

Pour faire confiance à un réseau de neurones déjà entraîné, on apprécie de pouvoir justifier ses décisions. On utilise donc des visualisations. La Q4 du TP précédent ouvrait ce genre de perspective en permettant d'observer les attributs isolés par les filtres de convolution de la première couche ; mais ils n'étaient lisibles que du fait de la compatibilité des dimensions avec les entrées (pré-pooling). Les méthodes étudiées ici se servent plutôt de l'influence du gradient propagé et permettent de ne plus se restreindre à cette première couche pour obtenir des visualisations interprétables.

Cartes de saillance

Les poids donnés à l'input le long de sa propagation sont connus car l'entraînement a déjà eu lieu. Sur un run, en rétropropageant les logits en sortie du réseau, on peut calculer le gradient du score de la classe réelle par rapport à l'image d'entrée. On obtient une liste de dérivées qui représentent le degré de sensibilité du score de sortie aux variations de chaque pixel, soit le « poids » de ces pixels dans le calcul de ce score. L'affichage de ces poids donne ce qu'on appelle une carte de saillance.

1 Montrer et interpréter les résultats obtenus.

On donne en beige les prédictions du modèle. Le code initial n'affichait que la classe réelle, mais on constate en fait certaines erreurs plus ou moins graves.

Intéressons-nous d'abord à l'aspect des cartes de saillance. Sur la colormap `pyplot.hot`, le rouge indique une forte activation. Or sur les images ci-dessous, les formes des objets à reconnaître sont souvent bien soulignées par une surface rouge. Par exemple, pour décider que l'image 1 représente un chien (la race prédite est plausible), le modèle semble s'être concentré sur la région où se trouve le chien : ce sont ces pixels-là qui ont le plus participé et surtout ceux de la tête. Ils ressortent bien sur la carte de saillance.



Figure 1 – Cartes de saillance obtenues pour diverses images. Classe réelle indiquée en noir, classe prédite par SqueezeNet en beige.

Mais ces cartes sont déconcertantes pour un humain.

→ Dans des cas où la prédiction est juste, on peut n'y retrouver aucune forme connue (grande dispersion pour le chardon et l'oiseau) ou constater une focalisation inattendue. La chaussette de Noël apparaît carrément en creux : c'est l'arrière-plan qui montre la plus forte réaction et la forme se détache sous forme d'ombre. Quant aux bottes de foin, seules deux comptent. Il semble manquer des points d'activation.

→ En regardant uniquement les cartes, les cas d'erreur ne sont pas séparables du reste. Pour le bateau, on a aussi cet effet de creux qui ne posait aucun problème de classification plus tôt. L'erreur est pire dans le cas des cygnes qui sont mépris pour un phare (en raison de l'eau autour), mais on pourrait lire la carte de saillance de la même façon que celle du chardon en n'y voyant qu'une certaine dispersion.

Que la prédiction soit juste ou non, on comprend en fait que la valeur du gradient en tel ou tel point et donc la couleur exacte de la carte n'est pas ce qui compte : elle doit être lue à une échelle globale. Par ailleurs, la valeur numérique du gradient ne concorde pas avec les points focaux sémantiques de l'image.

2

Discutez les limites de cette technique pour visualiser l'importance des différents pixels.

L'activation affichée sur les cartes ne correspond pas à ce qu'on attendait intuitivement. Sur les bottes de foin, la carte omet des informations importantes ; pour le bateau, le chardon, l'oiseau et autres, la réaction montrée du modèle n'est pas proportionnelle à l'importance sémantique des éléments.

Ce peut être la réalité du réseau, mais des choix théoriques sont aussi en cause comme son approximation par une fonction linéaire. Le fait d'outrepasser ainsi la non-linéarité du réseau peut partiellement expliquer la dissonance entre ce qui nous semble logiquement important et la distribution du rouge sur les cartes.

