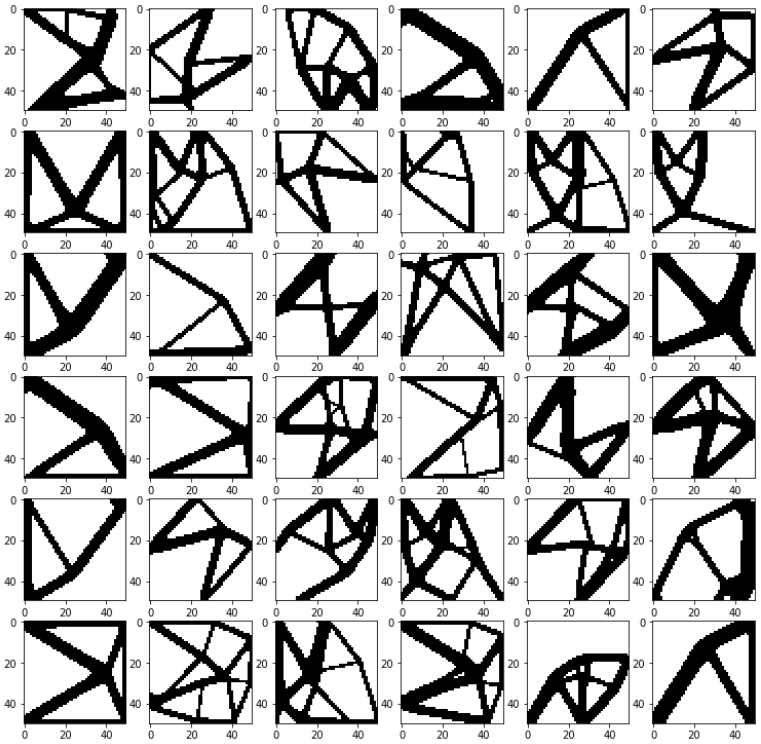
TP4 : DECOUVERTE DES RESEAUX ANTAGONISTES GENERATIFS (GAN)

Les Réseaux Antagonistes Génératifs (en anglais GAN Generative Adversarial Networks) ont été introduits par Iain Goodfellow (ancien ingénieur de recherche chez Google Brain, aujourd'hui directeur de l'IA chez Apple) et ont révolutionné les méthodes génératives non supervisées. Comme les Auto Encodeurs, les GAN permettent de générer des données « réalistes » à partir d'un ensemble réduit de données. Par « réalistes », on entend des données qui suivent la distribution statistique des entrées. Typiquement des GAN appliquées sur des images d'un genre donné (des visages humains par exemple) permettent de générer des visages qui « ressemblent » aux visages de la base de données d'origine. Les GAN ont rapidement permis de surpasser les autres modèles génératifs et sont désormais utilisés via des applis ou des logiciels pour générer ou modifier des images qu'il devient de plus en plus difficile de distinguer d'images réelles (pour la production de fake news notamment). Bien que leur résultat soit souvent spectaculaire l'entraînement de GAN reste délicat car il repose sur l'entraînement de deux architectures neuronales concurrentes. Le générateur qui va proposer des données au début aléatoire et un discriminateur qui va chercher à distinguer les images générées des vraies images. L'entraînement se fait en général « en crabe » avec une itération de SGD qui va mettre à jour les poids du générateur et une autre qui va améliorer le discriminateur. Si les architectures et fonctions de perte sont bien réglées, cette procédure peut converger vers un discriminateur qui n'est plus capable de distinguer les images générées des vraies images. Dans ce TP, nous allons découvrir la mise en données de GAN où les architectures sont profondes et illustrer les difficultés de convergence des GAN sur un exemple original de génération de design mécanique.

OBJECTIF : CREATION DE GAN POUR LA GENERATION DE DESIGN MECANIQUES

On cherche aujourd'hui à illustrer la création de GAN sur un exemple original : la création de design mécanique. Ces designs ont été créés par des outils dédiés (optimisation topologique en l'occurrence) et répondent à un cahier des charges précis (fixations et charge à supporter). On va travailler aujourd'hui avec une base de plusieurs milliers de tels designs, qu'on représente ci-dessous.



