toujours dans une même plage, attention à ajouter un epsilon 1.e-5 afin d'éviter une division par zéro. [3][1][2]

- 3.1.4. Centrer les valeurs de l'image en soustrayant la moyenne de cette dernière et ensuite diviser par l'écart type. [3][2]
- 3.1.5. Placer l'image dans l'intervalle [0,255].

Test : Concernant la partie 2.1.1 nous devons tout d'abord montrer que nous avons les mêmes résultats que dans l'expérience réalisée par François Chollet (créateur de Keras) sur cette méthode [3]. Cela sert à vérifier que la taille de l'image synthétique créée respecte les bonnes dimensions compatibles avec le filtre cible.

- 3.2. Faire passer l'image à travers le réseau pour récupérer les "features maps" de chaque filtre en parcourant les couches du modèle.
- 3.2.1. Parcourir les canaux du "feature map" et les afficher via l'interface graphique.

- 3.2.2. Placer l'image dans l'intervalle [0,255].
- 3.2.3. Seuiller l'image en noir et blanc pour faire ressortir les détails de cette dernière.
- 3.3. Sauvegarder les images obtenues suite à l'extraction des "features maps" au format .zip.

Test : Vérifier l'intégrité du fichier .zip ainsi que du bon seuillage des images.

4. Créer nouveau modèle simple à entraîner où on peut faire varier le nombre de filtres, couches et couche d'activation

- 4.1. Créer nouveau modèle depuis zéro avec l'interface Keras.
- 4.2. Paramétriser le nombre de couches avec fonctions d'activation, taille des filtres, fonction d'initialisation des filtres.
 - 4.2.1. Paramétriser le type de couche (Conv2D, MaxPooling, Dropout, Flatten, Dense).
 - 4.2.2. Paramétriser le nombre de filtres de forme carrée dans chaque couche ainsi que la taille des filtres (exemple : 3x3, 5x5, etc).
 - 4.2.3. Paramétriser la fonction d'initialisation des filtres (exemple : normal, gaussian, random, etc). [6]
 - 4.2.4. Choisir la fonction d'activation de chaque couche (exemple : ReLU, Softmax, Tanhn, Sigmoid, etc).
- 4.3. Charger un jeu de données déjà existant depuis le disque avec Keras ou créer le sien pour l'expérience (cf : [Gestion de jeu de données](#)).
- 4.4. Définir une fonction de loss pour afficher des métriques.
- 4.5. Sauvegarder l'architecture de modèle sur le disque au format .json

Test : Pour cette partie il y a plusieurs aspects à prendre en compte, comme vérifier la compatibilité de taille d'entrée et sortie entre chaque couche, assurer l'intégrité du fichier .json en faisant des chargements et des sauvegardes avec le plus de combinaisons possibles.

5. Entraîner le modèle créé

- 5.1. Lancer l'entraînement via l'interface Keras avec les hyperparamètres choisis lors de la création du modèle.
- 5.2. Récupérer les métriques des fonctions de "loss" et "accuracy" données par l'interface Keras lors de l'entraînement.

- 5.3. Afficher les graphes et courbes correspondantes aux métriques données par les fonction "loss" et "accuracy", afin de détecter les effets de "l'overtutting".
- 5.4. Sauvegarder le modèle en cache au cours de l'entraînement au bout de "n" époques afin d'arrêter l'apprentissage si on voit des effets d'overtutting apparaître.

Test : Vérifier que l'entraînement s'est arrêté au moment souhaité, pour ce faire nous aurons besoin de faire des sauvegardes en cache du modèle au cours de l'entraînement.

6. Sauvegarder le modèle keras entraîné en tant que fichier .h5 ou .keras

Test : Vérifier que l'utilisateur dispose de suffisamment de place de stockage afin de sauvegarder le modèle créé et entraîné.

7. Définir architecture de modèle de base, destiné à faire de la classification des formes géométriques simples. Ce modèle a besoin d'être entraîné avec un jeu des données fixé par l'équipe de développement et sera aussi accessible par l'utilisateur, ce réseau servira en tant que démonstration d'utilisation de l'outil comme de base pour reproduire l'expérience depuis zéro.

2.1.2 Gestion du jeu de données

Pour réaliser l'expérience de base demandée par le client nous avons besoin d'intégrer un outil de génération de jeu de données simples, à savoir des formes géométriques élémentaires, carrées, cercles, triangles, segments.

1. Permettre à l'utilisateur de refaire l'expérience de base (objet d'étude) d'un réseau à convolution destiné à la classification de formes géométriques simples.

1.1. Créer un jeu des données d'entraînement

- 1.1.1. Développer un module pour générer aléatoirement un jeu de données qui aura besoin :
 - des types de formes que l'on souhaite produire,
 - d'une loi de distribution pour chacune des formes géométriques choisie (on pourra utiliser la même loi de distribution pour plusieurs formes),

- d'un nombre d'images pour le jeu de donnée,
- d'une loi de distribution pour déterminer la distribution des formes dans le jeu de données.

Ce module produira un jeu de données d'images contenant des formes géométriques simples, de taille et de positions aléatoire selon une loi de distribution définie pour chaque forme, et répartis dans le jeu de donnée selon une distribution uniforme.

