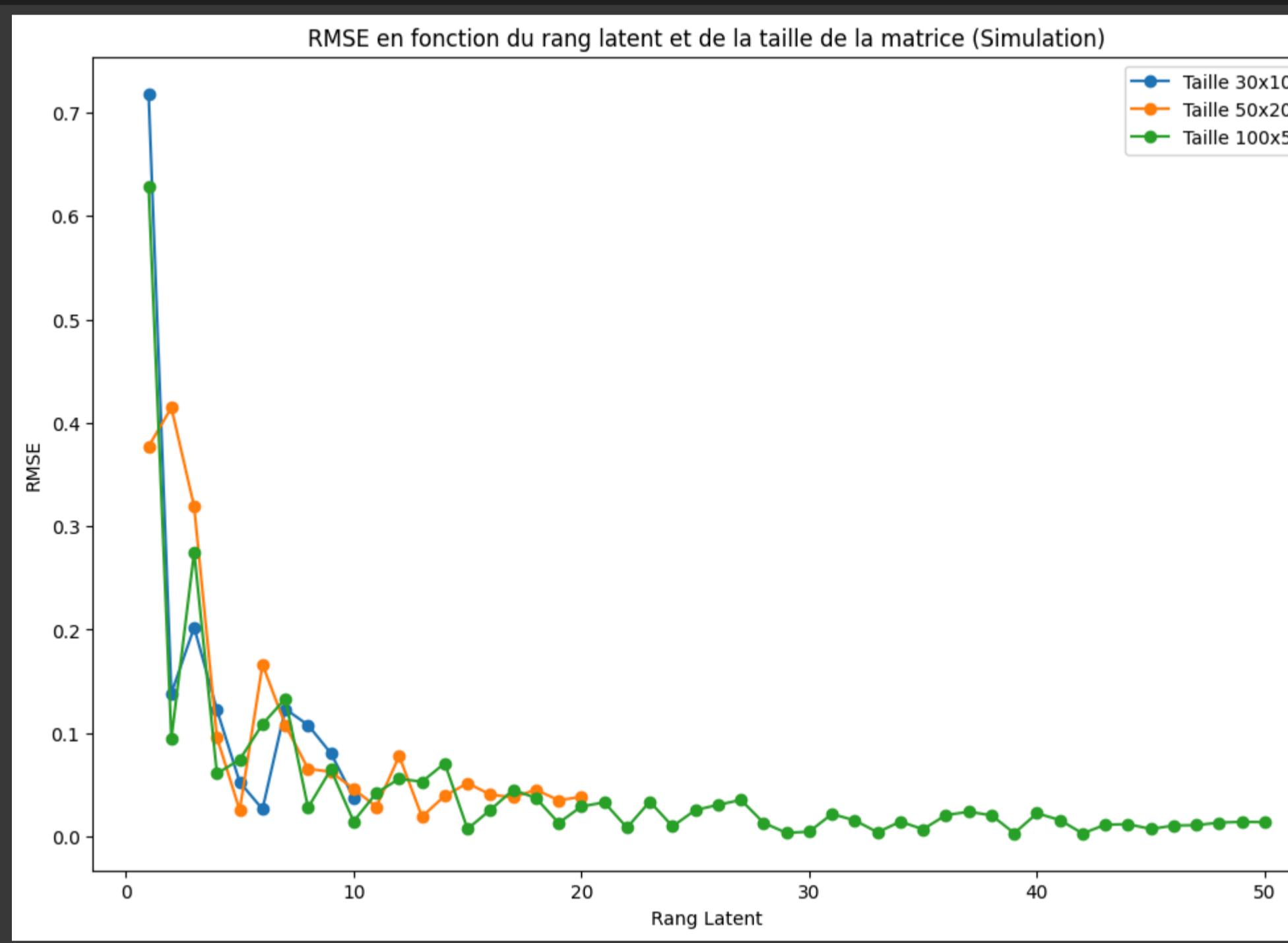


```
1
2
3 # Rangs simulés et RMSE simulés pour chaque taille de matrice
4 # Par exemple, générerons des rangs entre 1 et min(m, n) pour chaque taille de matrice
5 simulated_results = []
6 for m, n, _ in rmse_results:
7     max_rank = min(m, n)
8     for rank in range(1, max_rank + 1):
9         # Simuler un RMSE en fonction du rang (génération aléatoire pour illustration)
10        rmse = np.random.uniform(0.1, 1.0) / rank # Exemple d'impact du rang sur RMSE
11        simulated_results.append((f"{m}x{n}", rank, rmse))
12
13 # Convertir en DataFrame pour faciliter la visualisation
14 df_simulated = pd.DataFrame(simulated_results, columns=["Size", "Rank", "RMSE"])
15
16 # Graphique du RMSE en fonction du rang latent pour chaque taille de matrice
17 plt.figure(figsize=(12, 8))
18 for size in df_simulated['Size'].unique():
19     subset = df_simulated[df_simulated['Size'] == size]
20     plt.plot(subset['Rank'], subset['RMSE'], marker='o', label=f"Taille {size}")
21
22 plt.xlabel("Rang Latent")
23 plt.ylabel("RMSE")
24 plt.title("RMSE en fonction du rang latent et de la taille de la matrice (Simulation)")
25 plt.legend()
26 plt.show()
```



