

# Christal

Compte rendu n°6

## Interface graphique

Nous avons fait une refonte de l'interface graphique pour qu'elle prenne son aspect définitif.

Nous avons décidé qu'elle aurait une apparence assez douce et subdivisée en plusieurs petites fenêtres, chacune ayant un rôle en particulier.

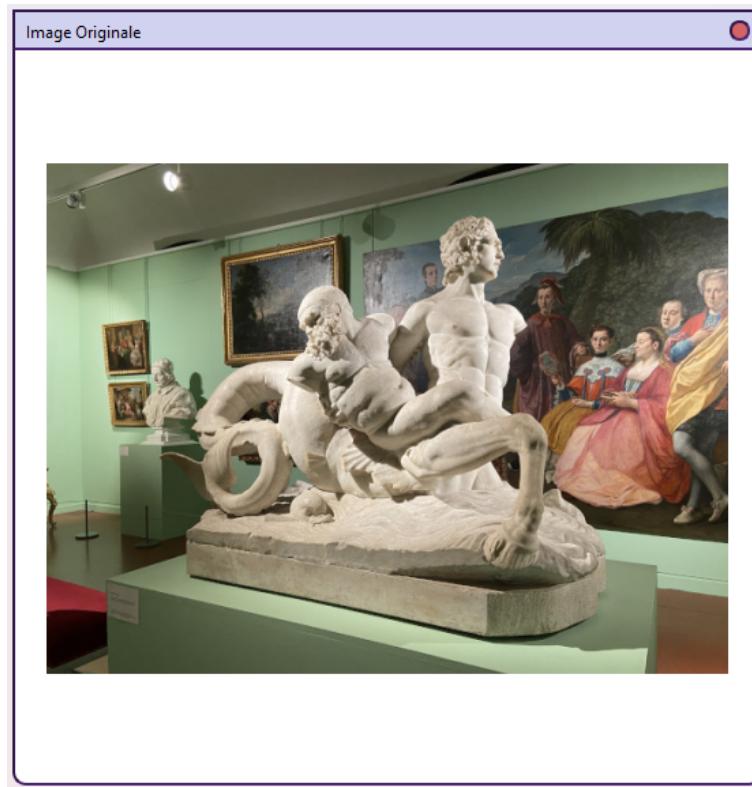
Ci-dessous, la fenêtre correspondant à l'ouverture d'une image. Celle-ci apparaît constamment sur notre application. Lorsque l'utilisateur appuie sur le bouton *ouvrir image*, la fenêtre de dialogue qui s'ouvre est automatiquement redirigée vers le dossier "Images" de l'ordinateur. Nous avons également choisi de filtrer les fichiers pour ne montrer que les images au format jpeg.

```
options = QFileDialog.Options()
dossierimage_path = QStandardPaths.writableLocation(QStandardPaths.PicturesLocation)
fileName, _ = QFileDialog.getOpenFileName(self, "Sélectionner une image",
dossierimage_path, "Images (*.jpg *.jpeg)", options=options)
```

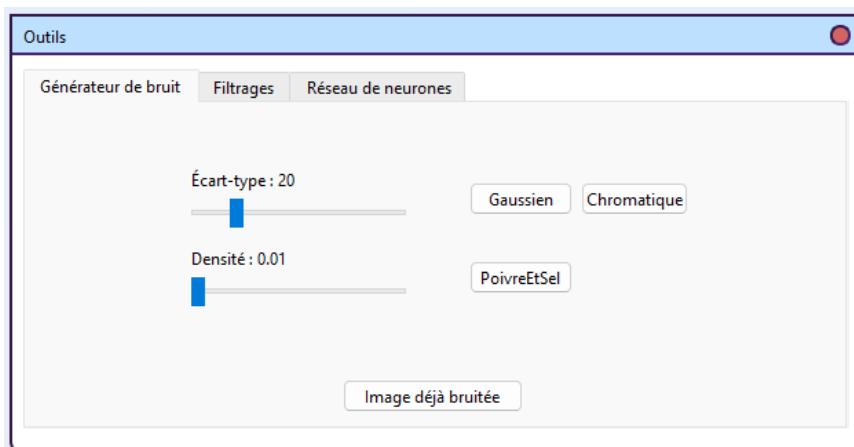


Une fois l'image sélectionnée, le chemin du fichier est alors modifié sur la fenêtre jaune et deux fenêtres supplémentaires apparaissent: une de couleur