QUESTIONS

1. **Sur GANlab, tester différents réglages des learning rate du discriminateur et du générateur (typiquement des grandes valeurs pour les 2, des petites valeurs pour les 2). Qu'observez-vous ? Comment interprétez-vous les courbes de métriques à droite de l'écran ?**

Pour vous aider vous pouvez regarder la courte vidéo de présentation de GAN lab.

2. **Suivez le TP pour importer les design mécanique. Avant tout entraînement de GAN, nous allons estimer la dispersion des données d'entrées et s'assurer que les différent design restent cohérents. De combien de design dispose-t-on ? Superposer sur un même graphique différents design. Chaque design a un certain niveau de matière (nombre de pixels de valeurs 1). Tracer l'histogramme de ce niveau de matière sur tous les design. Quelle est votre conclusion sur la cohérence des design ?**

Fonctions utiles : np.sum, pyplot.hist

3. **Pour entraîner notre première architecture de GAN, nous allons nous appuyer sur le tutorial de Tensorflow utilisé sur MNIST et l'adapter à nos design mécaniques. : https://www.tensorflow.org/tutorials/generative/dcgan Décrire les architectures de générateur et de discriminateur. Expliquer le calcul de la fonction perte. Modifier le code original pour prendre en compte les images de design de taille 50 x 50 en appelant tf.images.resize sur chaque batch. Faire tourner un premier apprentissage. Visualiser les courbes d'apprentissage obtenues**

Les premiers résultats obtenus par cette architecture de DCGAN sont prometteurs. Néanmoins, la convergence est bien souvent subtile à obtenir. On vous propose dans la suite d'utiliser une architecture plus complexe pour le générateur déjà écrite en tensorflow et de voir si cette dernière permet d'améliorer les résultats d'un point de vue visuel.

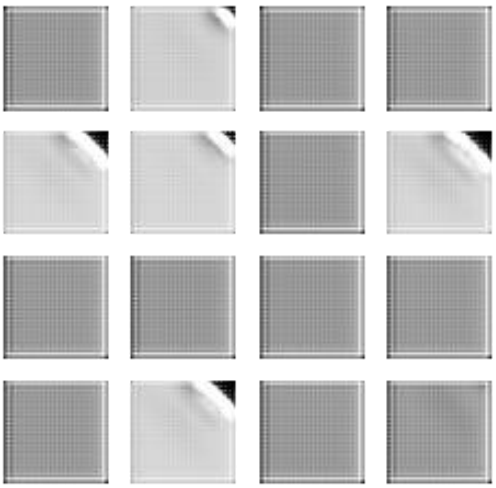
4. **Initialisation des architectures et la dimension latente.Que font les fonctions `initialize_weights_generator` et `initialize_weights_discriminator` et quelle est le rôle de la variable `nf`?Quelles sont les architectures utilisées ? Initialiser des architectures avec `nf=32`, Visualiser un exemple de génération**

Pour vous aider, on commence par initialiser les poids et biais de chacune des architectures via la fonction dédiée, une fois qu'on a les poids et biais, on appelle la fonction generator qui prend en entrée les dimensions latentes et les poids et biais du réseau générateur. Enfin, les dimensions latentes doivent être encodées sous la forme d'un tensor tensorflow : soit

```
noise = tf.random.normal([1,100])
```

5. **Premier entraînement du GAN. On va commencer par définir les optimiseurs sous la forme d'objet tensorflow et on va ensuite utiliser une fonction donnée train_step. Que fait la fonction train_step ? Réaliser un premier entraînement avec peu d'epochs ppour vous assurer que la démarche fonction ? Que donnent les premiers résultats ?**

On va commencer par définir les optimiseurs (classe tf.optimizers) et ensuite on va utiliser la fonction train_step et la fonction train, enfin la fonction generate_and_save permet de générer et de sauvegarder des images à l'issue de l'entraînement et donc d'estimer la qualité de reconstruction. Si tout se passe normalement, vous devriez avoir quelque chose comme ci-dessous



Fonctions utiles :

6. **Améliorer les résultats précédents en jouant sur les architectures, les paramètres, les fonctions de perte, les paramètres des optimiseurs et des taux d’apprentissage.**