Note méthodologique : RandAugment

Article : arXiv :1909.13719v2 [cs.CV] 14 Novembre 2019
Par : Ekin D. Cubuk ∗, Barret Zoph∗, Jonathon Shlens, Quoc V. Le
Google Research, Brain Team

Dataset retenu

Le Dataset utilisé pour cette démonstration est celui des images de biens de consommation.
Le Dataset comprend 2 colonnes :

- Main_category : comprend la catégorie de l'image à classifier => Target
- Image_path : Le chemin pour charger l'image à traiter

Ces produits sont regroupés en 7 catégories :

- Home Furnishing, Baby Care
- Watches
- Home Decor & Festive Needs
- Kitchen & Dining
- Beauty and Personal Care
- Computers

Il y a en tout 1050 images réparties de manières égales entre chaque catégorie, soit 150 images par catégories.

Exemples d'image :

Les concepts de l’algorithme récent

L’algorithme de RandAugment va procéder à une Data Augmentation aléatoire des images rentrées dans le modèle. Pour cela, il y a plusieurs paramètres à prendre en compte.

Les principaux paramètres de RandAugment sont :

- n : Il s’agit du nombre de transformations à appliquer à chaque image. Plus la valeur de n est grande, plus le nombre de transformations aléatoires appliquées à chaque image est élevée.
- m : Il s’agit de la magnitude de ces changements. Il permet de déterminer à quel point les transformations seront appliquées. Une valeur de m élevée va appliquer de fortes transformations à l’image.
- Value_range : Il s’agit de la plage de valeurs que prennent les pixels avant la transformation pour s’assurer que les transformations vont s’appliquer correctement.

Il y a 15 transformations possibles par RandAugment :

1. Transformation de l'échelle : Zoom in ou zoom out de l'image.
2. Translation : Déplacement de l'image horizontalement et/ou verticalement.
3. Rotation : Rotation de l'image autour de son centre.
4. Symétrie horizontale/verticale : Retournement horizontal ou vertical de l'image.
5. Distorsion élastique : Application de déformations élastiques à l'image.
6. Contraste : Ajustement du contraste de l'image.
7. Luminosité : Ajustement de la luminosité de l'image.
8. Saturation : Modification de la saturation des couleurs de l'image.
9. Teinte : Modification de la teinte des couleurs de l'image.
10. Éclairement : Application d'effets d'éclairage ou d'ombres à l'image.
11. Flou : Ajout d'un flou à l'image.
12. Coupe aléatoire : Sélection d'une partie aléatoire de l'image.
13. Ajout de bruit : Intégration de bruit aléatoire dans l'image.
14. Filtrage : Application de filtres (par exemple, filtres de convolution) à l'image.
15. Déformation : Application de déformations géométriques à l'image.

La modélisation

Le modèle accepte en entrée des données au format (224, 224, 3). Les images sont donc chargées sous ces dimensions, puis transformées en Array Numpy. On ajoute toutes les images traitées dans une liste que l'on transforme ensuite en tableau numpy.

Après je récupère les labels transformés par un LabelEncoder() donc de 0 à 6.

Ensuite, j'utilise la fonction train_test_split() pour séparer les données en 3 jeux de données distincts avec l'option stratify=y. Cette option permet de garantir que chaque classe soit représenté le plus équitablement possible dans chacun des jeux de données.

J'ai donc un jeu d'entrainement avec 551 images transformées. Un jeu de validation avec 184 images transformées, et un jeu de test de 315 images transformées.

Ces données vont rentrer dans la première couche du modèle qui est donc celle de RandAugment. Cette couche va donc appliquer sur chaque image 3 transformations parmi une quinzaine disponible. Le choix est fait au hasard. Les transformations seront de magnitude 0.1. Les mêmes magnitudes que j'avais utilisé pour l'entraînement du model avec la Data Augmentation classique. La Data augmentation faite précédemment faisait 3 images à partir d'une seule comme le RandAugment, mais avec ces trois transformations uniquement : Renversement horizontal, rotation et zoom.

Ensuite, les données passent dans une couche de rescaling, qui va normaliser la valeur des pixels en la divisant par 127.5 et un décalage de -1.0 également pour avoir des valeurs de pixels compris entre -1.0 et 1.0. Ce qui facilite le traitement ensuite par notre modèle VGG16.

Le modèle choisi est de type VGG16 avec les poids préentraînés de l'ensemble de données '' imagenet ''. La couche de prédiction a été enlevée pour correspondre à notre problème de classification. J'ai rajouté 4 couches supplémentaires à entrainer.