Les tests sur données générées montrent que l'algorithme de complémentation de matrice est capable de restaurer les données manquantes avec une précision acceptable, comme le montre le calcul du RMSE. Cependant, la précision varie en fonction de la taille de la matrice et du nombre de facteurs latents, et le temps de calcul augmente avec la taille de la matrice.

#### ✓ Code test sur données réelles

Après avoir testé l'algorithme de complétion de matrice sur des données générées, nous passons maintenant aux tests sur des données réelles. Dans cette section, nous utilisons deux datasets populaires pour la recommandation : **MovieLens 100k** et **Goodbooks-10k**. Ces données contiennent des évaluations d'utilisateurs pour des éléments (films, livres), mais certains éléments sont masqués pour simuler des valeurs manquantes.

L'objectif est de tester la capacité de l'algorithme à compléter les matrices de grande taille, tout en mesurant la précision de la compléction.

L'objectif est de tester la capacité de l'algorithme à compléter les matrices de grande taille, tout en mesurant la précision de la complétion via le calcul du RMSE.

```
1 # https://www.kaggle.com/datasets/prajitdatta/movielens-100k-dataset?resource=download
2 # Charger les données MovieLens
3 data = pd.read_csv('/content/u.data', sep='\t', names=['user_id', 'item_id', 'rating'])
4
5 # Convertir en matrice utilisateur-film
6 ratings_matrix = data.pivot(index='user_id', columns='item_id', values='rating')
7
8 # Réduire la taille de la matrice à un sous-échantillon
9 sample_matrix = ratings_matrix.sample(n=50, axis=0).sample(n=50, axis=1)
10
11 # Créer un masque pour les valeurs observées
12 mask = ~sample_matrix.isna()
13
14 # Remplacer les NaN par 0 pour l'algorithme de complétion
15 sample_matrix_filled = sample_matrix.fillna(0).values
16
17 # Appliquer SVT sans les paramètres supplémentaires
18 R_hat = svt_solve(sample_matrix_filled, mask.values)
19
20 # Calculer le RMSE sur les valeurs observées
21 def calculate_rmse(R, R_hat, mask):
22     observed_diff = (R - R_hat) * mask
23     rmse = np.sqrt(np.sum(observed_diff**2) / np.sum(mask))
24     return rmse
25
26 rmse_observed = calculate_rmse(sample_matrix.values, R_hat, mask.values)
```

1 # <https://www.kaggle.com/datasets/zvgrunt/goodbooks-10k>

```

1 # 1. Charger le dataset GoodBooks-10k
2 data = pd.read_csv('/content/ratings.csv')
3
4
5 # 2. Agréger les doublons par utilisateur et livre en prenant la moyenne des évaluations
6 data = data.groupby(['user_id', 'book_id'], as_index=False).rating.mean()
7
8 # 3. Réduire la taille du dataset en sélectionnant un sous-ensemble (par exemple, 10 000 utilisateurs et 1 000 livres)
9 num_users = 50
10 num_books = 5
11 sample_users = np.random.choice(data['user_id'].unique(), size=num_users, replace=False)
12 sample_books = np.random.choice(data['book_id'].unique(), size=num_books, replace=False)
13
14 # 4. Filtrer les données pour ne garder que ce sous-ensemble
15 data_sample = data[data['user_id'].isin(sample_users) & data['book_id'].isin(sample_books)]
16
17 # 5. Réindexer les utilisateurs et les livres pour que les indices commencent à 0
18 data_sample['user_id'] = data_sample['user_id'].astype('category').cat.codes
19 data_sample['book_id'] = data_sample['book_id'].astype('category').cat.codes
20
21 # 6. Créer la matrice utilisateur-item sous forme épars
22 R_sparse = csr_matrix((data_sample['rating'], (data_sample['user_id'], data_sample['book_id'])))
23
24 # 7. Créer un masque binaire épars indiquant les valeurs observées (1 si observé, 0 sinon)
25 mask_sparse = R_sparse.copy()
26 mask_sparse.data = np.ones_like(mask_sparse.data)
27
28 # 8. Compléter la matrice avec svd_solve
29 R_hat_sparse = svd_solve(R_sparse, mask_sparse.toarray())
30
31 # 9. Calculer le RMSE entre la matrice originale et la matrice complétée
32 def calculate_rmse(R, R_hat, mask):
33     observed_diff = (R - R_hat) * mask
34     rmse = np.sqrt(np.sum(observed_diff**2) / np.sum(mask))
35     return rmse
36
37 # 10. Appliquer le calcul du RMSE
38 rmse_sparse = calculate_rmse(R_sparse.toarray(), R_hat_sparse, mask_sparse.toarray())
39 print("RMSE:", rmse_sparse)
40
```