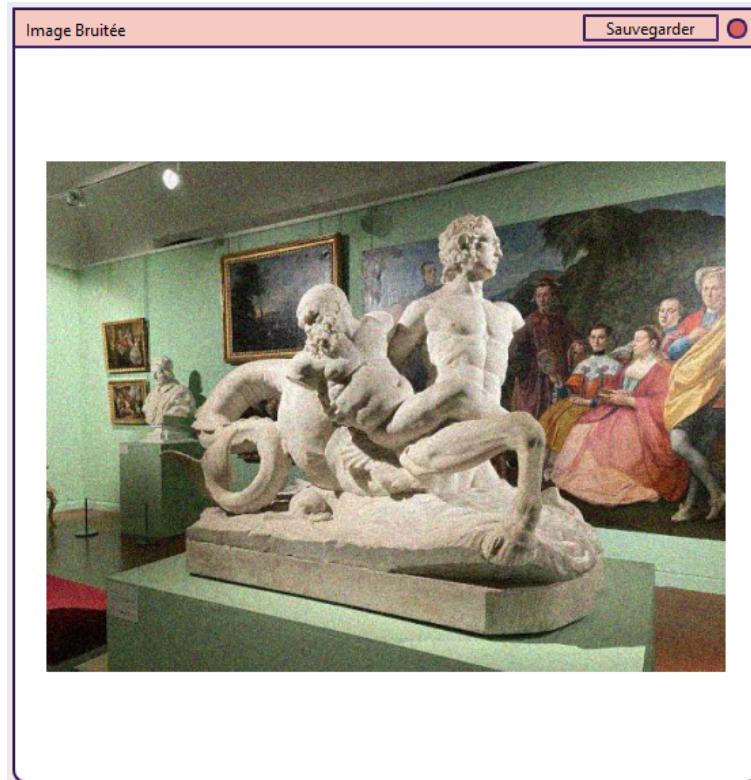
violette et une autre bleue. La première affiche l'image sélectionnée sur notre application tandis que la seconde permet d'accéder aux options de modification de la photo.



Nous pouvons alors ajouter ou non du bruit sur la photo originale. On peut alors modifier les paramètres des générateurs et choisir celui qui nous convient. L'écart-type concerne le bruit gaussien et le bruit chromatique tandis que la densité se réfère au bruit poivre et sel.