Ces 4 couches sont les suivantes :

- GlobalAveragePooling2D() : couche qui permet de réduire la dimension spatiale des données
- Dense(256, activation='relu') : Couche entièrement connectée avec 256 neurones, l'option 'relu' permet d'ajouter une non-linéarité au modèle
- Dropout(0.5) : désactivation de 50% des neurones pour éviter un overfitting du modèle
- Dense(7, activation='softmax') : couche de prédiction pour nos 7 catégories. L'option 'softmax' permet de normaliser les prédictions.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rand_augment (RandAugment)  (None, 224, 224, 3)       0

 rescaling (Rescaling)       (None, 224, 224, 3)       0

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 256)               131328

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 7)                 1799

=================================================================
Total params: 14847815 (56.64 MB)
Trainable params: 133127 (520.03 KB)
Non-trainable params: 14714688 (56.13 MB)
```

Une synthèse des résultats

Pour comparer la méthode, j'ai fait un comparatif avec un autre paramétrage du même modèle. L'autre modèle aura une Data Augmentation : application de 3 transformations par images, un renversement horizontal, une rotation de 10% et un zoom de 10% également.

La métrique de performance du modèle est la 'val_loss'. Cette métrique 'loss' va quantifier la différence entre les prédictions du modèle et les vrais labels. Plus la valeur est faible, meilleur sera le modèle. L'ajout de 'val', dit que la métrique va se baser sur les valeurs trouvées sur le jeu de validation. Pour les autres scores utilisés, je me suis également basé sur l'accuracy et le temps d'entrainement. C'est-à-dire la proportion entre les mauvaises prédictions et les bonnes prédictions. Plus le score est proche de 1, plus le modèle est performant. J'ai également récupéré le temps d'entrainement.

