

# Christal

Compte rendu n°5

## Ajout générateurs de bruit images couleurs

Afin d'avoir plus de choix et de comparaisons dans le bruit couleur, nous avons adapté deux de nos filtres en niveaux de gris pour les images en couleurs.

Nous avons ajouté le bruit gaussien qui ressemble au bruit chromatique mais dans celui-ci on ajoute la même valeur aléatoire à chaque canal (RGB) contrairement au chromatique où chaque canal (RGB) prend une valeur aléatoire.



Gaussien ( $\sigma = 60$ )



Chromatique ( $\sigma = 60$ )

De plus, nous avons ajouté le bruit poivre et sel aux images couleurs.



Poivre et Sel (densité = 0.05)

## Fin de l'implémentation du papier

Lors de la semaine dernière, nous avons eu un problème lors de l'implémentation du papier, nous n'avions pas réussi à bien implémenter la méthode qu'ils proposaient pour déterminer les zones sans textures, ce qui posait un problème lors du calcul du facteur de gain.

Avec les conseils de Monsieur Puech, nous avons essayé une méthode différente, nous avons donc calculé la variance dans différentes régions de l'image choisie au hasard et nous avons ensuite sélectionné les zones où la variance est en dessous d'une certaine valeur afin d'avoir les zones sans textures.

En faisant cela, nous arrivons à obtenir les mêmes résultats que ceux présentés dans le papier :



Dans l'ordre, nous avons l'image de départ, l'image avec un filtre bilatéral, l'image avec un filtre bilatéral après transformation avec le facteur de gain et la dernière image, résultat de l'implémentation du papier.

Nous avons donc des résultats équivalents aux résultats du papier, nous avons ensuite adapté un peu le code pour qu'il puisse fonctionner avec des images en couleur, voilà les résultats que nous avons obtenus, ici pour un bruit chromatique avec écart-type à 20 :

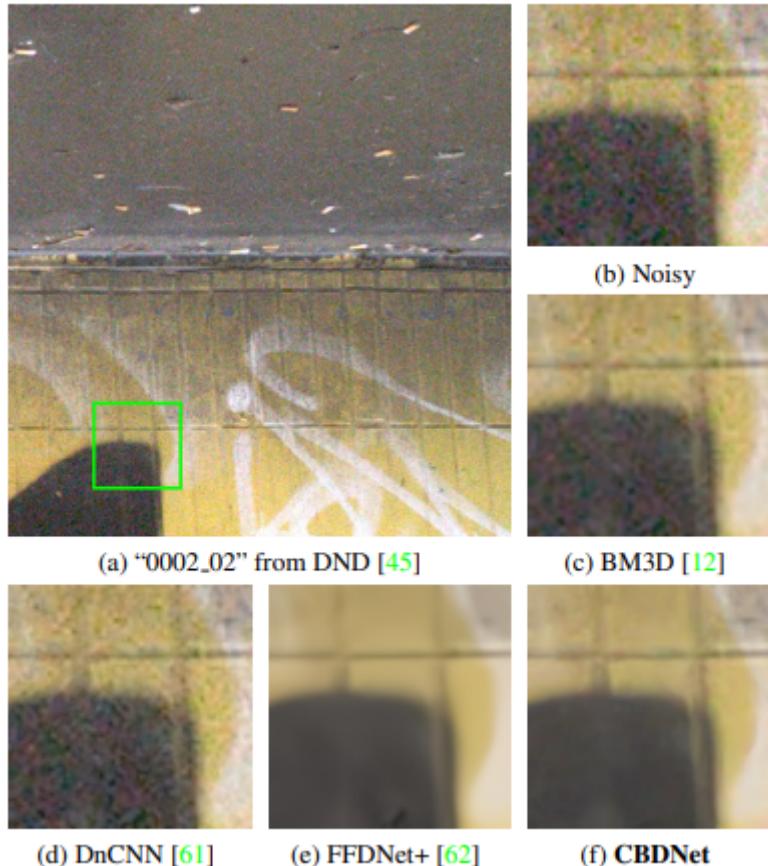


## Deep learning

Dans un premier temps, nous avons essayé d'installer et d'utiliser la solution proposée dans le papier *Toward Convolutional Blind Denoising of Real Photographs*. Pour rappel, dans ce papier, les auteurs proposent un réseau de débruitage convolutionnel (CBDNet) entraîné avec un modèle de bruit plus réaliste et des paires d'images bruitées et non bruitées. Ils intègrent également un sous-réseau d'estimation du bruit pour améliorer la précision de la réduction du bruit.