Cela étant, les cartes de saillance n'ont pas à concorder avec l'intuition. Elles ne représentent pas les classes, elles informent les usagers sur les mécanismes de décision du modèle, et ce pour une image précise. On peut donc oublier ce qui précède, mais même ce nouveau point de vue leur conserve les défauts suivants :

- En concentrant la coloration sur des pixels, des positions, elles ne peuvent que localiser les caractères d'intérêt. Etudier ces cartes, c'est donc inconsciemment tout ramener à des formes en oubliant que d'autres constantes (luminosité, colorimétrie) jouent. Cette visualisation de l'importance des pixels est donc biaisée.
- Du fait de leur rattachement à un seul run, on a également du mal à utiliser les cartes de saillance pour prédire ce que le réseau ferait face à une autre image de la même classe, alors que l'objectif initial était pourtant d'avoir des repères pour développer la confiance générale de l'utilisateur dans le modèle.

Voir l'article suivant (Alqaraawi et al.), qui propose de toujours compléter les cartes de saillance avec d'autres méthodes, et aussi de les échantillonner par classe pour obtenir une représentation plus générale.

3

Cette technique pourrait-elle servir à autre chose qu'à interpréter le réseau ?

L'interprétation du réseau est le but principal avec tout ce qu'elle implique d'améliorations potentielles (développement de l'attention, correction d'erreurs, optimisation des inputs). Mais on voit l'intérêt des cartes de saillance dans leur capacité à localiser des objets. Ce sont donc de bonnes alliées pour une tâche de segmentation, et rien n'empêche de superposer des cartes pour une implémentation multiconfiance.

4

Tester avec un autre réseau, par exemple VGG16.

VGG16 est plus lent. C'est le prix d'une meilleure précision : sur les 25 images, il se trompe moins.



Figure 2 – Cartes de saillance obtenues pour diverses images. Classe réelle indiquée en noir, classe prédictive par VGG16 en beige.

Cette fois, les points d'intérêt concordent avec ceux qu'on attend. Le chardon et l'oiseau sont visibles dans leurs cartes, l'œil de l'oiseau ressort. Toutes les bottes de foin sont couvertes par la carte de saillance et les formes des deux cygnes sont mieux isolées, ce qui a permis d'avoir une prédiction correcte.

Les phénomènes d'inversion (chaussette de Noël) existent là aussi, mais l'apparence générale des cartes est plus naturelle. L'architecture de VGG16 (pas de fire units, plus de poolings...) peut expliquer ce naturel. Avec VGG16, on traite toujours plutôt des régions ; il y a donc moins de pixelisation qu'avec SqueezeNet, dans lequel chaque pixel évolue effectivement de façon plus indépendante.

Exemples adversaires

On continue l'étude des décisions des réseaux en les mettant cette fois en difficulté. En utilisant une montée de gradient (des logits vers l'input), il est possible, sans modifier les poids appris, de générer des images subtilement arrangées de sorte à les tromper : des exemples adversaires.

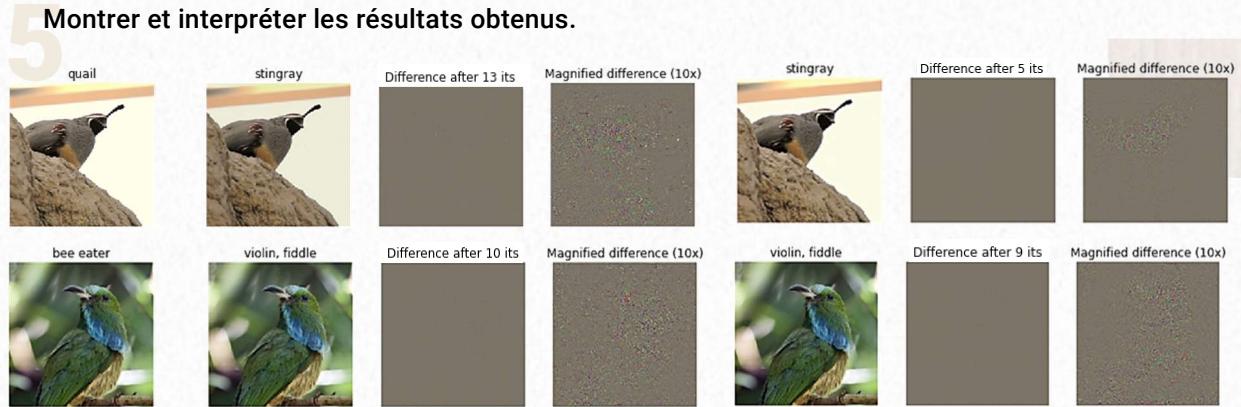


Figure 3 – Mise en difficulté de SqueezeNet (gauche) et de VGG16 (droite) par montée de gradient : confusion entre la classe originale sur la première image et une classe cible arbitraire. Les titres représentent la prédiction du modèle, or elle change, ce qui est inexplicable pour nous.