Test : L'aspect aléatoire du module doit être testé en répétant plusieurs fois l'expérience et comparer les résultats entre chaque exécution.

- 1.1.2. Permettre à l'utilisateur de choisir la quantité d'images synthétiques à générer.

Test : Tester les cas extrêmes avec un nombre d'images très grande de l'ordre de million et gérer les éventuels problèmes de mémoire.

- 1.1.3. Générer des formes monochromes dans un premier temps, avec et sans "couleur" à l'intérieur.

- 1.1.4. Générer des étiquettes et les référencer à notre jeu de données.

- 1.1.5. Séparer aléatoirement les données d'entraînement et les données de test selon un pourcentage choisi par l'utilisateur.

Test : Vérifier la correspondance des étiquettes avec les images générées.

1.2. Charger un jeu des données existant

- Charger fichier contenant le jeu des données en format .npz (format compatible NumPy) sauvegardé sous la forme (X_train,Y_train,X_test,Y_test).
[7]

1.3. Permettre à l'utilisateur de faire de l'augmentation des données sur le jeu des données généré/chargé avec des rotations et zooms aléatoire sur les données.

1.4. Définir la taille des formes générées : par défaut les images seront générées pour être compatibles avec le réseau à entraîner.

2.1.3 Interface graphique

1. Ouvrir des fenêtres secondaires grâce à des boutons

- 1.1. Avoir un bouton *tk.Button* dédié.

- 1.2. Ouvrir une fenêtre secondaire
- 1.3. Mettre la fenêtre secondaire au premier plan

2. Naviguer dans la mémoire du pc

- 2.1. Avoir un bouton *tk.Button* dédié.
 - 2.2. Ouvrir un explorateur de fichier grâce à *tk.filedialog.asksaveasfilename* pour la sauvegarde.
 - 2.3. Ouvrir un explorateur de fichier grâce à *tk.filedialog.askopenfilename* pour le chargement.
- Test** : Vérifier qu'on ouvre l'explorateur de fichier dans le répertoire home/C de l'ordinateur.
- 2.4. Récupérer les fichiers .zip pour [charger le modèle](#).
 - 2.5. Récupérer les fichiers .npz pour le jeu des données.
 - 2.6. Récupérer les fichiers .jpg ou .png pour afficher les features maps .

3. Créer un jeu des données.

- 5.1. Choisir la forme géométrique avec une *tk.Listbox*.
- 5.2. Avoir un bouton *tk.Button* pour réinitialiser le jeu des données
- 5.3. Avoir un bouton *tk.Button* pour exporter le jeu des données
- 5.4. Avoir un bouton *tk.Button* pour créer un jeu des données depuis un fichier JSON
- 5.5. Choisir la configuration générale du jeu des données avec une *tk.Frame* dédiée
 - 5.5.1. Choisir la largeur et la hauteur de l'image avec deux *tk.Spinbox*
 - 5.5.2. Choisir la taille du jeu des données avec une *tk.Spinbox*
 - 5.5.3. Choisir la graine aléatoire avec une *tk.Spinbox*
- 5.6. Choisir le pourcentage de chaque forme avec une *tk.Spinbox*
- 5.7. Choisir si les formes seront remplies ou non avec une *tk.Checkbutton*
- 5.8. Choisir si les formes auront les couleurs inversées avec une *tk.Checkbutton*
- 5.9. Avoir des *tk.Frame* faites pour choisir la norme et ses paramètres
 - 5.9.1. Choisir sa loi de distribution avec une *tk.Listbox* entre 'uniform', 'triangular', 'normal', 'poisson'

- 5.9.2. Choisir les différents paramètres pour chaque loi avec des *tk.Spinbox*
 - 5.9.2.1. Choisir low et high pour la loi 'uniform'
 - 5.9.2.2. Choisir loc et scale pour la loi 'normal'
 - 5.9.2.3. Choisir left, right et mode pour la loi 'triangular'
 - 5.9.2.4. Choisir lam pour la loi 'side'
- 5.10. Avoir des paramètres pour ajouter un carré
 - 5.10.1. Choisir une loi pour x et y du premier point
 - 5.10.2. Choisir une loi pour la rotation
 - 5.10.3. Choisir une loi pour la taille
- 5.11. Avoir des paramètres pour ajouter un rectangle
 - 5.11.1. Choisir une loi pour x et y du premier point
 - 5.11.2. Choisir une loi pour la rotation
 - 5.11.3. Choisir une loi pour la largeur
 - 5.11.4. Choisir une loi pour la hauteur
- 5.12. Avoir des paramètres pour ajouter un cercle
 - 5.12.1. Choisir une loi pour le x et y du point central
 - 5.12.2. Choisir une loi pour le rayon
- 5.13. Avoir des paramètres pour ajouter un triangle
 - 5.13.1. Choisir une loi pour x et y du premier point
 - 5.13.2. Choisir une loi pour x et y du deuxième point
 - 5.13.3. Choisir une loi pour x et y du troisième point

4. Sauvegarder un jeu des données.

- 4.1. Pouvoir exporter le jeu des données ou générer les formes.
- 4.2. Ouvrir un explorateur de fichier pour savoir où enregistrer le jeu des données.
- 4.3. Récupérer le chemin vers le dossier .npz à enregistrer.