Cela nous a pris pas mal de temps pour trouver une version python qui fonctionne, à télécharger et installer. Malheureusement, une fois que nous avons réussi à obtenir une version fonctionnelle et applicable à des images de n'importe quelle taille, les résultats que l'on a obtenus sur nos images n'étaient vraiment pas concluants. En effet, leur modèle étant entraîné sur des images avec lesquelles le bruit n'est pas toujours très important et où il faut zoomer sur l'image pour voir une différence. C'est donc cela qui pourrait expliquer les résultats peu convaincants que l'on a obtenus, quand on regarde les résultats qu'ils obtiennent, on peut voir

qu'ils zooment toujours sur une zone très petite, on va donc avoir une qualité globale de l'image améliorée, mais cela ne se verra pas toujours au premier coup d'œil.

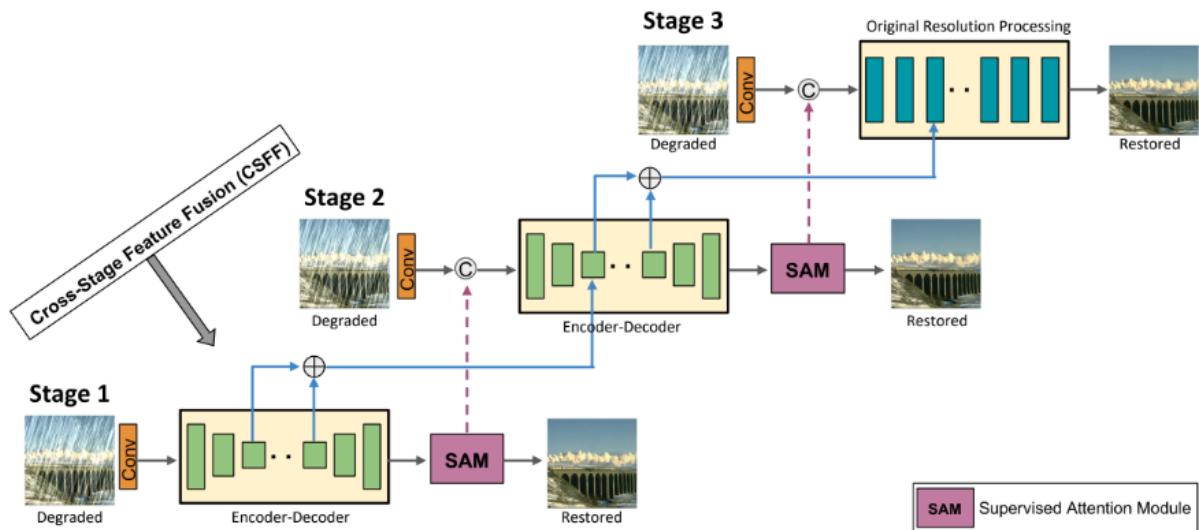


En conclusion, les résultats de ce modèle ne correspondent pas vraiment à ce que l'on souhaite pour notre application, nous avons donc continué nos recherches et nous avons trouvé le modèle MPRNet provenant du papier *Multi-Stage Progressive Image Restoration*.

Le Multi-Stage Progressive Image Restoration Network (MPRNet) est une architecture de réseau neuronal convolutionnel (CNN) en trois étapes conçue spécifiquement pour des tâches de restauration d'images. En effet, les caractéristiques clés de MPRNet comprennent sa structure en trois étapes et sa capacité à traiter plusieurs types d'artéfacts avec une seule architecture.

Avec ce réseau, il est ainsi possible de faire du débruitage, du défloutage ou de la "dépluie" (suppression des artéfacts de pluie). Nous nous intéresserons particulièrement au premier point dans le cadre de notre application.