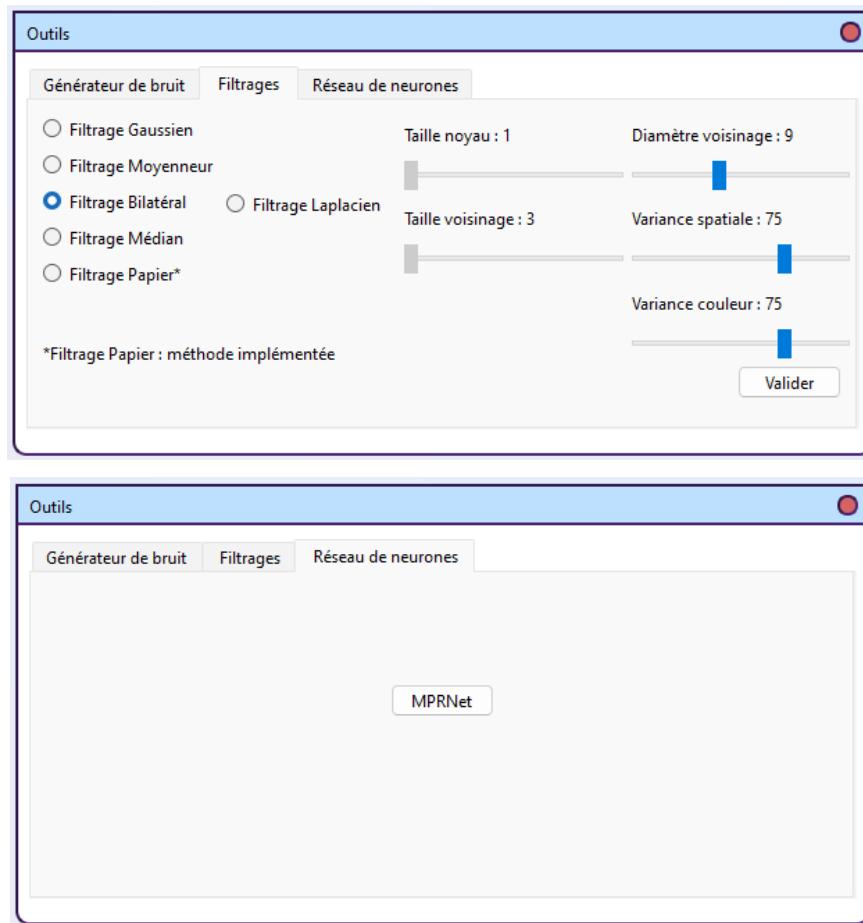


Une fois que l'image a été bruitée, une autre fenêtre de couleur rose s'ouvre et affiche alors le résultat.

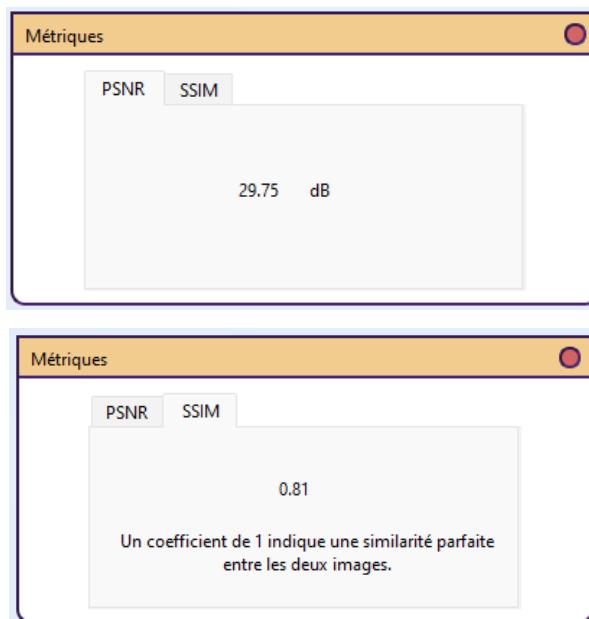


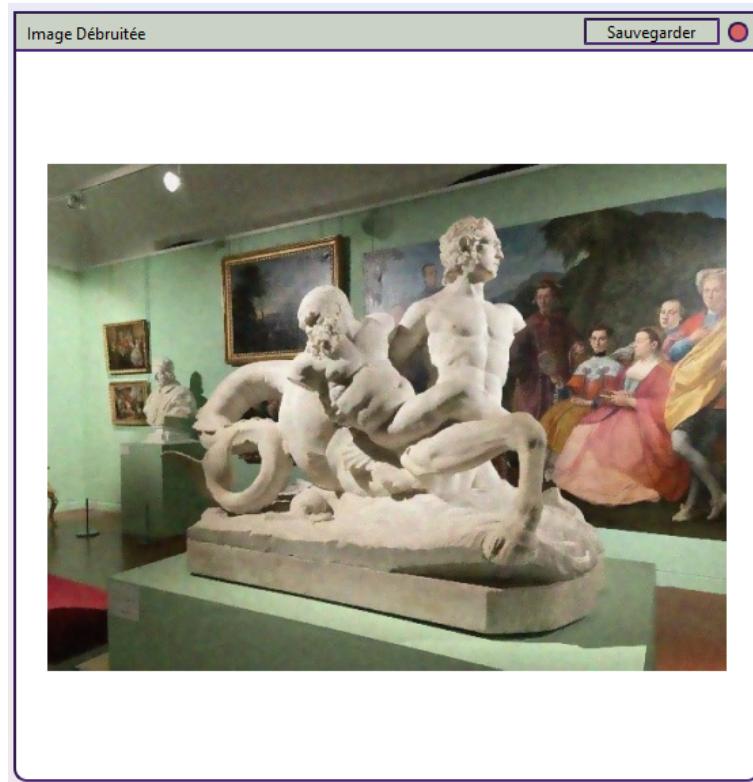
Il est à noter que sur la fenêtre des outils, les onglets *filtrages* et *réseau de neurones* sont visibles mais désactivés tant que l'on a pas appliqué une option de bruitage. De plus, si l'image originale que l'on a choisie est une image que l'on souhaite débruiter et qui n'a donc pas besoin de bruit supplémentaire, un bouton image déjà bruitée a été mis en place. La fenêtre violette disparaît et l'image s'affiche ainsi seulement sur la rose.