5. Charger un jeu de données

- 8.1. Ouvrir un explorateur de fichier pour savoir où récupérer les données .
- 8.2. Récupérer le chemin vers le dossier ou le fichier .npz à ouvrir.

6. Créer un modèle à entraîner.

- 6.1. Mettre un bouton *tk.Button* pour ajouter une couche au modèle.
- 6.2. Avoir une *tk.Listbox* pour choisir quelle couche ajouter entre "convolution", "dense", "pooling", "dropout", "flatten", "batchNormalisation".
- 6.3. Avoir un bouton *tk.Button* permettant de supprimer une couche.
- 6.4. Avoir un bouton *tk.Button* permettant de supprimer toutes les couches sauf la première
- 6.5. Avoir une première couche de convolution obligatoire.
- 6.6. Avoir le numéro de chaque couche d'afficher et qui se met à jour si suppression.
- 6.7. Avoir des paramètres de configuration
 - 6.7.1. Choisir la taille de l'image (largeur, hauteur) avec deux *tk.Spinbox*.
 - 6.7.2. Choisir la fonction d'optimisation avec une *tk.Listbox* entre "sgd" et "adam".
 - 6.7.3. Choisir la fonction de loss avec une *tk.Listbox* entre "categorical_crossentropy" et "mean_squared_error".
- 6.8. Mettre la valeur minimale en valeur par défaut
- 6.9. Créer un modèle type pour ajouter une couche de convolution.
 - 6.9.1. Choisir la taille du noyau (carré) pour chaque couche avec une *tk.Spinbox*.
Test : Vérifier que la taille minimale est 3.
 - 6.9.2. Choisir le nombre de filtres avec une *tk.Spinbox*.
Test : Vérifier que le nombre minimal de filtre est 1.
 - 6.9.3. Choisir le padding pour chaque couche avec une *tk.Listbox* entre "valid" et "same".
 - 6.9.4. Choisir la fonction d'activation pour chaque couche avec une *tk.Listbox* entre "sigmoid", "softmax", "softplus", "softsign", "tanh", "selu", "relu" et "exponential".
- 6.10. Créer un modèle type pour ajouter une couche de dense.

- 6.10.1. Choisir la dimension de l'espace de sortie pour chaque couche avec une *tk.Spinbox*.
Test : Vérifier que la dimension minimale est 1.
- 6.10.2. Choisir la fonction d'activation pour chaque couche *tk.Listbox* entre "sigmoid", "softmax", "softplus", "softsign", "tanh", "selu", "relu" et "exponential".
- 6.11. Créer un modèle type pour ajouter une couche de "MaxPooling".
- 6.11.1. Choisir la taille de la fenêtre pour chaque couche avec une *tk.Spinbox*.
Test : Vérifier que la dimension minimale est 2.
- 6.12. Créer un modèle type pour ajouter une couche de "Dropout".
- 6.12.1. Choisir la valeur pour chaque couche avec une *tk.Spinbox*.
Test : Vérifier que la valeur est comprise entre 0.1 et 1.0.

7. Sauvegarder un modèle.

- 7.1. Créer un modèle avec les couches mises dans l'ui dans l'ordre.
- 7.2. Ouvrir un explorateur de fichier pour savoir où enregistrer les données .
- 7.3. Récupérer le chemin vers le dossier ou le fichier .zip à enregistrer.

8. Charger un modèle.

- 8.1. Ouvrir un explorateur de fichier pour savoir où récupérer les données .
- 8.2. Récupérer le chemin vers le dossier ou le fichier .zip à ouvrir.

9. Afficher le sommaire d'un modèle

- 9.1. Avoir un bouton *tk.Button* dédié.
- 9.2. Pouvoir ouvrir une fenêtre secondaire avec *tk.Text* dedans
- 9.3. Gérer la taille de la fenêtres avec des *tk.Scrollbar*
- 9.4. Récupérer le sommaire d'un modèle précis en tant que texte
- 9.5. Pouvoir afficher le sommaire dans la nouvelle fenêtre

10. Entraîner un modèle

- 10.1. [Charger un modèle](#) avec un bouton *tk.Button* dédié
- 10.2. [Charger un dataset](#) avec un bouton *tk.Button* dédié

- 10.3. Pouvoir récupérer les paramètres pour séparer le dataset
 - 10.3.1. Choisir le pourcentage d'images de test avec une *tk.Spinbox*
 - 10.3.2. Avoir un *tk.Label* pour afficher le pourcentage d'images de train
 - 10.3.3. Avoir un *tk.Checkbutton* pour choisir si on veut des images de validation
 - 10.3.4. Choisir le pourcentage d'images de validation avec une *tk.Spinbox*
- Test : Vérifier que cela n'est modifier que si le *tk.Checkbutton de validation* est activé
- Test : Vérifier que la somme des pourcentages soit bien égal à 1
- 10.4. Pouvoir récupérer les paramètres pour l'entraînement
 - 10.4.1. Choisir le nombre d'epochs avec une *tk.Spinbox*
 - 10.4.2. Choisir le batch size avec une *tk.Spinbox*
 - 10.4.3. Choisir si on veut du early stopping avec un *tk.Checkbutton*
 - 10.4.4. Pouvoir choisir les paramètres du early stopping
 - 10.4.4.1. Choisir la patience avec une *tk.Scale*
 - 10.4.4.2. Choisir le moniteur avec une *tk.Listbox* entre 'val_loss' et 'accuracy'

11. Afficher une image au centre de la fenêtre

- 11.1. Avoir une fenêtre dédiée
- 11.2. Utilisation de *tk.Canvas* pour afficher l'image.