## Proposed Architecture



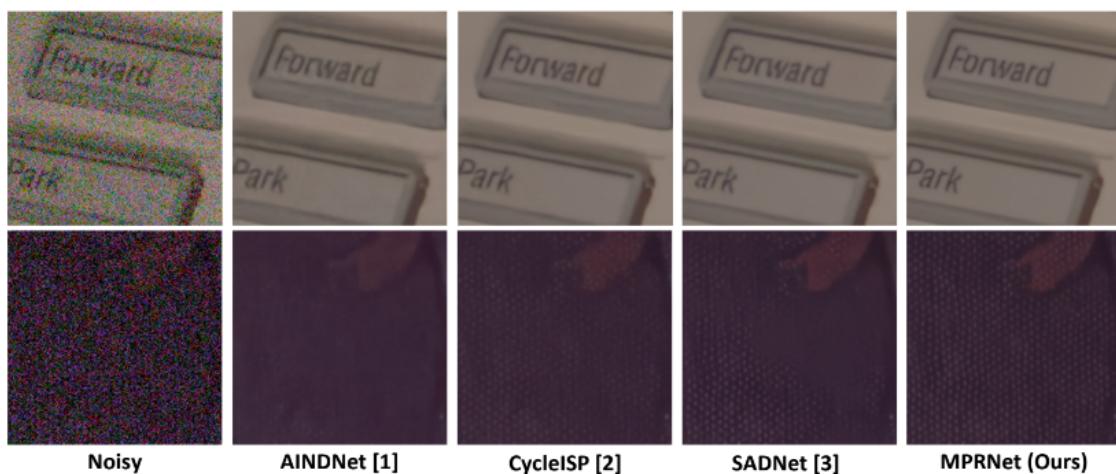
Ce type d'architecture multi-étapes peut capturer et traiter des informations à différentes échelles, permettant au réseau d'apprendre des caractéristiques et des motifs complexes dans les données.

La structure multi-étapes de MPRNet offre plusieurs caractéristiques clés :

- 1) Un encodeur-décodeur pour apprendre des informations contextuelles à plusieurs échelles dans les deux premières étapes : L'encodeur capture les informations à différentes échelles, tandis que le décodeur restaure l'image en utilisant ces informations.
- 2) Préservation des détails spatiaux fins de l'image d'entrée en travaillant à la résolution d'origine dans la dernière étape : La dernière étape du réseau est conçue pour traiter l'image à sa résolution d'origine, ce qui permet de préserver les détails spatiaux fins.
- 3) Un module d'attention supervisée (SAM) : Il prend en compte les caractéristiques apprises à chaque étape et les utilise pour mettre à jour les caractéristiques ultérieures, facilitant ainsi un apprentissage progressif et adaptatif.

- 4) Fusion de caractéristiques entre les étapes Cross-Stage Feature Fusion (CSFF) pour propager les caractéristiques contextualisées à plusieurs échelles de l'étape précoce à l'étape tardive : Cela permet de tirer parti des informations apprises à différentes résolutions et échelles.

En combinant ces éléments, MPRNet vise à offrir une approche complète pour la restauration d'images, prenant en compte à la fois les informations à grande échelle et les détails fins de l'image. Sa structure modulaire permet une certaine flexibilité et une adaptation à diverses tâches de restauration d'images.



[1] Kim et al., "Transfer learning from synthetic to real-noise denoising with adaptive instance normalization", CVPR, 2020.

[2] Zamir et al., "CycleISP: Real image restoration via improved data synthesis", CVPR, 2020.

[3] Chang et al., "Spatial-adaptive network for single image denoising", ECCV, 2020.

Les auteurs proposent donc trois modèles qu'ils ont entraînés avec les paramètres suivants :