Voici les résultats :

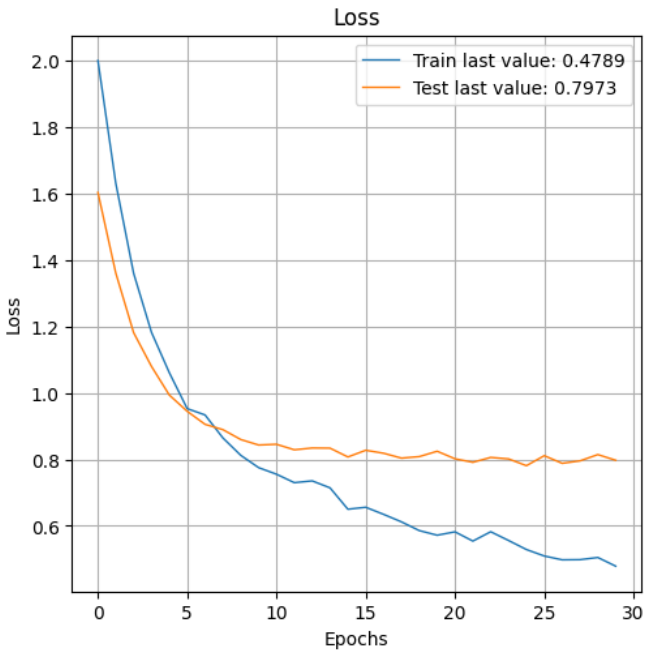
Scores :

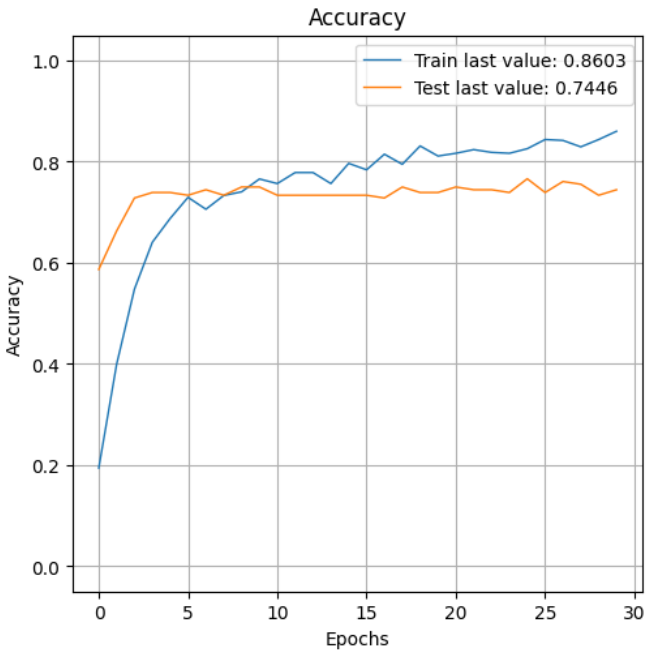
Score \ Modèle	Data Augmentation	RandAugment
Best Training Accuracy	0.8893	0.8693
Best Validation Accuracy	0.7446	0.7391
Best Test Accuracy	0.7619	0.7683
Temps d'entrainement	23m 11s	17m 30s

On peut voir plusieurs améliorations, notamment au niveau du score sur le jeu de Test. Ce jeu correspond aux données que le modèle ne connaît pas, donc le résultat le plus important.
On voit également une réduction significative du temps d'entraînement.

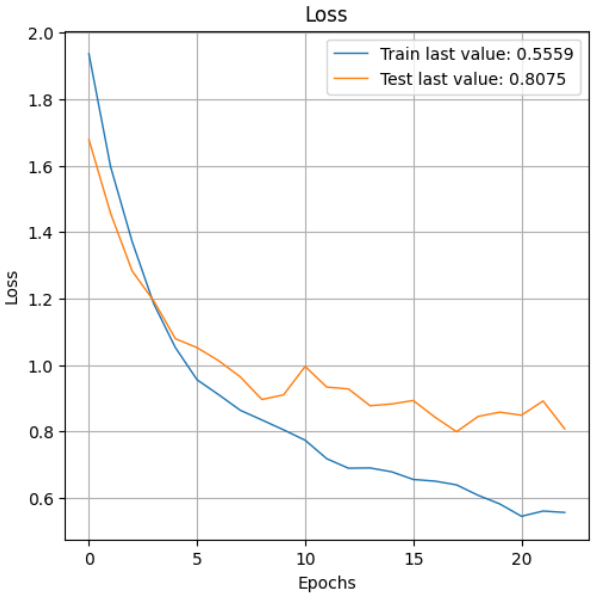
Courbes d'entrainement :

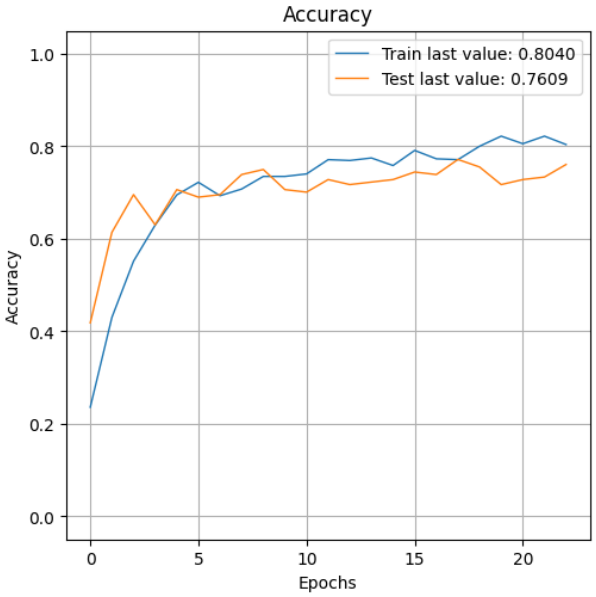
Modèle avec Data Augmentation





Modèle avec RandAugment :



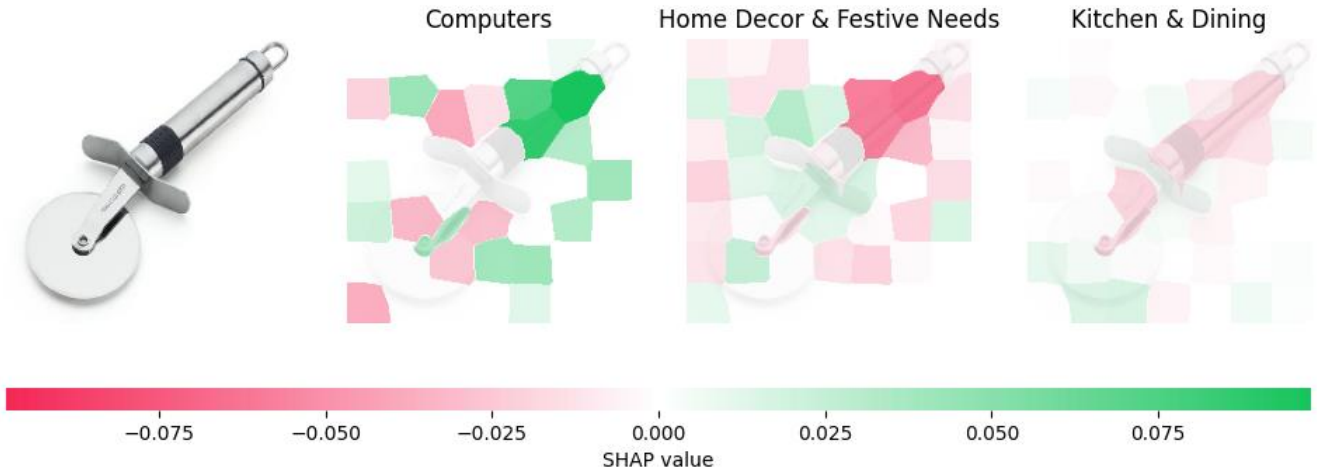


La courbe de loss, on peut noter une augmentation des scores, donc une perte de performance, mais on peut aussi noter que les scores sont plus proches, donc on limite l'overfitting sur les données.

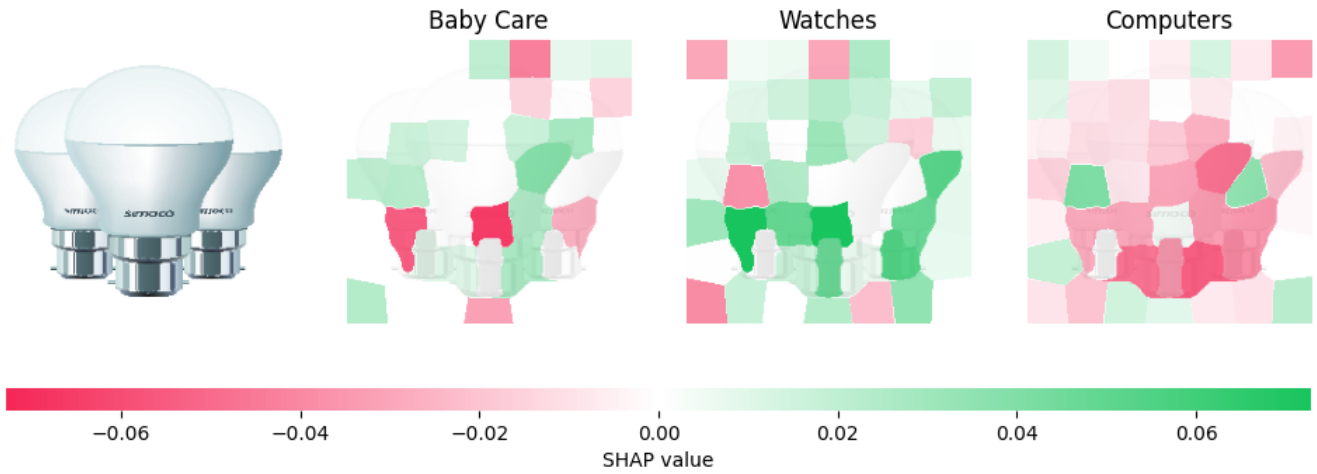
De même la baisse du score d'accuracy pour les jeux d'entraînement et de validation, on peut noter une plus faible variation, ce qui démontre, qu'il y a moins d'overfitting sur les données.

L’analyse de la feature importance globale et locale du nouveau modèle

Pour la feature importance, j’ai choisi d’utiliser la librairie SHAP. J’ai également choisi un échantillon d’image et j’ai décidé d’utiliser des masques de pixels, pour repérer des zones, plutôt que des pixels. Je vais représenter ces features sur des graphiques. Donc mon algorithme va mettre en valeur des zones de l’image. Chaque image aura 4 représentations, la première est l’image de base, et ensuite, les 3 suivantes vont représenter les valeurs des zones par classe, avec des couleurs différentes. La couleur verte sera pour les valeurs positives, la couleur rouge pour les valeurs négatives. Une zone en vert servira donc à prédire la classe représentée.