Sur ces images, SqueezeNet est floué en très peu d'itérations. Le faux est indiscernable de l'original, seule l'amplification de l'erreur permet de constater qu'il y a effectivement des changements. D'ailleurs, l'aspect des modifications ne rappelle pas l'étiquette cible : il s'agit de variations quasi-aléatoires pour nous. En essayant avec VGG16 au lieu de SqueezeNet, on remarque que la tromperie prend plus de temps, mais moins d'itérations et est plus concentrée sur la région précise où se tiennent les sujets principaux. C'est dû au fonctionnement différent des deux réseaux (la notion de pooling doit encore être en cause).

Nous avons essayé les classes les plus absurdes, et la tromperie marche toujours en adaptant le pas de la montée de gradient ; mais s'il est fixé à une petite valeur, il peut falloir plus de 100 itérations (cf. Notebook).

6 Quelles conséquences cela peut-il avoir pour l'utilisation de réseaux de convolution en pratique ?

Tout modèle convolutionnel est sensible aux attaques adversariales, et elles sont craintes quand il a des responsabilités. L'exemple des voitures autonomes filtre dans le débat public : des modifications discrètes des panneaux de signalisation, volontaires ou non, peuvent flouer leurs détecteurs et provoquer des accidents. Le problème est que ces attaques sont invisibles pour les humains et qu'on voit mal comment les surveiller.

7 Discutez des limites de cette façon naïve de créer des images adversaires. Proposez des alternatives.

L'attaque dépend ici des paramètres appris du modèle et d'images correctement classées. Une façon de l'empêcher serait de ne pas y donner accès, mais des générateurs d'images peuvent remplacer un training set caché, et le modèle est toujours exposé à moins d'être très spécial - on peut trouver une attaque valable à partir d'une architecture publique. Nous n'avons pas pu observer facilement un tel transfert entre VGG16 et SqueezeNet (fonction `transferFooling`), mais c'est possible pour des réseaux plus proches.

Cette intuition est bien à la source de ce qu'on appelle les « black-box attacks » et l'article **suivant** (Papernot et al.) décrit un protocole : on peut appliquer la montée de gradient à un modèle connu, entraîné pour avoir des classifications similaires à celles de la cible, et attaquer avec l'image résultante. Destiné aux cibles potentielles, **cet autre article** (Demontis et al.) donne un moyen d'estimer la faisabilité de ce genre de transfert pour s'en prémunir. La question est donc toujours en recherche côté attaque et côté défense.

On explique le succès de ces attaques par l'apprenabilité même des réseaux. Nous l'optimisons souvent via des activations ReLU, dont les gradients ne peuvent pas saturer pour les réels positifs. Cela donne une bonne réactivité à l'entraînement... mais aussi une faille de sécurité : il est possible de pousser facilement le gradient vers des valeurs arbitraires. Un modèle manipulé pourra donc avoir une très grande confiance dans sa prédiction alors qu'il classifie des exemples adversariaux.

Visualisation de classes

Nous avons constaté que les prédictions d'un modèle étaient faciles à dévier vers une classe cible. Or, visualiser les attributs modélisés d'une classe (là aussi avec une montée de gradient) pourrait permettre de les renforcer sur les inputs. Nous implémentons ici quelque chose de similaire au système utilisé par Google DeepDream, qui a surtout été plébiscité pour ses applications créatives.