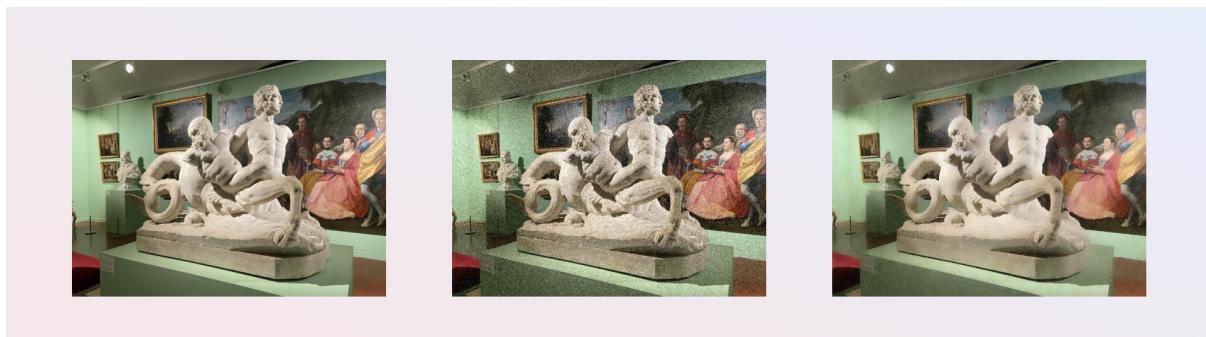
- un patch size de 256\*256 qui correspond donc à la taille de chaque sous région que l'on va extraire de l'image pour entraîner le modèle
- un batch size de 16 qui correspond au nombre d'itérations avant de mettre à jour les poids du modèle
- un nombre d'epochs de 400 000 qui correspond au nombre de fois que l'ensemble complet des données d'entraînements est présenté au modèle
- et enfin, ils utilisent l'optimiseur Adam avec un taux d'apprentissage de 0.0002 qui diminue progressivement à 0.000001, cet optimiseur va permettre d'ajuster les poids du modèle et avoir un taux d'apprentissage qui diminue comme ça au fil de

l'entraînement va permettre d'obtenir une convergence efficace tout en évitant des problèmes liés à des taux trop bas ou trop élevés.

## Interface graphique

Nous avons décidé de remanier toute l'application afin qu'elle soit cohérente concernant les résultats que nous souhaitons mettre en avant.

Dans un premier temps, nous avons décidé de rajouter une troisième fenêtre pour afficher le résultat d'une image dont nous venons de générer un bruit. Nous avons donc : Image entrée > Image bruitée > Image sortie.

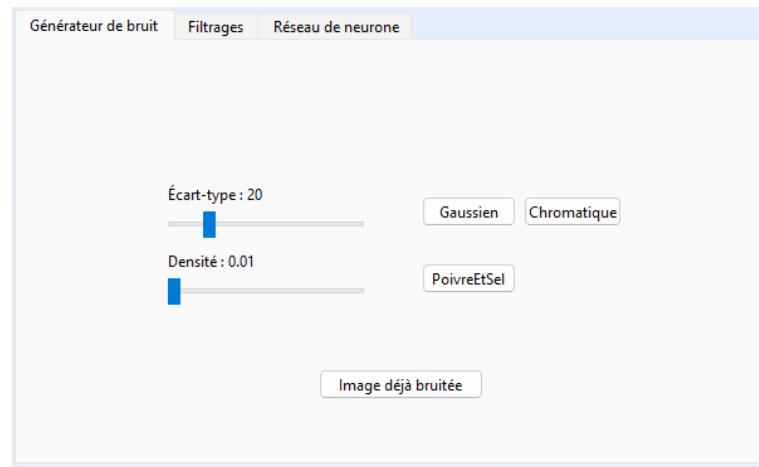


Il a fallu ainsi modifier toute une partie du code pour appliquer les bruits et afficher les résultats sur cette image en particulier et ensuite faire en sorte que les filtres soient appliqués sur cette image et que les résultats obtenus aillent sur l'image de sortie que l'on avait déjà.

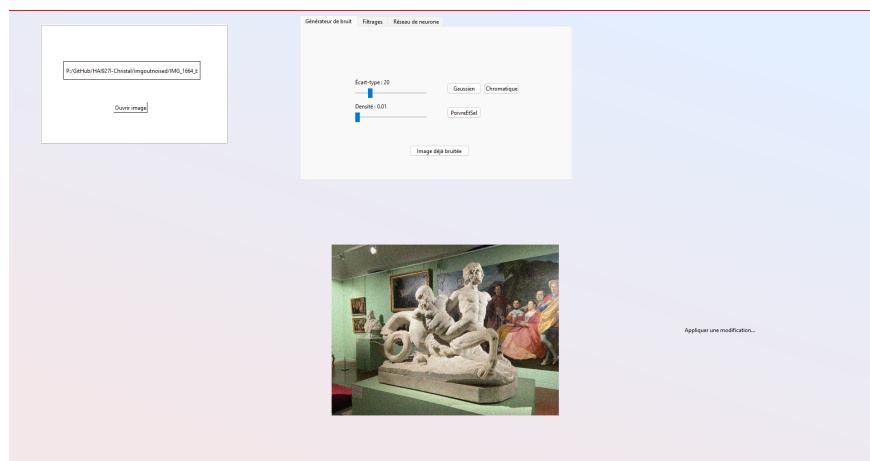
Nous stockons ainsi une seconde image temporaire dans le dossier temp de l'ordinateur. Nous la nommons "ImgChristalTmpNoisy.jpg".

```
def enregistrerImageTmpNoisy(image):
    chemin_dossier_temp = tempfile.gettempdir() + "\ImgChristalTmpNoisy.jpg"
    print(chemin_dossier_temp)
    image.save(chemin_dossier_temp)
```

