Université de Strasbourg

Laboratoires iCube

Génération de cartes de saillances d'images 2D et 3D

Travail d'Etude et de Recherche

Ayoub ADIL

Encadrement : Antonio Capobianco, Flavien Lecuyer

14 Mai 2022

Table des matières

1 Remerciements

En premier lieu, je tiens à remercier mes encadrants M. Antonio CAPOBIANCO et M. Flavien LECUYER. Je leur remercie pour leurs disponibilités et la qualité de leur encadrement. Je tiens aussi à remercier tous mes professeurs à l'université de Strasbourg pour leurs efforts pendant toute l'année universitaire et toute personne ayant contribuer à réaliser ce travail.

2 Résumé

Cet article présente un cadre pour prédire l'attention visuelle d'images omnidirectionnelles. La configuration clé de notre architecture est la prédiction simultanée de la carte de saillance et d'un chemin de balayage correspondant pour un stimulus donné. Le cadre implémente un réseau neuronal convolutif complété par un module d'attention pour générer des cartes de saillance.

3 Introduction

Les techniques de Deep Learning, et particulièrement les techniques de réseaux de neurones convolutifs permettent d'obtenir de très bons résultats. Plusieurs librairies – pour la plupart en Python - proposent d'ailleurs des outils permettant de générer ces cartes de saillance. La prédiction des phénomènes d'attention visuelle à partir de tout type de média est d'une utilité précieuse pour les créateurs de contenu ainsi que l'évaluation de l'expérience utilisateur dans des environnements 2D et 3D. Pour cela, on utilise le plus souvent des cartes de saillance. Les approches diffèrent cependant selon que l'on considère des images 2D et des images 3D, les données de profondeur exerçant une influence significative sur les effets de saillance. Les applications de réalité virtuelle (VR) offrent une haute qualité d'immersion pour l'expérience des utilisateurs. La plupart des applications VR se présentent sous la forme de la vidéo 360, alors que les images sont représentées sous un nouveau format de contenu multimédia appelé image omnidirectionnelle. Ces images couvrent tout l'espace de visualisation sphérique, où l'utilisateur a la liberté d'assister à n'importe quelle direction juste en pointant sa tête dans n'importe quelle direction [1]. Le rendu de la fenêtre d'affichage 360 des images est pris en charge par de nombreux types de transformations de mappage de coordonnées sphère à plan, la projection équirectangulaire (ERP) est l'un des formats les plus largement utilisés de projection de mappage de qualité uniforme. Il projette le contenu sphérique sur un seul plan 2D haute résolution, où les coordonnées longitudinales et latitudinales de la sphère sont représentées respectivement sur les axes ERP horizontal et vertical.

Contrairement à la diffusion traditionnelle par fenêtre fixe de contenu 2D, l'expérience immersive est délivrée à l'aide de technologies récentes tel que Head Mounted Display (HMD). Ils sont habilités par la capacité d'étudier l'espace sphérique, leur permettant d'avoir la meilleure expérience immersive réaliste avec une consommation élevée de ressources. Par conséquent, la capacité de prédire la fenêtre fréquentée qui correspond aux orientations des mouvements de tête en amont, permet d'optimiser le processus de livraison et de fournir

4 Contexte du problème

Le travail sert à enseigner l'ergonomie des interfaces 2D et 3D pour des fins pédagogiques. L'objectif est de fournir un outil permettant de générer des cartes de saillance d'éléments d'interface 2D et 3D (capture d'écran d'applications, sites web statiques ou dynamiques, interfaces d'environnements de réalité virtuelle).

5 Travaux réalisés

5.1 Cartes de saillance

La gestion de la saillance de certains éléments est importante pour l'élaboration d'une cartographie où les éléments principaux doivent ressortir de manière claire. L'analyse automatique de la saillance visuelle est une branche de la vision par ordinateur qui tente de reconstruire le processus humain de saillance par des méthodes informatiques.