12. Afficher une grille d'images cliquables dans une fenêtre

- 12.1. Afficher les fenêtres dans une grille carrée
- 12.2. Avoir toutes les images de même taille
- 12.3. Pouvoir cliquer sur les canva avec un *bind*

13. Afficher les noms des couches.

- 13.1. Récupérer le nom des couches.
- 13.2. Afficher les noms dans l'interface graphique avec des *tk.Label*.

14. Afficher des graphiques.

- 14.1. Récupérer les données à partir de tableau.
- 14.2. Transformer le tableau en image avec `tf.keras.preprocessing.image.array_to_img` et `tk.TK.PhotoImage`
- 14.3. Afficher les images avec un `tk.Canvas`

15. Sauvegarder les images dans un fichier zip

- 15.1. Avoir un bouton `tk.Button` dédié
- 15.2. Récupérer le chemin avec un `filedialog.askopenfilename`

2.1.4 Non prioritaire

- Gérer des images en couleur.
- Zoomer sur les images.
- Exporter les résultats.
- Arrêter l'entraînement si on détecte de l'over/under fitting.
 - Récupérer la "loss" et "l'accuracy" pendant l'entraînement.
 - Les afficher en temps réel.
 - Sauvegarder le modèle à des temps régulier.
 - Arrêter l'entraînement à tout moment.

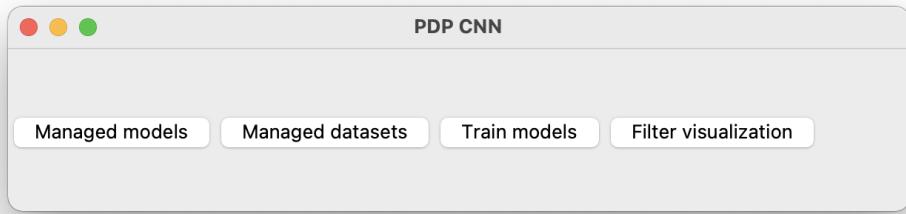
2.2 Besoins non fonctionnels

- 1.1. Avoir différents main pour lancer les parties du code séparément
- 1.2. Pouvoir manipuler des images en couleur
- 1.3. Pouvoir manipuler des images qui ne sont pas en 2D
- 1.4. Faire fonctionner l'application sous Jupyter.
- 1.5. Garantir la reproductibilité des expériences réalisées en s'appuyant sur la théorie de Vapnik-Chervonenkis pour la minimisation des erreurs pendant l'apprentissage.

3 Architecture et logiciel

3.1 Exemples de bon fonctionnement

L'utilisateur lance l'application depuis le code (main.py) : la fenêtre principale du projet s'ouvre.



L'utilisateur peut ensuite soit :

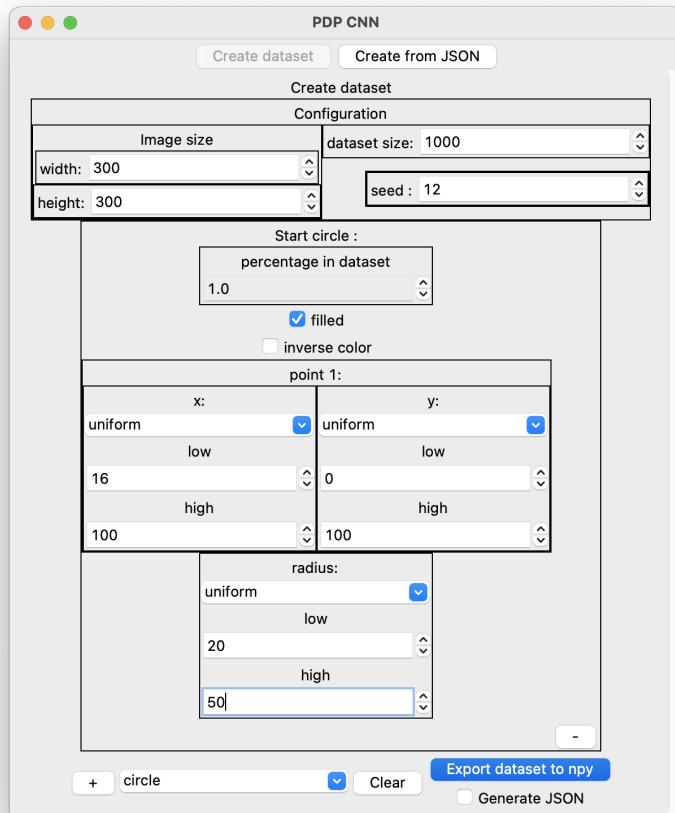
- Créer un Model (Managed models)
- Créer un jeu des données (Managed dataset)
- Entraîner un Models (Train models)
- Visualiser le contenu des différents filtres (Filter visualization)

Créer un jeu de données

L'utilisateur décide de créer un jeu de données, en cliquant sur le bouton (Managed dataset) il est automatiquement redirigé vers la fenêtre de gestion du jeu de données.