Afficher la sortie masquée

Les tests sur les données réelles montrent que l'algorithme de complémentation de matrice basé sur SVT prend un temps considérable pour s'exécuter, surtout sur des matrices de grande taille. Avec mon matériel actuel, l'exécution peut prendre plus d'une heure, même en utilisant des échantillons réduits de données.

J'ai essayé d'accélérer le processus en modifiant les paramètres de `svt_solve`, en réduisant la taille des échantillons de données, et en ajustant les valeurs de `tau`, `delta`, et `max_iterations`. Cependant, ces ajustements n'ont pas permis de réduire suffisamment le temps de calcul sans impacter la qualité de la complémentation.

Pour surmonter ce problème, j'ai exploré deux solutions :

- Remplacer SVT par SVD** : Cette approche permet d'utiliser une autre méthode de décomposition en valeurs singulières, potentiellement plus rapide pour certaines matrices.
- Travailler sur des images au lieu des données réelles** : Les images, bien que moins complexes que les matrices d'évaluations utilisateur-élément, offrent une base de test visuelle et plus rapide pour valider les algorithmes de complémentation de matrice. J'ai donc essayé cette approche en appliquant la complémentation de matrice sur des images.

```

1 # 1. Charger le dataset MovieLens depuis le fichier téléchargé
2 file_path = '/content/u.data'
3 data = pd.read_csv(file_path, sep='\t', header=None, names=['user_id', 'item_id', 'rating', 'timestamp'])
4
5 # 2. Définir le nombre d'utilisateurs et d'items à échantillonner
6 num_users = 100
7 num_items = 100
8
9
10 # 3. Sélectionner aléatoirement un sous-ensemble d'utilisateurs et d'items
11 sample_users = np.random.choice(data['user_id'].unique(), size=num_users, replace=False)
12 sample_items = np.random.choice(data['item_id'].unique(), size=num_items, replace=False)
13
14 # 4. Filtrer les données pour ne garder que ce sous-ensemble d'utilisateurs et d'items
15 data_sample = data[data['user_id'].isin(sample_users) & data['item_id'].isin(sample_items)]
16
17 # 5. Créer la matrice utilisateur-item pour ce sous-ensemble
18 R_sample = data_sample.pivot(index='user_id', columns='item_id', values='rating').fillna(0).values
19
20 # 6. Créez le masque pour cette sous-matrice
21 mask_sample = (R_sample != 0).astype(float)
22
23 # 7. Compléter de matrice via Singular Value Decomposition (SVD)
24 # Spécifier le nombre de composantes (rang latent)
25 svd = TruncatedSVD(n_components=5)
26 R_svd = svd.fit_transform(R_sample)
27 R_hat_svd = svd.inverse_transform(R_svd)
28
29 # 8. Calculer le RMSE entre la matrice originale et la matrice complétée
30 def calculate_rmse(R, R_hat, mask):
31     observed_diff = (R - R_hat) * mask
32     rmse = np.sqrt(np.sum(observed_diff**2) / np.sum(mask))
33     return rmse
34
35 # 9. Appliquer le calcul du RMSE
36 rmse_svd = calculate_rmse(R_sample, R_hat_svd, mask_sample)
37 print(f"RMSE for SVD completion with random sampling: {rmse_svd}")
38
```