Pour faire, il faut construire une enveloppe (que nous allons l'appeler par la suite un wrapper) autour de notre modèle. Pour simplifier l'explication, on peut imaginer le wrapper qui parcourt, avec l'image, le réseau de neurones, et à chaque itération note les endroits les plus influents où un vecteur de classe a été activé. Ainsi en sortant du réseau le wrapper a une carte de saillance qu'on en peut la superposer avec l'image d'origine et ainsi simuler la saillance visuelle.

Les cartes d'activation de classe (CAM) [2] sont des techniques puissantes utilisées dans la vision par ordinateur pour les tâches de classification. Elles permettent d'inspecter l'image à catégoriser et de comprendre quelles parties/pixels de cette image ont le plus contribué à la sortie finale du modèle.

En gros, imaginons que nous construisions un CNN dans le but de classer les photos de personnes en "Chien" et "Chat", puis que nous lui fournissions une nouvelle photo et qu'il renvoie l'étiquette "Chien". Avec l'outil CAM, nous serions en mesure de voir quelles parties de la photo activent le plus la classe "Homme". Cela peut être très utile si nous voulons améliorer la précision de notre modèle et si nous devons comprendre quelles couches doivent être modifiées, ou si nous devons pré-traiter différemment les images de l'ensemble d'entraînement.

CAM, Grad-CAM, Grad-CAM++, Smooth Grad-CAM++, Score-CAM, SS-CAM, IS-CAM, XGrad-CAM et **Layer-CAM** , sont tous des cartes d'activation de classe très importantes dans le cadre de notre recherche. Elles répondent parfaitement à nos besoin. Chacune de ces CAMs utilisent un modèle en entrée, ce modèle doit bien évidemment être entraîné. Dans un premier temps, nous allons utilisé un modèle pré-entraîné pour tester les performances de chacune des CAMs.

Pendant la recherche nous avons trouvé une bibliothèque (TorchCAM) qui utilise les mécanismes d'accrochage de PyTorch pour récupérer de manière transparente toutes les informations nécessaires pour produire l'activation de la classe [4] sans efforts supplémentaires de la part de l'utilisateur.