8 Montrer et interpréter les résultats.

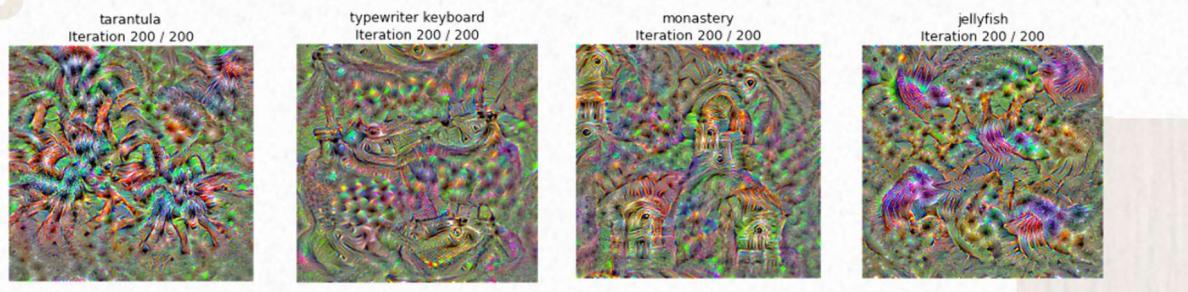


Figure 4 – Rêves de tarentules, de machines à écrire, de monastères et de méduses obtenus via SqueezeNet.

En partant d'un bruit, il est facile de construire des images de la classe voulue si l'on a accès au modèle. Notons néanmoins que les couleurs n'ont rien de naturel. Pour un humain, ces images sont distordues mais lisibles *a posteriori* : les pattes des tarentules, la forme de la machine à écrire, les coupoles du monastère, les méduses. Les objets apparaissent plusieurs fois et on peut en retrouver les attributs caractéristiques à toutes les échelles (on voit facilement six méduses de formes et de tailles variées). Cette multiplicité artificielle semble renforcer les chances de détecter la bonne classe cible en sortie.

Ces résultats ont été obtenus avec les paramètres par défaut et ne sont pas déterministes (à partir du même bruit initial, des maxima légèrement différents peuvent être mis en valeur d'une exécution à l'autre).

9 Essayer de varier le nombre d'itérations, le learning rate, le poids de la régularisation.

La variation du nombre d'itérations ne change pas le nombre de « foyers » présents pour les formes reconnaissables. Ci-dessous, ils sont déjà bien mis en place à partir de 50-100 itérations. Continuer à itérer permet d'asseoir ces motifs et de les faire ressortir davantage en les contrastant.

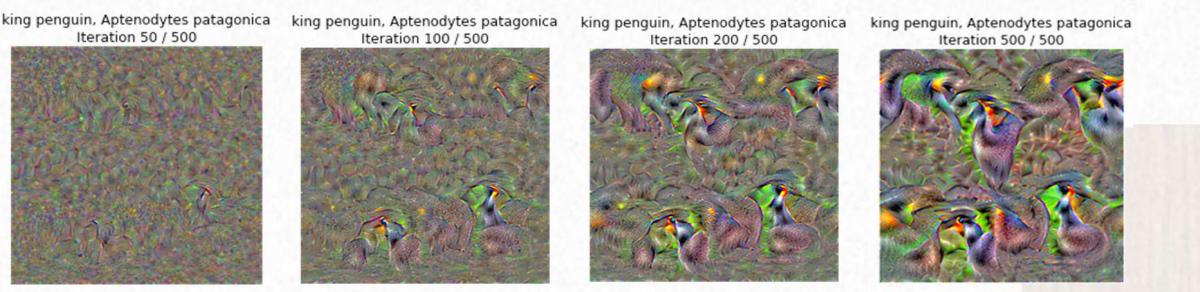


Figure 5 - Rêves de pingouins à divers stades de complexification.

La variation du learning rate modifie l'échelle et la position des formes. Sur le sablier, le même nombre d'itérations donne un résultat d'autant plus coloré et vif que le learning rate est grand (construction rapide).