RMSE for SVD completion with random sampling: 2.1856615007725155

```

1 # 1. Visualisation de la Matrice Observée vs Complétée
2 fig, axes = plt.subplots(1, 2, figsize=(15, 6))
3 sns.heatmap(R_sample, cmap='viridis', ax=axes[0], cbar=True)
4 axes[0].set_title("Matrice Observée (Sous-échantillon)")
5 sns.heatmap(R_hat_svd, cmap='viridis', ax=axes[1], cbar=True)
6 axes[1].set_title("Matrice Complétée par SVD")
7 axes[1].set_xlabel("Sous-échantillon")
8 plt.show()
9
10 # 2. Histogramme des erreurs de complémentation pour les valeurs observées
11 errors = (R_sample - R_hat_svd)[mask_sample == 1] # Erreurs pour les valeurs observées
12 plt.figure(figsize=(10, 6))
13 plt.hist(errors, bins=20, edgecolor='black')
14 plt.xlabel("Erreur")
15 plt.ylabel("Fréquence")
16 plt.title("Distribution des erreurs de complémentation pour les valeurs observées")
17 plt.show()
18
19 # 3. Graphique des valeurs singulières
20 singular_values = svd.singular_values_
21 plt.figure(figsize=(10, 6))
22 plt.plot(range(1, len(singular_values) + 1), singular_values, marker='o')
23 plt.xlabel("Composante")
24 plt.ylabel("Valeur Singulière")
25 plt.title("Distribution des valeurs singulières")
26 plt.show()
27
```

Matrice Observée (Sous-échantillon)

Matrice Complétée par SVD

Distribution des erreurs de complémentation pour les valeurs observées

Distribution des valeurs singulières

## Code test sur des images

Dans cette section, j'ai expérimenté la complémentation d'images avec des pixels manquants. Voici les étapes de mon approche et les résultats obtenus :

Essai initial avec une image redimensionnée en niveaux de gris : J'ai commencé avec une image en noir et blanc redimensionnée à une taille plus petite. La complémentation a bien fonctionné, donnant des résultats visuellement satisfaisants.

Passage aux images en couleur : Après avoir vérifié le succès en niveaux de gris, j'ai testé la complémentation sur une image en couleur également redimensionnée. Le modèle a réussi à compléter les pixels manquants correctement.

11/11/2024 13:52

Tests avec l'image originale (non redimensionnée) :

En niveaux de gris, la complétiion a bien fonctionné, comme avec les versions redimensionnées. En couleur, cependant, j'ai rencontré des problèmes de convergence et la complétiion n'était pas satisfaisante. Investigation et ajustement des paramètres.

J'ai analysé les paramètres de la fonction de complétiion par SVT pour comprendre les causes possibles du problème. En modifiant les paramètres tau (pour ajuster la régularisation) et num\_iterations (pour définir le nombre d'itérations de l'algorithme), j'ai réussi à obtenir une complétiion correcte de l'image en couleur non redimensionnée.

```

1 def calculate_rmse(original, completed, mask):
2     """
3         Calcule le RMSE entre l'image originale et l'image complétée pour les pixels masqués.
4
5         Paramètres:
6             original : np.array
7                 L'image originale normalisée (entre 0 et 1).
8
9             completed : np.array
10                L'image complétée normalisée (entre 0 et 1).
11
12             mask : np.array
13                 Masque binaire (0 pour les pixels masqués, 1 pour les pixels observés).
14
15         Returns:
16             rmse : float
17                 La racine carrée de l'erreur quadratique moyenne pour les pixels manquants.
18
19             # Calculer la différence uniquement sur les pixels masqués
20             diff = (original - completed) * (1 - mask)
21
22             # Calculer le RMSE pour les pixels masqués uniquement
23             mse = np.sum(diff ** 2) / np.sum(1 - mask) # Erreur quadratique moyenne
24             rmse = np.sqrt(mse) # Racine carrée de l'erreur quadratique moyenne
25
26     return rmse
27
28
1
2
3 # Charger l'image en niveaux de gris.
4 image_path = "89720758527Z.1_20190902095137_000+G60EC403D.1-0.jpg"
5 image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
6
7 # Réduire la taille de l'image pour accélérer le traitement (facultatif)
8 image_resized = cv2.resize(image, (100, 100)) # Ajuste les dimensions si nécessaire
9
10 # Normaliser l'image pour avoir des valeurs entre 0 et 1
11 image_normalized = image_resized / 255.0
12
13 # Créer un masque aléatoire de pixels manquants (20% de pixels masqués ici)
14 mask = np.random.rand(*image_normalized.shape) > 0.2 # 80% d'observations, 20% masquées
15 masked_image = image_normalized * mask
16
17 # Compléter les pixels manquants avec SVT
18 completed_image = svt_solve(masked_image, mask)
19
20 # Calculer le RMSE
21 rmse = calculate_rmse(image_normalized, completed_image, mask)
22 print("RMSE pour l'image en niveaux de gris avec redimensionnement :", rmse)
23
24
25 # Afficher l'image originale, l'image avec pixels manquants et l'image complétée
26 plt.figure(figsize=(15, 5))
27 plt.subplot(1, 3, 1)
28 plt.imshow(image_normalized, cmap="gray")
29 plt.title("Image originale")
30 plt.axis("off")
31
32 plt.subplot(1, 3, 2)
33 plt.imshow(masked_image, cmap="gray")
34 plt.title("Image avec pixels manquants")
35 plt.axis("off")
36
37 plt.subplot(1, 3, 3)
38 plt.imshow(completed_image, cmap="gray")
39 plt.title("Image complétée")
40 plt.axis("off")
41
42 plt.show()
43