Ici on voit la classe prédite la première est computers, et que le manche est la zone qui a contribué le plus à cette prédiction. On peut voir également que la même zone a contribué négativement lors de la prédiction pour la classe Home Decor & Festive Needs. De même dans une moindre mesure pour la classe Kitchen Dining.



Pour cet exemple, les bases des ampoules ont contribué négativement à la prédiction pour la classe Baby Care. Mais la forme globale a contribué positivement. Contrairement, les bases ont contribué positivement pour la prédiction de la classe Watches.

Les limites et les améliorations possibles

Les limites sont que RandAugment applique des transformations de manière aléatoire, ce qui peut entraîner des augmentations non souhaitées ou inutiles. Un manque de contrôle fin sur le choix des transformations peut affecter la qualité des données générées.

Rand Augment est également sensible aux choix des hyperparamètres, notamment ‘n’ et ‘m’. Un choix inapproprié peut diminuer les performances du modèle.

RandAugment est pour l’instant limité à certains domaines de machine learning. Les principaux domaines dans lesquels il reste à étudier sont la segmentation des images, la perception 3D, la reconnaissance vocale et audio.

Pour les améliorations, il peut être intéressant de personnaliser la liste des transformations en fonction des tâches ou du domaine.

Il faut garder à l’esprit que dans certains cas, le fait de garder les anciennes méthodes de Data Augmentation permet d’obtenir de meilleurs résultats.