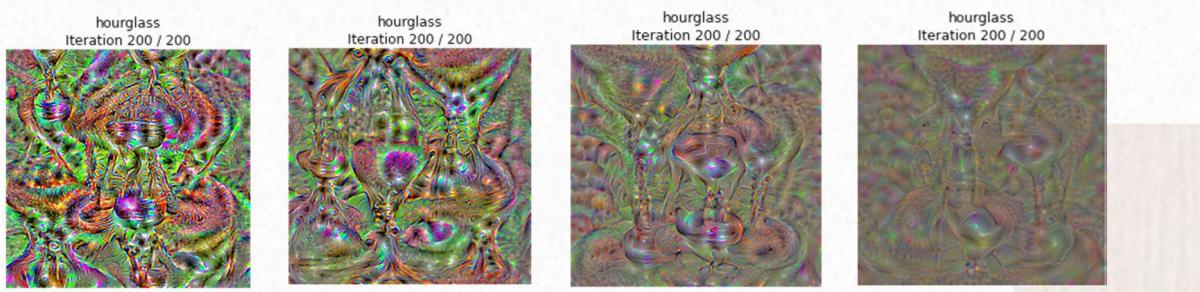


Figure 6 – Rêves de sabliers pour des pas d'apprentissage différents (10, 5, 2, 0.5 de gauche à droite).

Prendre un learning rate plus petit donne un résultat grisâtre car la construction par contrastation n'est pas finie. L'apprentissage aurait gagné à être allongé. Cela a aussi une influence sur l'échelle et la qualité : les objets dessinés pour un learning rate de 10 (grand) sont plus petits et leur forme est moins cohérente.

On fera varier la constante de régularisation en question suivante.

10

Essayez d'utiliser une image d'ImageNet comme image source au lieu d'une image aléatoire. Vous pouvez utiliser pour classe cible la classe réelle. Commentez l'intérêt.

On essaie la génération à partir d'une image, mais avec des constantes de régularisation croissantes (pour terminer la question précédente) : on constate que les foyers de création restent les mêmes, mais que l'image originale est plus ou moins visible à travers le rêve. Il existe une valeur d'équilibre (ici 0.5) où le niveau de détail du rêve est maximal ; au-delà, il disparaît lui aussi. L'excès de régularisation a donc l'air de ternir l'image sans modifier le nombre et la position des formes rêvées.

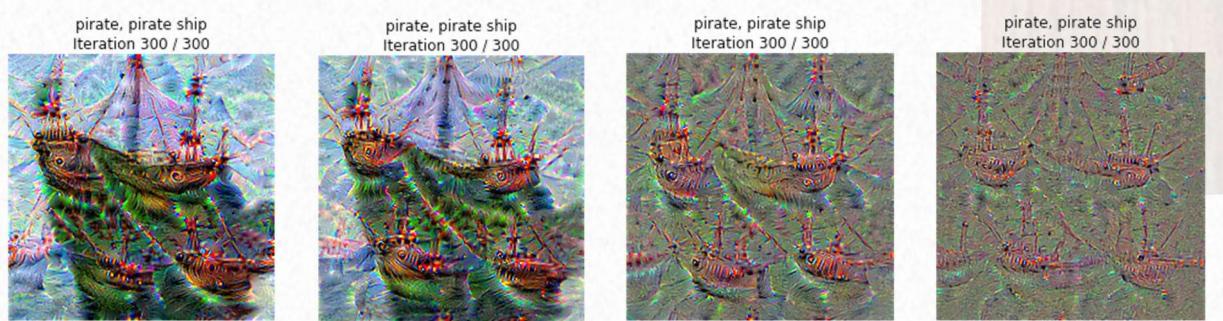


Figure 7 – Rêves de bateaux pirates pour des valeurs de régularisation différentes (0.00001, 0.1, 0.5, 1 de gauche à droite).

La régularisation L2 sert en fait à éviter que l'image traitée s'éloigne trop de l'original, mais c'est peine perdue pour la vision humaine. Le modèle, lui, y gagne. Améliorer ainsi « discrètement » la visibilité de la classe sur une image d'étiquette connue peut être une façon de le renforcer : il saura plus facilement reconnaître les images car il y verra plus d'attributs représentatifs.

Pour l'utilisateur, visualiser ces attributs artificiels permet aussi de repérer des biais dans la représentation de la classe. Nous avons ici l'impression que les bateaux pirates apparaissent souvent sous un certain angle, ce qui peut expliquer la difficulté que le modèle a eue à classifier celui-ci (en contre-pied).

D'humeur créative, nous avons aussi essayé des générations de classes complètement indépendantes sur notre ensemble de travail. La transformation est impressionnante : toute trace de la classe originale est perdue. Ici, la modification est trop évidente pour parler de tromperie comme en Q5.