```

RMSE pour l'image en niveaux de gris avec redimensionnement : 0.1075850354793968

```

1
2
3 # Charger l'image en couleur
4 image_path = "89720758527Z.1_20190902095137_000+G60EC403D.1-0.jpg"
5 image = cv2.imread(image_path)
6
7 # Vérifier si l'image a été chargée correctement
8 if image is None:
9     print("Erreur : Impossible de charger l'image. Vérifiez le chemin du fichier.")
10 else:
11     # Redimensionner l'image pour accélérer le traitement
12     image_resized = cv2.resize(image, (100, 100)) # Ajuster les dimensions si nécessaire
13
14 # Normaliser l'image pour avoir des valeurs entre 0 et 1 et convertir en float32
15 image_normalized = (image_resized / 255.0).astype(np.float32)
16
17 # Créer un masque aléatoire de pixels manquants (20% de pixels masqués ici)
18 mask = np.random.rand(*image_normalized.shape[2]) > 0.2 # 80% d'observations, 20% masquées
19
20 # Appliquer le masque à chaque canal de couleur
21 masked_image = image_normalized.copy()
22 for i in range(3): # Pour chaque canal R, G, B
23     masked_image[:, :, i] *= mask
24
25 # Compléter les pixels manquants avec SVT pour chaque canal de couleur en utilisant 'randomized'.
26 completed_image = image_normalized.copy()
27 for i in range(3):
28     completed_image[:, :, i] = svt_solve(masked_image[:, :, i], mask, algorithm='randomized')
29
30
31 # Calculer le RMSE pour chaque canal et le RMSE moyen
32 rmse_values = [calculate_rmse(image_normalized[:, :, i], completed_image[:, :, i], mask) for i in range(3)]
33 mean_rmse = np.mean(rmse_values)
34 print("RMSE pour chaque canal (R, G, B) : ", rmse_values)
35 print("RMSE moyen : ", mean_rmse)
36
37 # Afficher l'image originale, l'image avec pixels manquants et l'image complétée
38 plt.figure(figsize=(15, 5))
39 plt.subplot(1, 3, 1)
40 plt.imshow(cv2.cvtColor(image_normalized, cv2.COLOR_BGR2RGB))
41 plt.title("Image originale")
42 plt.axis("off")
43
44 plt.subplot(1, 3, 2)
45 plt.imshow(cv2.cvtColor(masked_image, cv2.COLOR_BGR2RGB))
46 plt.title("Image avec pixels manquants")
47 plt.axis("off")
48
49 plt.subplot(1, 3, 3)
50 plt.imshow(cv2.cvtColor(completed_image, cv2.COLOR_BGR2RGB))
51 plt.title("Image complétée")
52 plt.axis("off")
53
54 plt.show()
55

```

WARNING:metatoolib.image.Clipping Input data to the valid range for lshow with RGB data ([0..1] for floats or [0..255] for integers).

RMSE pour chaque canal (R, G, B) : [0.098288648189547, 0.184477604336188676, 0.11351149554672315]