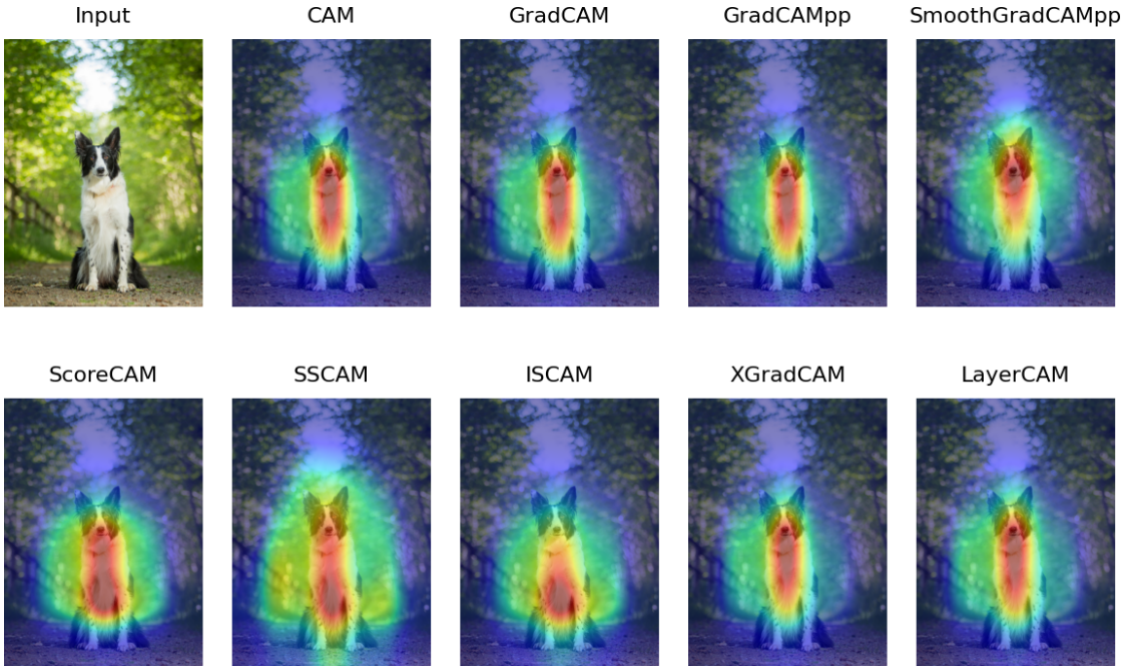

FIGURE 1 – Cartes d'activation créées à l'aide d'un Resnet-18 pré-traité.

5.2 Implémentions des CAMs

Nous importons la méthode voulue de **TorchCam**, et on lui passe en paramètre notre modèle. L'utilisation de la carte d'activation se fait comme suit :

— On lit une image.
— On fait les pré-traitements nécessaires sur l'image.
— On passe notre modèle entraîné au CAM en paramètre .
— On applique la CAM sur l'image.
— On affiche le résultat en superposant l'image originale et son RAW CAM.

```python
def MethodCAM(imgDir):
    img = read_image(imgDir)
    input_tensor = preProcess(img)
    model = resnet18(pretrained=True).eval()
    cam_extractor = CAM(model, 'layer4', 'fc')
    #with torch.no_grad():
    out = model(input_tensor.unsqueeze(0))
    activation_map = cam_extractor(out.squeeze(0).argmax().item(), out)
    result = overlay_mask(to_pil_image(img), to_pil_image(activation_map[0].squeeze(0), mode='F'), alpha=0.5)
    return result
```

FIGURE 2 – Méthode en python : : Application de CAM.

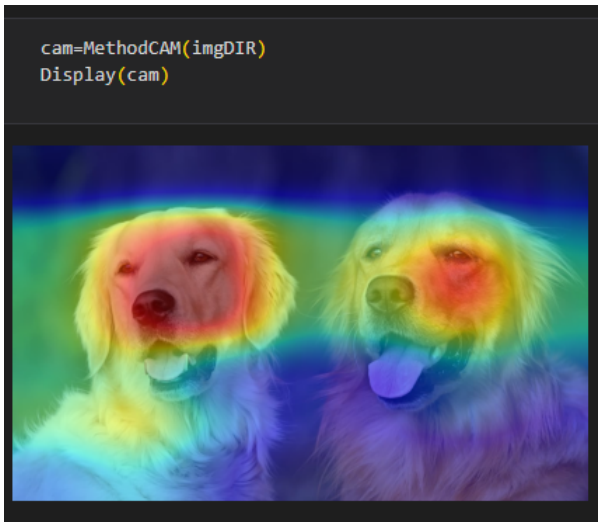

FIGURE 3 – Affichage du résultat : : Application de CAM.

De la même manière, avec les autres CAMs et sur la même image, on obtient les résultats si-dessous. On remarque que toutes les méthodes donnent des résultats très acceptables si les modèles sont **bien entraînés**.

NB : Ces méthodes ne diffèrent pas beaucoup en temps de calcul (entre 1 et 2 sec dans nos machines).

Algo	temps d'execution en moyenne
CAM	environ 1.1s
ScoreCAM	non implementé
SSCAM	non implementé
ISCAM	non implementé
GradCAM	environ 1.2s
GradCAMpp	environ 1.4s
SmoothGradCAMpp	environ 1.9s
XGradCAM	environ 1.3s
LayerCAM	environ 1.3s

FIGURE 4 – temps de calcul des différentes méthodes TorchCam.