Les couleurs de l'image de départ peuvent parfois être conservées, mais jamais celles de la cible :

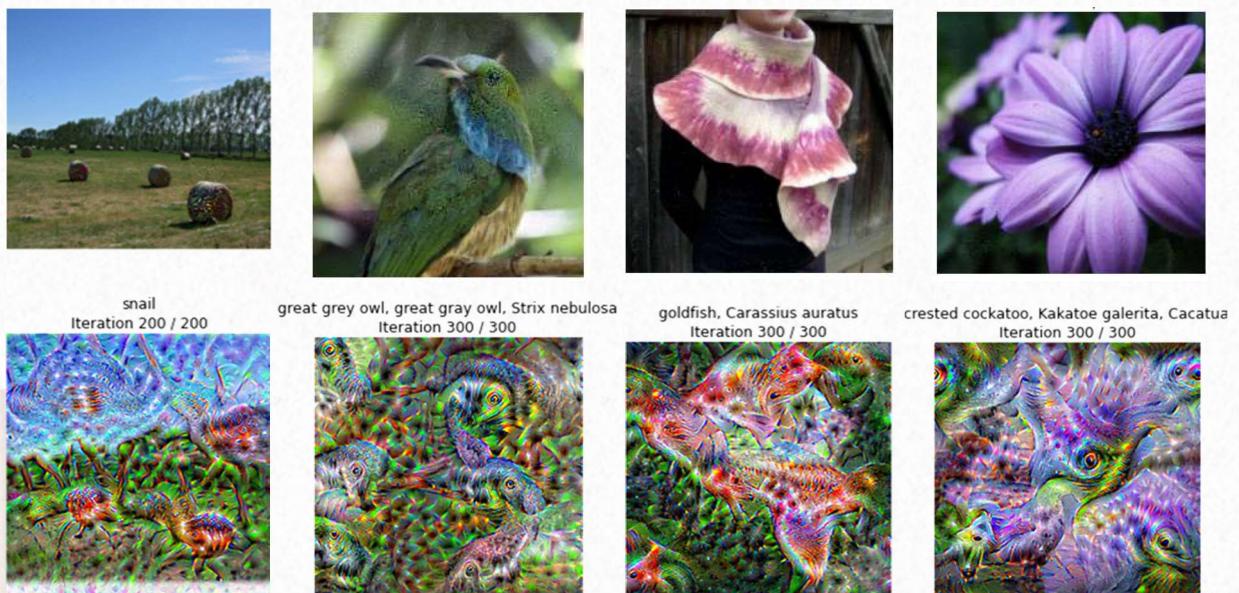


Figure 8 – Rêves d'escargots, chouettes, poissons rouges et kakatoès à partir d'images non liées.

11

Essayer avec un autre réseau, par exemple VGG16.

VGG16 est encore une fois plus lent, et les sorties sont différentes de celles de SqueezeNet. En appliquant le même traitement qu'à la dernière image de la Fig. 8, on a constaté que le même paramétrage rendait la sortie illisible : on décide d'abaisser le learning rate, d'étendre l'apprentissage et de régulariser davantage (Fig. 9). Quels que soient les paramètres néanmoins, **VGG16 ne fait pas apparaître des formes entières dans les rêves**. Il ne s'agit que de petits détails, des attributs isolés : la crête, les yeux, ou un bec d'oiseau.