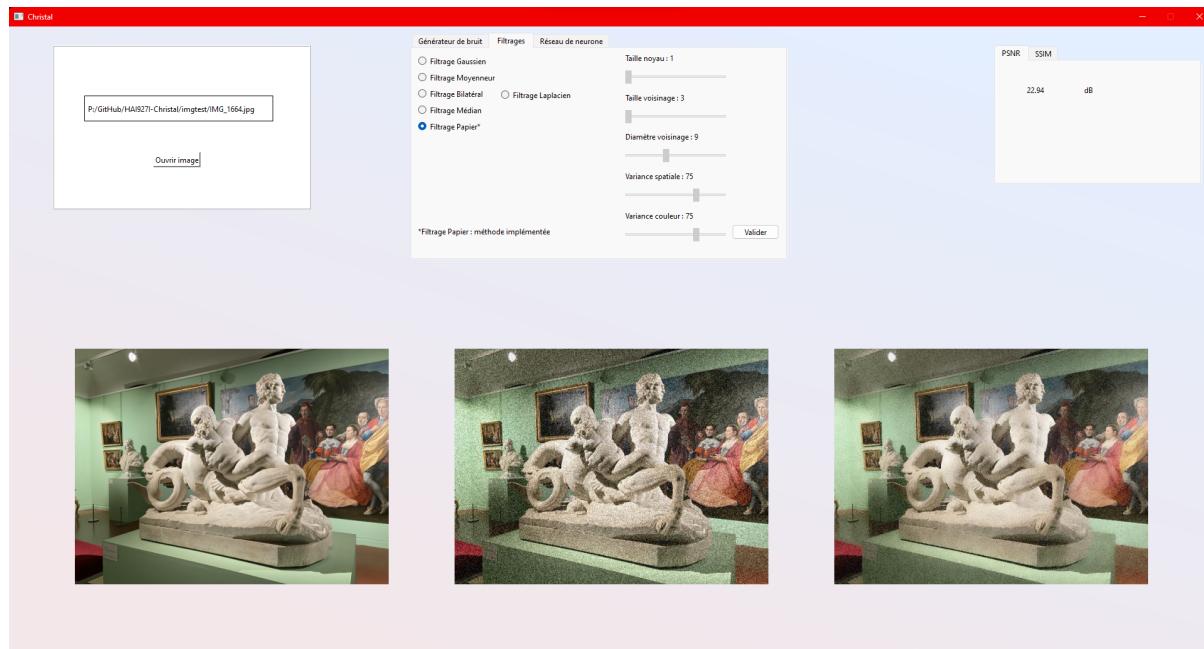
Dans le cas où dès le départ nous avons décidé d'importer une image avec du bruit, nous pouvons ainsi choisir de ne pas générer de bruit supplémentaire en appuyant sur un nouveau bouton.



Ce bouton permet alors de supprimer le premier affichage de l'image pour le décaler sur l'affichage de l'image bruitée.