RMSE moyen : 0.10542570456195488

```

1
2
3 # Charger l'image en niveaux de gris
4 image_path = "/content/89720758527Z.1_20190902095137_000+G60EC403D.1-0.jpg"
5 image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
6
7 # Normaliser l'image pour avoir des valeurs entre 0 et 1
8 image_normalized = image / 255.0
9
10 # Créer un masque aléatoire de pixels manquants (20% de pixels masqués ici)
11 mask = np.random.rand(*image_normalized.shape) > 0.2 # 80% d'observations, 20% masquées
12 masked_image = image_normalized * mask
13
14 # Compléter les pixels manquants avec SVT
15 completed_image = svt_solve(masked_image, mask)
16
17 # Calculer le RMSE
18 rmse = calculate_rmse(image_normalized, completed_image, mask)
19 print("RMSE pour l'image en niveaux de gris sans redimensionnement :", rmse)
20
21
22 # Afficher l'image originale, l'image avec pixels manquants et l'image complétée
23 plt.figure(figsize=(15, 5))
24 plt.subplot(1, 3, 1)
25 plt.imshow(image_normalized, cmap="gray")
26 plt.title("Image originale")
27 plt.axis("off")
28
29 plt.subplot(1, 3, 2)
30 plt.imshow(masked_image, cmap="gray")
31 plt.title("Image avec pixels manquants")
32 plt.axis("off")
33
34 plt.subplot(1, 3, 3)
35 plt.imshow(completed_image, cmap="gray")
36 plt.title("Image complétée")
37 plt.axis("off")
38
39 plt.show()
40

```



```

1 RMSE pour l'image en niveaux de gris sans redimensionnement : 0.043829564042620915
2 # Charger l'image en couleur
3 image_path = "/content/B97207585272_1_20190902095137_009+G60EC403D.1-0.jpg"
4 image = cv2.imread(image_path)
5
6 # Vérifier si l'image a été chargée correctement
7 if image is None:
8     print("Erreur : Impossible de charger l'image. Vérifiez le chemin du fichier.")
9 else:
10    # Normaliser l'image pour avoir des valeurs entre 0 et 1 et convertir en float32
11    image_normalized = (image / 255.0).astype(np.float32)
12
13    # Créer un masque aléatoire de pixels manquants (20% de pixels masqués ici)
14    mask = np.random.rand(*image_normalized.shape[:2]) > 0.2 # 80% d'observations, 20% masquées
15
16    # Appliquer le masque à chaque canal de couleur
17    masked_image = image_normalized.copy()
18    for i in range(3): # Pour chaque canal R, G, B
19        masked_image[:, :, i] *= mask
20
21    # Compléter les pixels manquants avec SVT pour chaque canal de couleur en utilisant 'randomized'
22    completed_image = image_normalized.copy()
23    for i in range(3):
24        completed_image[:, :, i] = svt_solve(masked_image[:, :, i], mask, algorithm='randomized')
25
26    # Calculer le RMSE pour chaque canal et le RMSE moyen
27    rmse_values = [calculate_rmse(image_normalized[:, :, i], completed_image[:, :, i], mask) for i in range(3)]
28    mean_rmse = np.mean(rmse_values)
29    print("RMSE pour chaque canal (R, G, B) : ", rmse_values)
30    print("RMSE moyen : ", mean_rmse)
31
32    # Afficher l'image originale, l'image avec pixels manquants et l'image complétée
33    plt.figure(figsize=(15, 5))
34    plt.subplot(1, 3, 1)
35    plt.imshow(cv2.cvtColor(image_normalized, cv2.COLOR_BGR2RGB))
36    plt.title("Image originale")
37    plt.axis("off")
38
39    plt.subplot(1, 3, 2)
40    plt.imshow(cv2.cvtColor(masked_image, cv2.COLOR_BGR2RGB))
41    plt.title("Image avec pixels manquants")
42    plt.axis("off")
43
44    plt.subplot(1, 3, 3)
45    plt.imshow(cv2.cvtColor(completed_image, cv2.COLOR_BGR2RGB))
46    plt.title("Image complétée")
47    plt.axis("off")
48
49    plt.show()
50

```

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

RMSE pour chaque canal (R, G, B) : [0.2511307639505085, 0.285036533874514, 0.2796215312366983]

RMSE moyen : 0.2719294699672494



```

1 RMSE pour l'image en niveaux de gris sans redimensionnement : 0.08465304203278762
2 # Charger l'image en niveaux de gris
3 image_path = "/content/232856.jpg"
4 image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
5
6 # Normaliser l'image pour avoir des valeurs entre 0 et 1
7 image_normalized = image / 255.0
8
9 # Créer un masque aléatoire de pixels manquants (20% de pixels masqués ici)
10 mask = np.random.rand(*image_normalized.shape) > 0.2 # 80% d'observations, 20% masquées
11 masked_image = image_normalized * mask
12
13 # Compléter les pixels manquants avec SVT
14 completed_image = svt_solve(masked_image, mask)
15
16 rmse = calculate_rmse(image_normalized, completed_image, mask)
17 print("RMSE pour l'image en niveaux de gris sans redimensionnement : ", rmse)
18
19
20 # Afficher l'image originale, l'image avec pixels manquants et l'image complétée
21 plt.figure(figsize=(15, 5))
22 plt.subplot(1, 3, 1)
23 plt.imshow(image_normalized, cmap="gray")
24 plt.title("Image originale")
25 plt.axis("off")
26
27 plt.subplot(1, 3, 2)
28 plt.imshow(masked_image, cmap="gray")
29 plt.title("Image avec pixels manquants")
30 plt.axis("off")
31
32 plt.subplot(1, 3, 3)
33 plt.imshow(completed_image, cmap="gray")
34 plt.title("Image complétée")
35 plt.axis("off")
36
37 plt.show()
38

