Master 1 IMAGINE

GASC Thibault
DIAB Ingo

Compte rendu 4 Projet : Détection de zones copiées-déplacées dans des images

Année Universitaire 2022-2023

Table des matières

I Travail de la semaine

1 Partie sans CNN

La semaine dernière nous sommes arrivé à détecter les points clés sur une image. Comme mentionné dans le précédent compte rendu, l'algorithme SIFT (que l'on a utilisé pour la détection) nous a permit d'avoir des descripteurs pour chaque point clés. C'est en comparant les descripteurs que l'on va détecter s'il y a des zones de copiées-déplacées.

Pour comparer les descripteurs, nous allons utiliser l'algorithme de clustering DBSCAN. Pour bien comprendre le fonctionnement de cet algorithme, j'ai trouvé un site qui expliquait bien le principe (lien disponible sur le git dans le dossier /Documents).

L'algorithme DBSCAN, Density-Based Spatial Clustering of Applications with Noise, permet de partitionner des données. Il prend en paramètre une distace appelée epsilon, noté *epsilon* et le nombre minimum de points, noté *min_sample*, devant se trouver à une distance epsilon pour que les points soient considérer comme un cluster. Ces paramètre sont modifiables selon les données à traiter. L'algorithme va classer chaque point selon 3 classes : **Core**, **Border** et **Noise**.

Un point est classé **Core** si dans un rayon epsilon, il y a au moins un nombre de points min_sample (y compris le point au centre du cercle). À contrario, si le nombre de points est inférieur à min_sample, le point en question est classé **Border**. Si le point est seul dans un rayon epsilon, alors il est classé **Noise**.

Voici un schéma représentatif avec un $min\_sample = 3$:

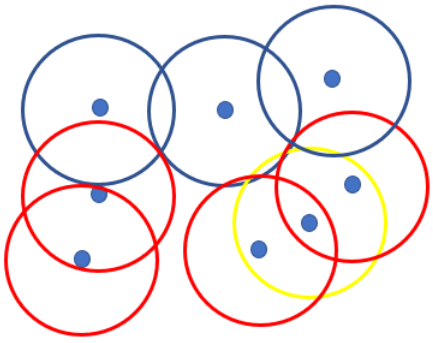

FIGURE 1 – Clusters par DBSCAN

Les points classés **Core** sont représenté en jaune, ceux classés **Border**

sont représentés en rouge et les **Noise** en bleu.

Cet algorithme donne des résultats plutôt intéressant pour la détection de zones copiées-déplacées. En voici un exemple :

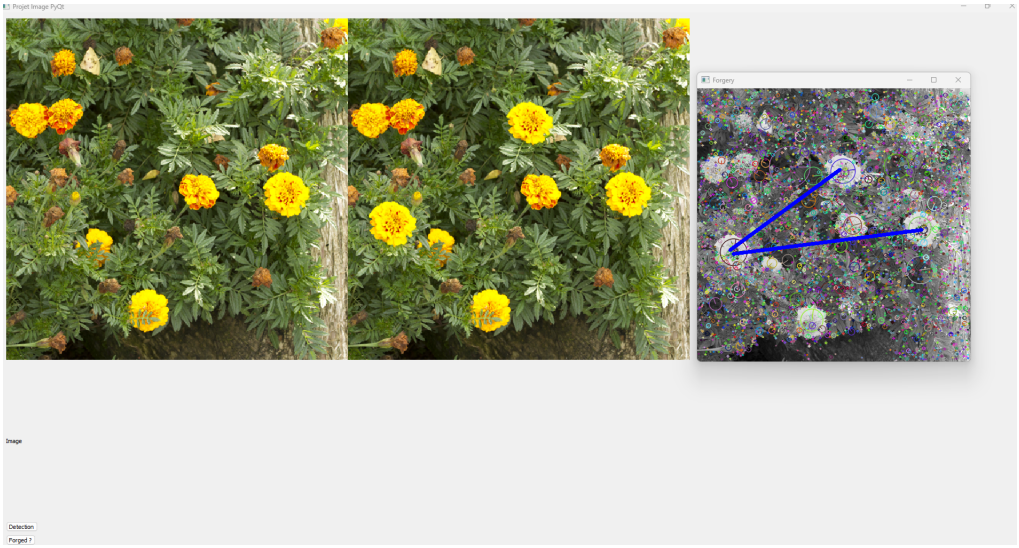

FIGURE 2 – résultat obtenu après DBSCAN

L'image originale est à gauche, l'image falsifiée au milieu et l'image résultat, après l'algorithme DBSCAN, est à droite. Les traits bleu sur l'image nous montre les mêmes zones qui "apparaissent" plusieurs fois dans l'image (zones copiées-déplacées).