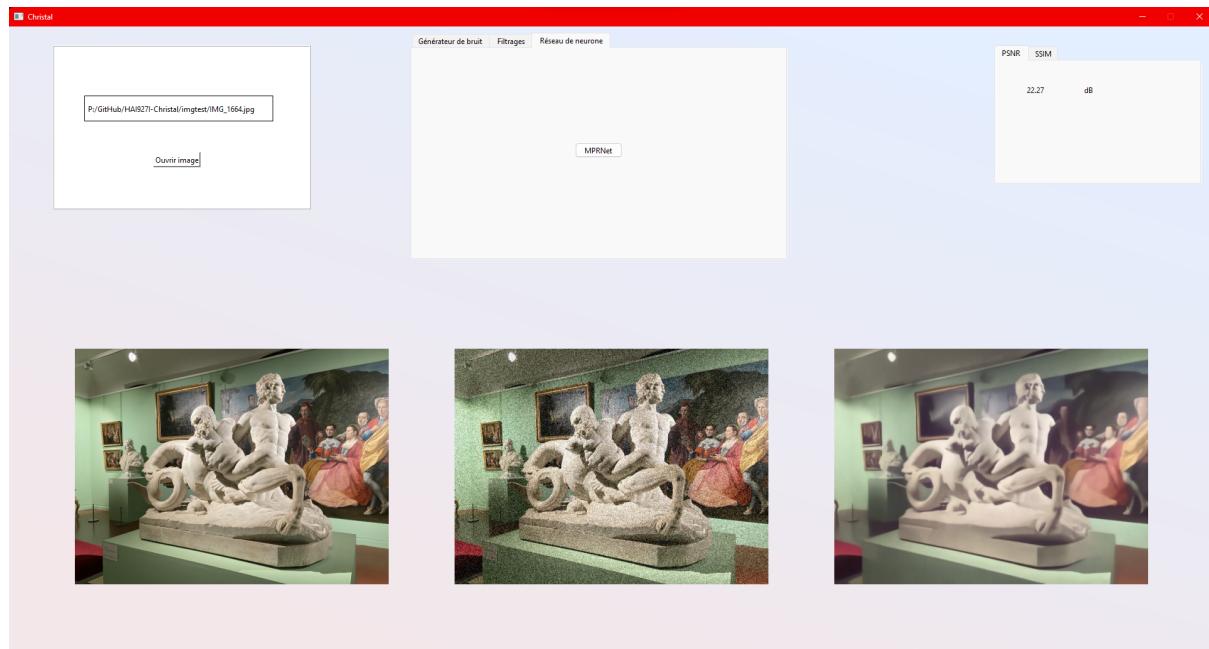
Nous avons également adapté et ajouté notre méthode papier à nos filtres traditionnels.



Cette méthode prend plus de temps que les autres pour s'appliquer sur l'image et on doit alors attendre que l'affichage se mette à jour.

De plus, nous avons implémenté le réseau de neurones “MPRNet”. Nous avons modifié une partie du code source d'origine afin de pouvoir l'appliquer à l'image que nous avons importée dans notre application et non plus à l'ensemble d'un dossier. Aussi, nous avons pour le moment choisi de ne mettre que Denoising. Celle-ci prend également du temps à générer l'image de sortie.

Nous avons gardé les paramètres d'origine utilisés dans le modèle pré-entraîné téléchargeable sur le github du réseau. Ce modèle est sauvegardé dans le fichier “model\_denoising.pth”.



Maintenant que nous avons nos trois images, nous pouvons calculer avec plus de cohérence les métriques. Nous appliquons ainsi ces derniers à l'image d'entrée et l'image de sortie.

Nous avons également ajouté un bouton permettant de sauvegarder les images bruitées et débruitées.

## Pour la semaine prochaine

Nous allons mettre en place nos tests sur différents datasets pour analyser les résultats de nos différentes méthodes, nous allons également nous occuper de la réalisation de notre poster scientifique.

## Sources

-Efficient poisson denoising for photography: [EFFICIENT POISSON DENOISING FOR PHOTOGRAPHY H. Talbot<sub>1</sub>, H. Phelippeau<sub>1</sub>, M. Akil<sub>1</sub>, S. Bara<sub>2</sub>](#)

-Toward Convolutional Blind Denoising of Real Photographs:  
<https://www4.comp.polyu.edu.hk/~cslzhang/paper/CVPR19-CBDNet.pdf>

-Multi-Stage Progressive Image Restoration:  
[https://openaccess.thecvf.com/content/CVPR2021/papers/Zamir Multi-Stage Progressive Image Restoration CVPR 2021 paper.pdf](https://openaccess.thecvf.com/content/CVPR2021/papers/Zamir_Multi-Stage_Progressive_Image_Restoration_CVPR_2021_paper.pdf)  
[https://drive.google.com/file/d/1-L43wj-VTppkrR9AL6cPBjl2RJi3Hc\\_z/view](https://drive.google.com/file/d/1-L43wj-VTppkrR9AL6cPBjl2RJi3Hc_z/view)