```

RMSE pour l'image en niveaux de gris sans redimensionnement : 0.08465304203278762



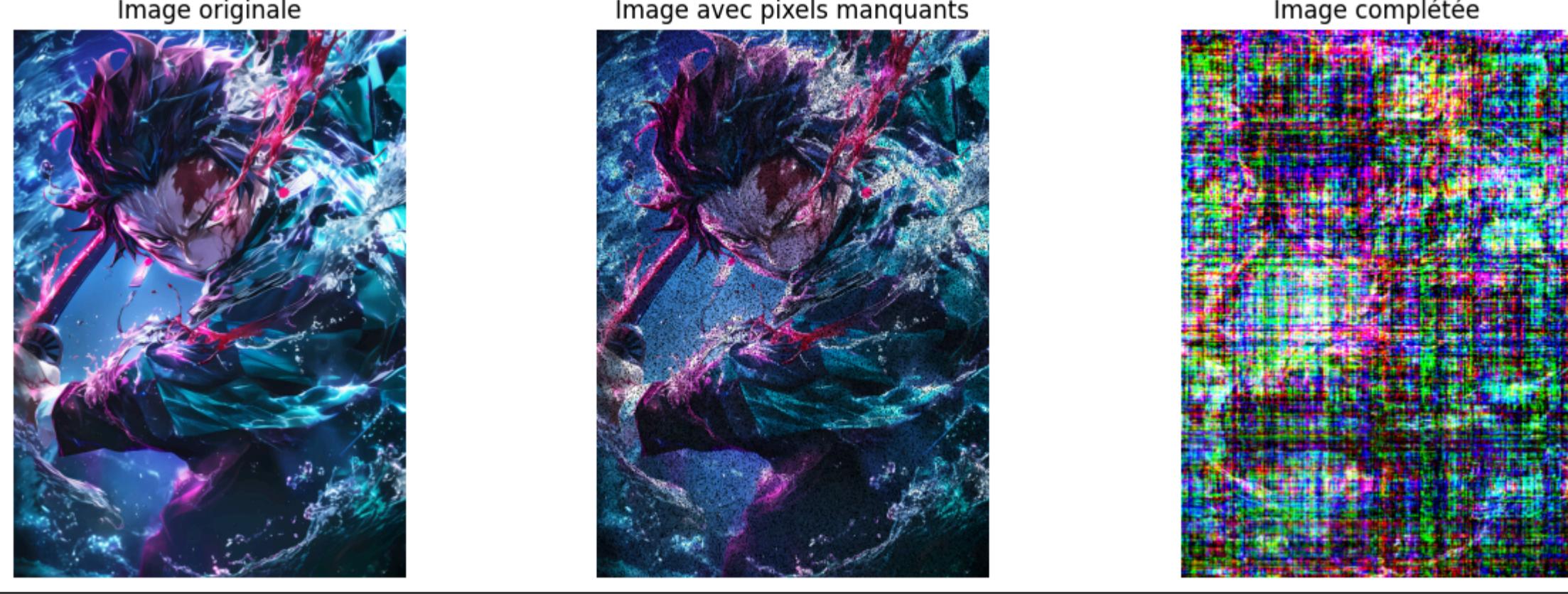
```

1 RMSE pour l'image en couleur
2 image_path = "/content/232856.jpg"
3 image = cv2.imread(image_path)
4
5 # Vérifier si l'image a été chargée correctement
6 if image is None:
7     print("Erreur : Impossible de charger l'image. Vérifiez le chemin du fichier.")
8 else:
9    # Normaliser l'image pour avoir des valeurs entre 0 et 1 et convertir en float32
10    image_normalized = (image / 255.0).astype(np.float32)
11
12    # Créer un masque aléatoire de pixels manquants (20% de pixels masqués ici)
13    mask = np.random.rand(*image_normalized.shape[:2]) > 0.2 # 80% d'observations, 20% masquées
14
15    # Appliquer le masque à chaque canal de couleur
16    masked_image = image_normalized.copy()
17    for i in range(3): # Pour chaque canal R, G, B
18        masked_image[:, :, i] *= mask
19
20    # Compléter les pixels manquants avec SVT pour chaque canal de couleur en utilisant 'randomized'
21    completed_image = image_normalized.copy()
22    for i in range(3):
23        completed_image[:, :, i] = svt_solve(masked_image[:, :, i], mask, algorithm='randomized')
24
25    # Calculer le RMSE pour chaque canal et le RMSE moyen
26    rmse_values = [calculate_rmse(image_normalized[:, :, i], completed_image[:, :, i], mask) for i in range(3)]
27    mean_rmse = np.mean(rmse_values)
28    print("RMSE pour chaque canal (R, G, B) : ", rmse_values)
29    print("RMSE moyen : ", mean_rmse)
30
31
32 # Afficher l'image originale, l'image avec pixels manquants et l'image complétée
33 plt.figure(figsize=(15, 5))
34 plt.subplot(1, 3, 1)
35 plt.imshow(cv2.cvtColor(image_normalized, cv2.COLOR_BGR2RGB))
36 plt.title("Image originale")
37 plt.axis("off")
38
39 plt.subplot(1, 3, 2)
40 plt.imshow(cv2.cvtColor(masked_image, cv2.COLOR_BGR2RGB))
41 plt.title("Image avec pixels manquants")
42 plt.axis("off")
43
44 plt.subplot(1, 3, 3)
45 plt.imshow(cv2.cvtColor(completed_image, cv2.COLOR_BGR2RGB))
46 plt.title("Image complétée")
47 plt.axis("off")
48
49 plt.show()
50