Devant cette fenêtre, l'utilisateur charge depuis le disque un fichier JSON ou choisit de créer le jeu de données.

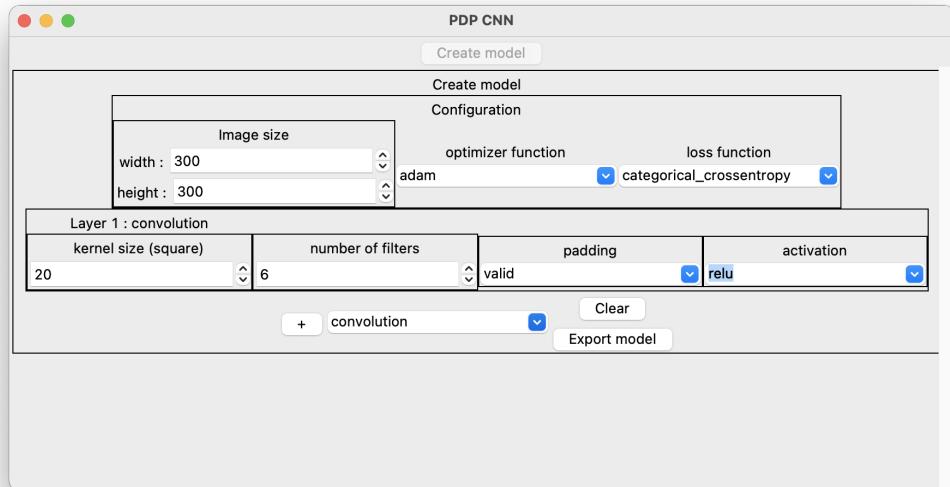


Dans cette fenêtre, l'utilisateur va pouvoir créer un jeu de données en renseignant les différents champs correctement.

- **Étape 1** : Remplir le champ configuration avec la taille des données, la graine et les dimensions de l'image
- **Étape 2** : Ajouter une figure géométrique qu'on souhaite générer avec le bouton (+) tout en renseignant les normes de distributions de la figure
- **Étape 3** : Exporter le jeu de données sous le format npy (choisir si on veut en plus le fichier JSON ou non)

Créer un modèle

L'utilisateur peut créer un modèle , en cliquant sur le bouton (Managed models) il est automatiquement redirigé vers la fenêtre de gestion du modèle.

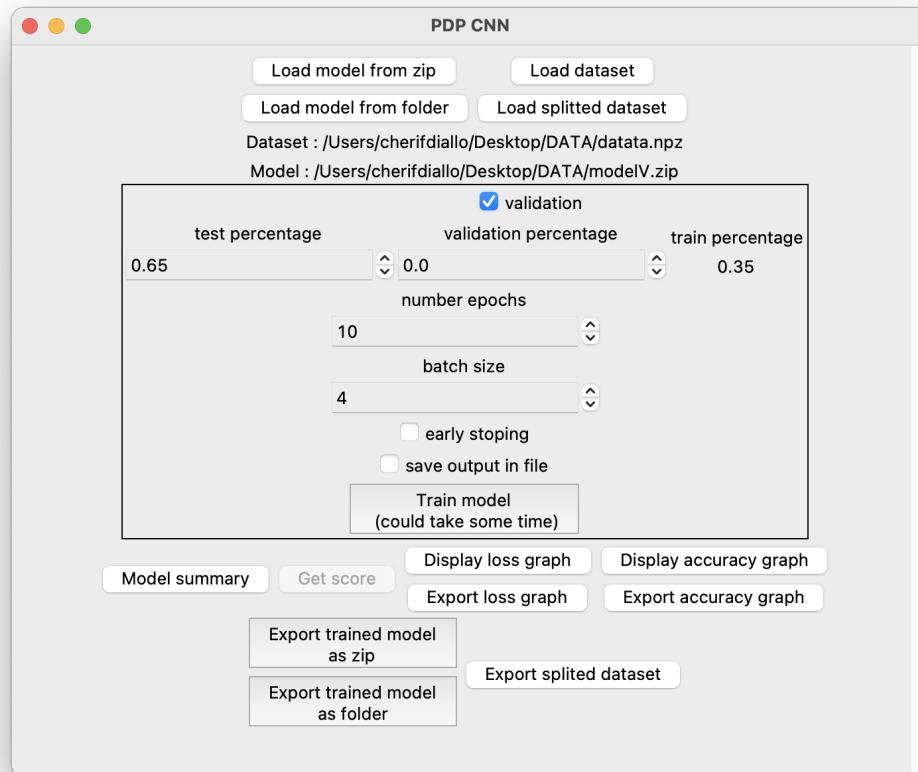


La gestion du modèle :

- **Étape 1** : Configurer le modèle : les dimensions de l'image, la fonction de perte et la fonction d'optimisation.
- **Étape 2** : Ajouter les layers avec un bouton de navigation par exemple (dense,dropout,flatten etc...). NB :la première couche de convolution est obligatoire.
- **Étape 3** : Exporter le modèle pour l'entraînement.