Après la mise en place du bruit, on peut alors choisir de filtrer ou d'utiliser le réseau de neurones. En ce qui concerne les filtres, les sliders sont activés ou désactivés s'ils correspondent au filtre sélectionné. De plus, même si un bouton valider est présent, la mise à jour du résultat se fait automatiquement.



Une nouvelle fenêtre de couleur verte apparaît, affichant le résultat mais également une autre orange pour montrer les métriques.

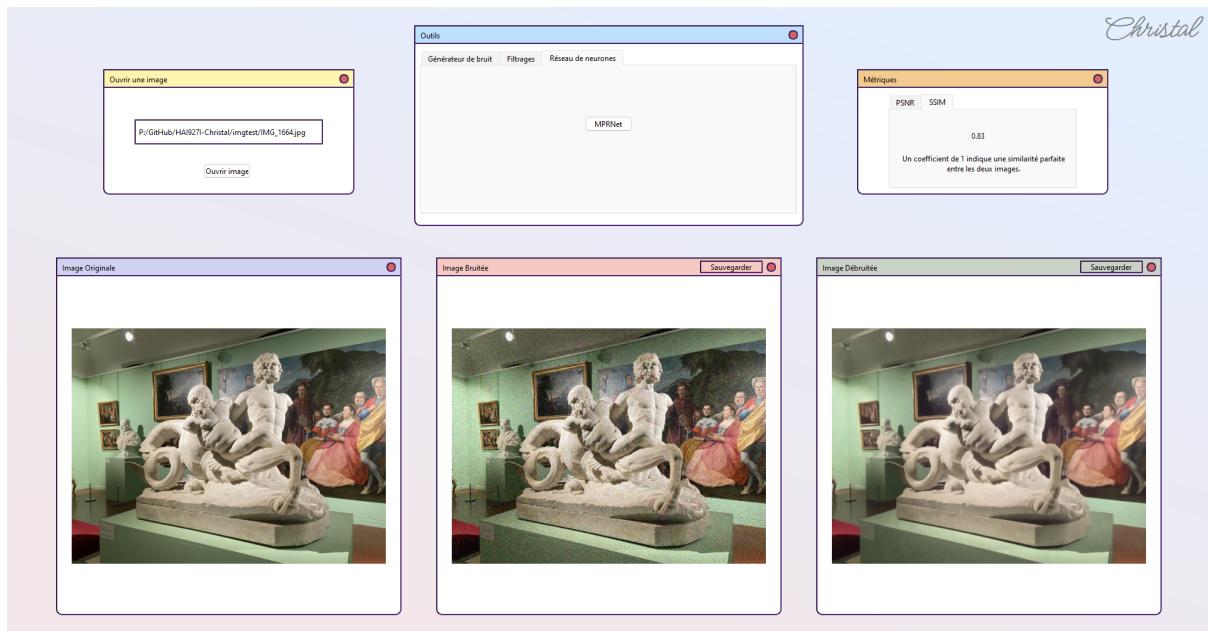




Il est à noter que des boutons “sauvegarder” sont cliquables sur les fenêtres bruitée et débruitée, permettant ainsi à l’utilisateur de les enregistrer à l’endroit même où se situe l’image originale.

```
def sauvegardeImage(self, image, mode):
    options = QFileDialog.Options()
    if(mode == 1):
        mode_nom = "bruitée"
    else:
        mode_nom = "débruitée"
    file_name, _ = QFileDialog.getSaveFileName(self, "Enregistrer l'image " +
mode_nom, self.ImageIn_path, "Images (*.jpg)", options=options)
```

Voici donc le rendu global de l’application :



## NIMA et métriques

Durant cette semaine, nous avons essayé de trouver une implémentation fonctionnelle d'un NIMA.