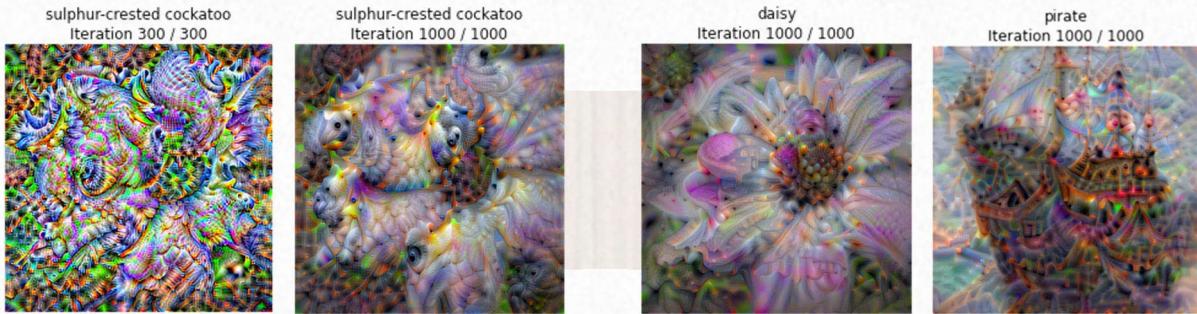


Figure 9 – Rêves de kakatoès avec VGG16 : impact du paramétrage

Figure 10 – Rêves de renforcement avec VGG16 sur deux images

En Fig. 10, on a essayé VGG16 pour le renforcement d'attributs (Q10), et ses modifications sont encore une fois plus subtiles que celles de SqueezeNet : il structure la texture de l'image. Sur le bateau, rien de lisible, mais il a ajouté un cœur discret à l'arrière-plan pour la fleur. Son architecture fait qu'il modélise les classes sous forme de toutes petites régions associées, et il les recrée là où il s'attendait à les voir.

Les autres rêves présentés ci-dessous servent à montrer une conservation des couleurs de la classe cible, qu'on n'avait pas du tout dans SqueezeNet (cf. Q10) et qui se voyait déjà un peu dans la crête de l'oiseau en Fig. 9. Ci-dessous, les taches du léopard des neiges sont noires et blanches, le violon a la bonne teinte. On peut rapporter cela à l'absence de fire units dans VGG16, donc à l'absence de convolutions 1x1 + reconstructions apprises qui dénaturent les couleurs en perturbant le traitement des canaux d'origine.

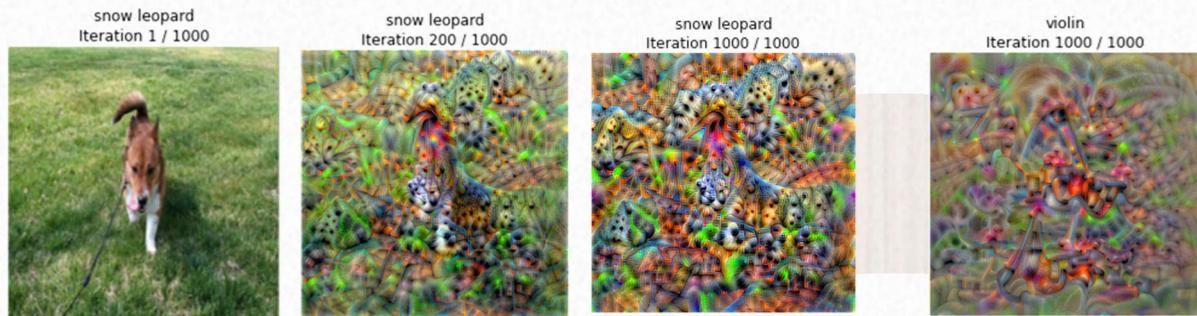


Figure 11 – Autres rêves

Le naturel de ce renforcement d'attributs (aussi appelé *activation maximisation*) peut être optimisé. Des ressources trouvées ici et là suggèrent d'ajouter un flou gaussien, et un prior pour maximiser la vraisemblance des générations (en supposant les caractéristiques des images « naturelles » connues).

Conclusion

Les techniques indépendantes de ce TME ont permis de jeter un œil sur la mécanique de VGG16 et de SqueezeNet. Leur nature de réseaux Deep fait qu'on peut y exploiter les mêmes phénomènes liés à la propagation du gradient, mais on a constaté que chaque architecture y réagissait différemment. En développement, des modes de visualisation de ces réactions peuvent donc aider à choisir un réseau adapté pour une tâche précise, qu'on veuille maximiser sa fiabilité (et le bombarder d'attaques adversariales), comprendre ses prédictions isolées (avec des cartes de saillance), ou interpréter la représentation faite des classes (en le laissant souligner les indices trouvés dans les entrées). Dans le dernier cas, on peut aussi simplement apprécier la créativité apparente de ces images – même si l'on sait qu'elle ne dépend en rien de décisions spontanées.