Entraîner un modèle

L'utilisateur peut entraîner son modèle , en cliquant sur le bouton (Train models) il est automatiquement redirigé vers la fenêtre d'entraînement du modèle

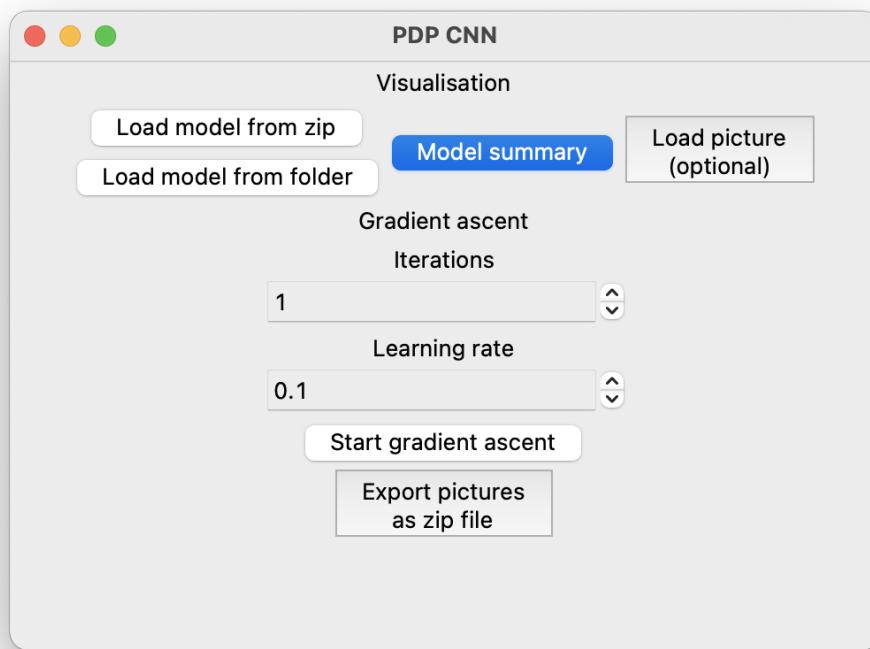


La gestion d'entraînement du modèle :

- **Étape 1** : L'utilisateur commence par charger le modèle et le jeux de données pour l'entraînement
- **Étape 2** : L'utilisateur défini le nombre d'epochs souhaitée pour son entraînement et le batch size
- **Étape 3** : Il lance ensuite l'entraîner du modèle
- **Étape 4** : Après la fin de l'entraînement l'utilisateur à plusieurs options, entre autre consulter le résumé du modèle (Model summary), afficher et/ou enregistrer les graphiques de perte ou d'efficacité (Display loss graph, Export loss graph, Display accuracy graph, Export accuracy graph)
- **Étape 5** : Si besoin, l'utilisateur peut aussi exporter le modèle entraîné sous le format zip (ou en tant que fichier) ou exporter le jeu de données qui a été séparé pendant la manipulation.

Visualisation des filtres

L'utilisateur peut visualiser les filtres d'un modèle entraîné en cliquant sur le bouton (Filter visualization).

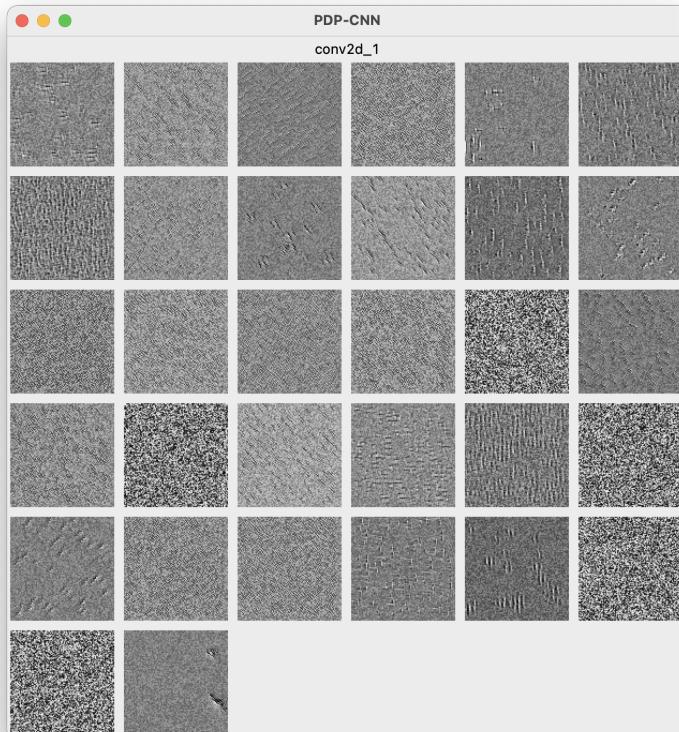


La gestion de la visualisation du modèle :