Un Neural Image Assessment est un modèle conçu pour évaluer la qualité esthétique d'une image. L'objectif de NIMA est de simuler le jugement humain en attribuant une note de qualité à une image donnée.

Ce modèle est généralement entraîné sur des images déjà annotées, on va demander à des gens de donner des scores à des images et notre réseau va apprendre à noter des images en se basant sur leurs caractéristiques visuelles.

Ce modèle nous paraît pertinent pour notre projet, effectivement, il est possible que lors du débruitage d'une image, nous ne possédions pas l'image "ground truth" correspondante, il sera donc intéressant d'avoir une métrique pour évaluer nos résultats.

Voilà à quoi ressemble les résultats fournis avec NIMA :



4.008 +- (1.849)



4.464 +- (1.828)



3.003 +- (2.199)



3.243 +- (1.828)



4.879 +- (1.699)



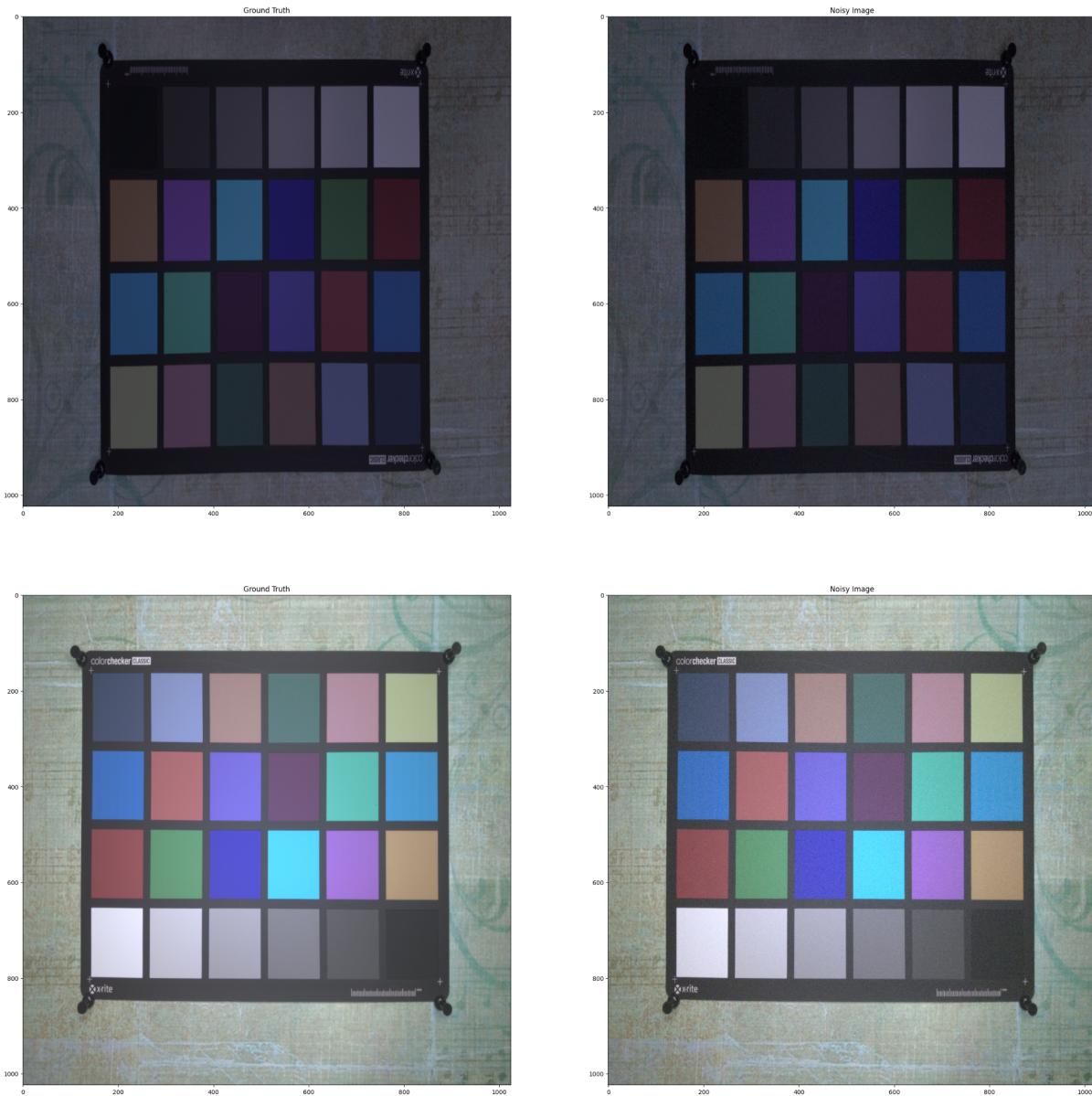
2.033 +- (1499)