```

RMSE pour chaque canal (R, G, B) : [0.502821741821711, 0.45973992928360813, 0.4839066849646451]

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



```

1 # Charger l'image en couleur
2 image_path = "/content/232856.jpg"
3 image = cv2.imread(image_path)
4
5 # Vérifier si l'image a été chargée correctement
6 if image is None:
7     print("Erreur : Impossible de charger l'image. Vérifiez le chemin du fichier.")
8 else:
9     # Normaliser l'image pour avoir des valeurs entre 0 et 1 et convertir en float32
10    image_normalized = (image / 255.0).astype(np.float32)
11
12    # Créer un masque aléatoire de pixels manquants (20% de pixels masqués ici)
13    mask = np.random.rand(*image_normalized.shape[:2]) > 0.2 # 80% d'observations, 20% masquées
14
15    # Appliquer le même masque à chaque canal de couleur
16    masked_image = image_normalized.copy()
17    for i in range(3): # Pour chaque canal R, G, B
18        masked_image[..., i] *= mask
19
20    # Compléter les pixels manquants avec SVT pour chaque canal de couleur en utilisant 'randomized'
21    completed_image = image_normalized.copy()
22    for i in range(3):
23        completed_image[..., i] = svt_solve(masked_image[..., i], mask, tau=200, delta=1.0, max_iterations=1500, algorithm='randomized')
24
25    # Calculer le RMSE pour chaque canal et le RMSE moyen
26    rmse_values = [calculate_rmse(image_normalized[..., i], completed_image[..., i], mask) for i in range(3)]
27    mean_rmse = np.mean(rmse_values)
28    print("RMSE pour chaque canal (R, G, B) : ", rmse_values)
29    print("RMSE moyen : ", mean_rmse)
30
31
32
33 # Afficher l'image originale, l'image avec pixels manquants et l'image complétée
34 plt.figure(figsize=(15, 5))
35 plt.subplot(1, 3, 1)
36 plt.imshow(cv2.cvtColor(image_normalized, cv2.COLOR_BGR2RGB))
37 plt.title("Image originale")
38 plt.axis('off')
39
40 plt.subplot(1, 3, 2)
41 plt.imshow(cv2.cvtColor(masked_image, cv2.COLOR_BGR2RGB))
42 plt.title("Image avec pixels manquants")
43 plt.axis('off')
44
45 plt.subplot(1, 3, 3)
46 plt.imshow(cv2.cvtColor(completed_image, cv2.COLOR_BGR2RGB))
47 plt.title("Image complétée")
48 plt.axis('off')
49
50 plt.show()
51

```

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).  
RMSE pour chaque canal (R, G, B) : [0.1919805610088239, 0.18889837738892086, 0.19707728893059345]  
RMSE moyen : 0.192652075773461