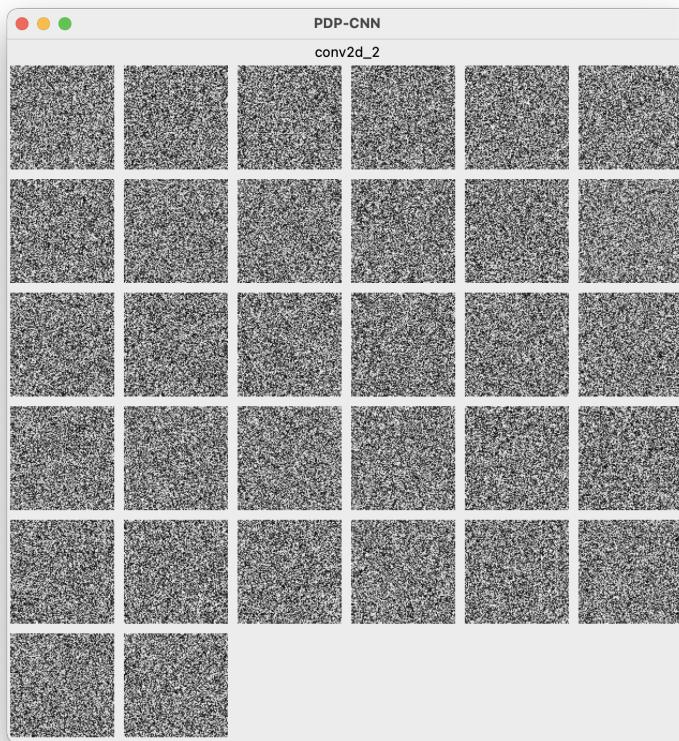
- **Étape 1** : L'utilisateur commence par charger un modèle déjà entraîné
- **Étape 2** : L'utilisateur définit les paramètres de la montée de gradient
- **Étape 3** : Si besoin il peut consulter le résumé du modèle (Model summary)

Model summary		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 100, 100, 32)	320
max_pooling2d (MaxPooling2D)	(None, 50, 50, 32)	0
conv2d_1 (Conv2D)	(None, 50, 50, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 25, 25, 32)	0
conv2d_2 (Conv2D)	(None, 25, 25, 32)	9248
max_pooling2d_2 (MaxPooling2D)	(None, 12, 12, 32)	0
flatten (Flatten)	(None, 4608)	0
dropout (Dropout)	(None, 4608)	0
dense (Dense)	(None, 64)	294976
dense_1 (Dense)	(None, 2)	130

- **Étape 4 :** Ensuite il lance la montée de gradient sur le modèle entraîné, pour visualiser les filtres

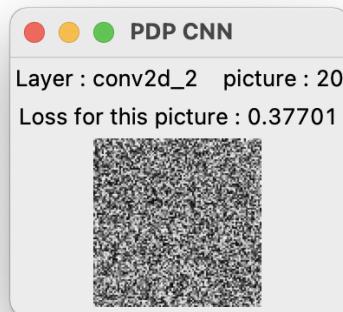


Exemple d'image obtenue dans la première couche conv2D



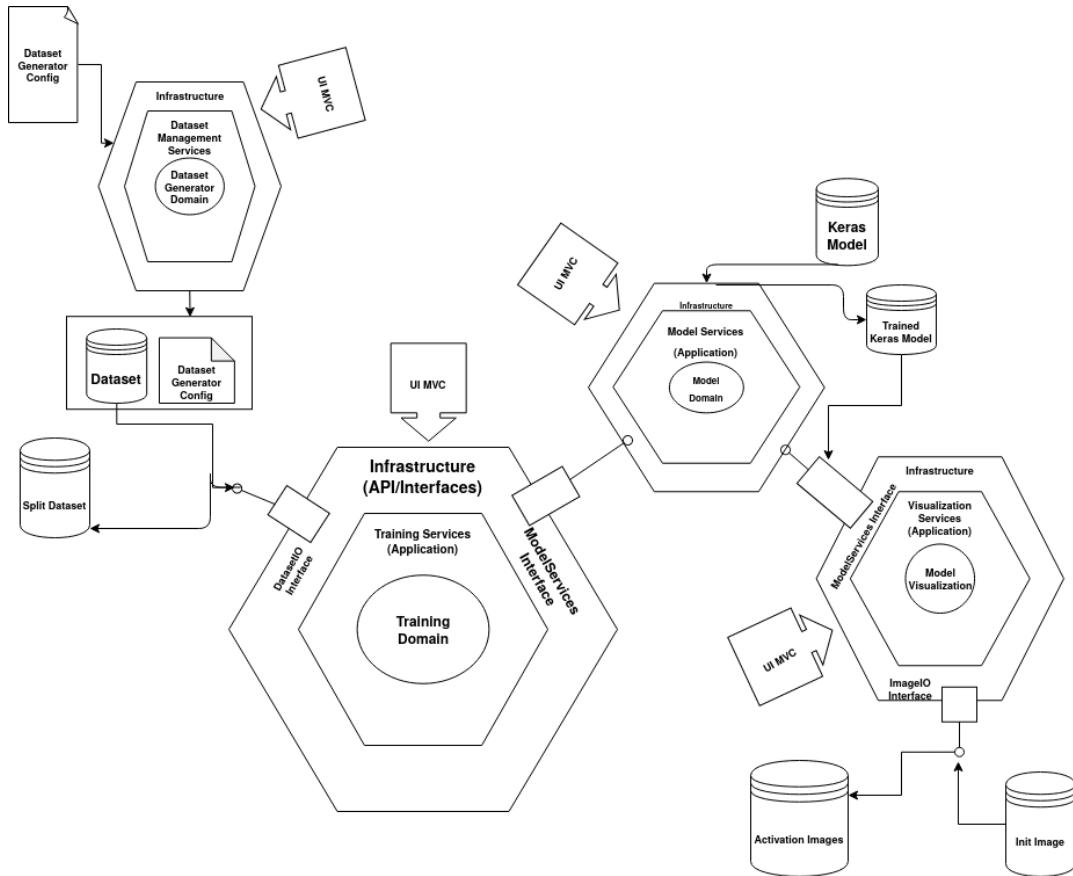
Exemple d'image obtenue dans la deuxième couche conv2D

En cliquant sur l'image, on obtient ses informations dans une fenêtre séparée, à savoir l'image de la feature map, avec son numéro, le numéro de sa couche et la perte qui lui est associée.