2 Partie CNN

Concernant la partie CNN, nous avons pallié au problème de RAM en prenant un échantillon réduit du dataset que nous avons trouvé. Nous avons réduit le nombre d'images au maximum pour faire en sorte de ne pas surcharger la RAM. Après plusieurs essais, nous avons décider de prendre 500 images par classe (1 000 images en tout). Après avoir pré-traité les images comme expliqué lors du compte rendu 3, nous avons créer un modèle.

Ce modèle contenait 3 couches de convolutions 2D ainsi que 3 couches de MaxPooling (on alternait Conv2D et MaxPooling). Il y avait ensuite une couche Flatten pour transformer l'image en vecteur. Enfin nous avions deux couches intermédiaires de 1000 neurones ainsi qu'une couche de sortie de 1 neurone.

Malheureusement, l'entraînement n'était pas bon. L'accuracy sur le jeu de test et d'entraînement était d'environ 50% (même parfois moins). La loss sur le jeu de test et d'entraînement stagnait aux alentours de 0.6 et ne descendait que très rarement à 0.5. Nous avons donc modifié tous les paramètres possible afin d'améliorer notre modèle. Nous avons rajouté des couches de convolutions, des couches de neurones intermédiaires mais rien ne marchait. Nous avons donc essayé de diminuer/augmenter le learning rate de notre modèle mais ça n'a pas non plus marché. Nous nous sommes ensuite demandé si le dataset que nous avions trouvé était réellement efficace. Nous avons trouvé un autre dataset sur Internet, beaucoup moins conséquent et nous avons entraîner notre modèle dessus afin de voir, en sachant que le modèle ne serait pas très fiable, s'il y avait des changements au niveau de l'accuracy et de la loss. Mais, même avec ce dataset, cela n'a pas marché.

Nous avons donc cherché des documents pour voir d'où pourrais venir ce problème. Nous avons trouvé l'existence de modèles tel que VGG16, VGG19 et InceptionV3. Ces modèles peuvent être chargés à partir de Keras. Lors du chargement du modèle, nous pouvons décider si nous voulons utiliser le modèle en entier (Features extractor + Neural Network) ou si nous voulons seulement nous intéresser au Features extractor. Pour un premier test, nous avons donc choisi d'utiliser le Features extractor d'InceptionV3, nous avons ensuite ajouter 2 couches de 4096 neurones intermédiaires chacune. La classification que nous voulons étant binaire (0 = Original, 1 = Forged) alors nous avons ajouté une couche output contenant 1 seul neurone et une fonction d'activation sigmoid.

Pour finir nous avons compiler le modèle avec l'optimizer adam (learning rate de base à 0.001) et une fonction de loss binary_crossentropy (classification binaire). Après un peu plus de 2h d'entraînement, l'utilisation du features extractor d'InceptionV3 nous permet, avec 5 folds, 10 epochs/fold et 500 images par classe, d'avoir une accuracy de 74.5%.

FIGURE 3 – Dernière epoch (10) du dernier fold (5)

Nous pouvons observer que l'accuracy sur le training set est de 81%, ce qui est à une distance raisonnable de 74.5% et montre donc qu'il n'y a pas de sur-apprentissage.

Enfin nous avons tester le modèle sur quelques images. Les images appartiennent au dataset d'entraînement donc le test ne sera pas concluant mais c'est pour vérifier que le modèle ne nous renvoi pas de résultats absurde.

```
[ ]  _imageOriginal = _loadedOriginal[50].reshape(-1, IMG_SIZE, IMG_SIZE, 3);
     _imageOriginal=_imageOriginal.astype('float')
     _imageOriginal=_imageOriginal/255.0
     prediction = loaded_model.predict(_imageOriginal)
     print(prediction)
     _imageForgery = _loadedForgery[30].reshape(-1, IMG_SIZE, IMG_SIZE, 3);
     _imageForgery=_imageForgery.astype('float')
     _imageForgery=_imageForgery/255.0
     prediction = loaded_model.predict(_imageForgery)
     print(prediction)

     1/1 [==============================] - 0s 266ms/step
     [[0.00025315]]
     1/1 [==============================] - 0s 178ms/step
     [[0.9995689]]
```

FIGURE 4 – Prédiction sur 2 images

Nous pouvons voir que, pour l'image originale, le modèle prédit qu'il y ait 0.025% de chance que l'image soit falsifiée (donc le modèle nous dit que l'image est l'original) et, pour l'image falsifiée, le modèle prédit qu'il y ait 99.96% de chance que l'image soit falsifiée (donc le modèle nous dit que l'image est falsifiée). Les résultats de la prédiction sont assez conséquent car il s'agit d'images que le modèle connaît.

Pour les prochaines séances, nous comptons tester ce modèle sur des images que le modèle ne connaît pas. Ensuite, selon les résultats, corriger ou améliorer le modèle pour qu'il possède une plus grande accuracy. Pour l'instant nous nous sommes interessé seulement à l'accuracy mais nous comptons aussi prendre en compte d'autre mesure tel que le Recall, le F1-Score,... afin d'avoir un meilleur modèle.