Pour le moment, nous avons tenté de faire fonctionner plusieurs projets que nous avons récupérés sur github, des versions utilisant Tensorflow ou bien PyTorch, mais nous n'avons pas encore réussi à en faire fonctionner un pour l'instant.

Nous rencontrons différentes erreurs liées aux versions utilisées qui ne sont plus forcément compatibles, l'utilisation de fonctions qui n'existent plus, etc... Certains projets proposent des fichiers de requirements mais là encore, nous nous retrouvons avec des erreurs de packages introuvables.

Nous allons tenter un dernier dépôt se servant de docker afin de ne pas avoir de problème avec l'environnement que nous allons utiliser, mais pour le moment, NIMA n'est pas encore fonctionnel sur notre projet et nous ne pouvons donc pas le tester avec nos images.

Nous avons également trouvé le dataset que nous allons utiliser pour tester nos différents filtres, le *Smartphone Image Denoising Dataset*. Ce dataset propose des paires d'images avec une image "ground truth" donc sans bruit et qui nous sera utile pour calculer nos métriques, et une image bruitée.



Pour le moment, nous avons utilisé le *SIDD-Small Dataset* qui propose 160 paires d'images.

Nous avons décidé de tester nos différents filtres (bilatéral, gaussien, moyenneur, médian, filtre du papier et MPRNet) et de calculer différentes métriques pour évaluer nos résultats.

Pour le moment, nous calculons :

- Le SNR afin de mesurer la qualité de notre image en comparant le signal au bruit
- Le PSNR pour calculer la qualité de la reconstruction d'une image en comparant la différence entre l'image originale et l'image reconstruite.
- Le RMSE dans le but de calculer la différence moyenne entre les pixels des deux images.
- Le SSIM pour mesurer la similarité structurelle entre deux images en considérant la luminance, le contraste et la structure.

L'utilisation de ces différentes métriques nous paraît suffisante afin d'évaluer correctement les résultats de nos différentes méthodes, nous souhaiterions donc y ajouter en dernier les résultats fournis par NIMA pour avoir une métrique plus pour le côté esthétique, mais pour le moment, c'est encore en cours.

## Poster

En ce qui concerne notre poster, celui-ci est en cours de développement. Nous utilisons Photoshop pour le faire. Nous avons choisi de reprendre les couleurs de nos précédentes présentations.

Nous essayons actuellement de faire rentrer les informations suivantes de manière claire et concise : bruits, filtres (notamment Efficient Poisson Denoising for Photography), explications, réseau de neurones (MPRNet), résultats. Il y a eu beaucoup de réflexion tout au long de la semaine ainsi que de multiples modifications sur les parties, que ce soit au niveau du texte que de la répartition sur l'affiche.

Afin de garder le suspens sur notre poster final, voici une version floutée de la progression la réalisation de celui-ci :



## Pour la semaine prochaine

Nous allons mettre en forme nos résultats, tenter de faire fonctionner NIMA, finir notre poster et préparer notre présentation orale.

## Sources

- |   |       |           |          |
|---|-------|-----------|----------|
| -Smartphone   | Image | Denoising | Dataset: |
| <a href="https://www.eecs.yorku.ca/~kamel/sidd/dataset.php">https://www.eecs.yorku.ca/~kamel/sidd/dataset.php</a>   |       |           |          |
| -Introducing  | NIMA: | Neural    | Image    |
| <a href="https://blog.research.google/2017/12/introducing-nima-neural-image-assessment.html">https://blog.research.google/2017/12/introducing-nima-neural-image-assessment.html</a> |       |           |          |