- **Étape 5** : Si besoin, l'utilisateur peut aussi exporter les images au format zip

3.2 Schémas



Notre architecture est basée sur le principe Domain Driven Design [8], permettant de séparer les différentes parties du code en encapsulant les concepts "métiers" dans une couche interne et permettant les traitements de ce dernier via des services. Chaque hexagone représente une partie distincte de notre application (création d'un jeu de donnée, création de modèle, entraînement de modèle, visualisation des couches de convolution) et se sépare en 4 packages :

- **application** : Représente les méthodes proposant des différents traitements pour un hexagone donnée. Le module "contrôleur" de l'UI va

interagir avec la couche application de chaque hexagone via des identifiants uniques.

- **infra** : Comprends tout le code technique, entrées sorties, manipulation des fichiers, interfaces avec plusieurs frameworks, le code de l'infrastructure est souvent en dehors de l'hexagone et permet de garder un contexte étanche entre chaque partie du code.
- **domain** : Code lié aux concepts métiers, dans cette partie on va retrouver une définition des concepts comme Modèle (réseau des neurones).
- **ui** : Interface graphique de l'hexagone.

Architecture interface utilisateur

L'interface utilisateur (ui) est quant-à elle écrite sur la base du MVC (Modèle, Vue, Contrôleur), séparant l'affichage graphique, les interactions utilisateur et les données à manipuler. Chaque package ui a une base de MVC et, au besoin, des classes supplémentaires faisant des tâches spécifiques (par exemple : afficher une fenêtre contenant du texte)

3.3 Bibliothèques externes

- tensorflow
- tkinter
- numpy
- pytest
- matplotlib
- skimage
- dataclass-type-validator
- dataclass-json

4 Analyse du fonctionnement et tests

4.1 Analyse du fonctionnement

Toute les modules de notre applications fonctionnent, les tests permettent d'assurer que des bugs majeurs n'apparaissent pas pendant son utilisation. Cependant, l'utilisation de dataset et de modèles keras (soit générés depuis notre application, soit importés depuis ailleurs), augmentent énormément les

différents cas d'erreur et tous n'ont pas pu être testés ou découvert pendant notre période d'implémentation.

4.2 Tests

4.2.1 Description des tests

Test de couverture



4.2.2 Discussion des résultats

Test de couverture Quand on regarde les pourcentages par rapport aux différents packages de chaque hexagone, on note que l'un est le package (peut import l'hexagone) qui est le moins couvert. Les hexagones de visualisation et train ont aussi moins de tests que dataset_management et model_management.

4.3 Problèmes, défauts, bugs

- A la fermeture de l'application, il peut arriver que des fenêtres ne se terminent pas correctement, et restent en tant que processus active impossible à fermer
- Il peut arriver que de nouvelles fenêtres apparaissent derrière celles déjà présentes
- Mis à part la distribution uniforme, les autres distributions risquent de provoquer un certain nombre d'erreurs si l'utilisateur renseigne des distributions pour lesquelles ont a de forte chances de tomber hors de l'intervalle 0 : taille de l'image ou
- rotation des rectangles et carrés non implémentées
- Les tests pourraient être plus robustes pour la partie dataset management.

5 Conclusion

5.1 Améliorations

- Avoir les pourcentages qui se mettent à jour en temps réel pour la génération de dataset
- Pouvoir charger un dataset json dans l'ui pour pouvoir le modifier
- Faire en sorte que les fenêtres de parcours de document dans le pc s'ouvrent tout le temps au même endroit
- Afficher l'entraînement dans une fenêtre séparée (et non dans un fichier)
- Générer des formes géométriques coupées
- Pouvoir écrire les valeurs directement dans l'ui pour le nombre d'epochs et la taille du batch pour l'entraînement

Références

- [1] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, “Understanding neural networks through deep visualization. (English),” *ICML Deep Learning Workshop*, 2015.
- [2] F. Chollet, *Deep Learning with Python*. Manning, 2017.
- [3] F. Chollet, “Visualizing what convnets learn.” https://keras.io/examples/vision/visualizing_what_convnets_learn/, 2020.
- [4] Keras, “Keras api.” <https://keras.io/api/>, 2020.
- [5] Tensorflow, “Tensorflow api.” https://www.tensorflow.org/api_docs, 2020.
- [6] Keras, “Keras kernel initialization api.” <https://keras.io/api/layers/initializers/>, 2020.
- [7] T. Oliphant, “Numpy doc api.” <https://numpy.org/doc/>, 2020.
- [8] E. Evans, “Domain driven design.” <https://www.domainlanguage.com/ddd/>, 2003.