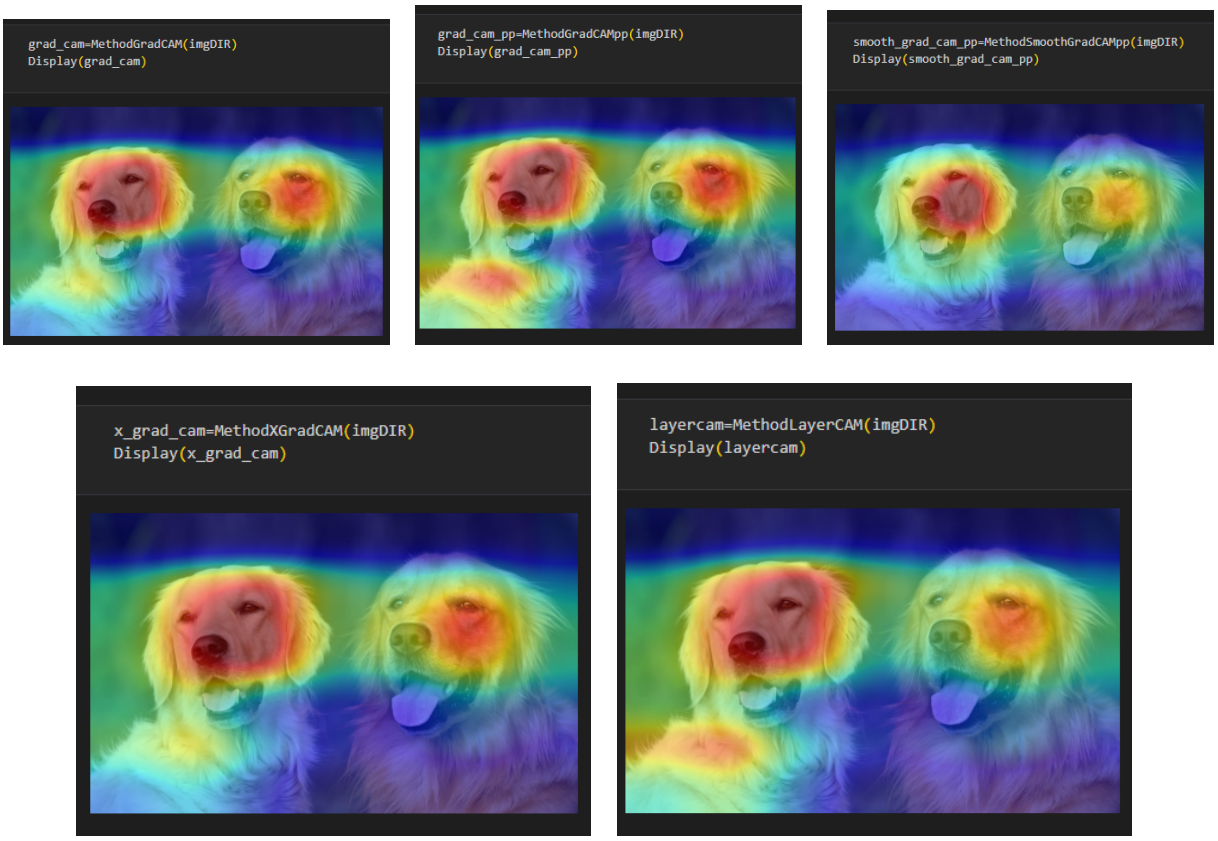

FIGURE 5 – Affichage des résultats : : Grad-CAM, Grad-CAM++, Smooth Grad-CAM++, XGrad-CAM et Layer-CAM

5.3 Modèle CNN

Dans la partie qui précède, nous avons supposé avoir un modèle CNN entraîné avec lequel nous avons construit les cartes de saillance pour simuler l'attention visuelle dans les images. Maintenant intéressons nous au modèle CNN.

Nous nous concentrerons sur l'un des algorithmes les plus puissants du Deep Learning, le réseau de neurones convolutifs ou CNN [3], qui sont de puissants modèles de programmation permettant la reconnaissance d'images en attribuant automatiquement à chaque image fournie en entrée, une étiquette correspondant à sa classe. Dans le cadre de ce projet, les étiquettes n'auront pas une importance pour l'utilisateur.

Nous avons coder un un modèle de réseau de neurones convolutif, qui prend une base de données étiquetée, nous l'avons entraîné, testé et validé. En dessous le code python qui lit les fichiers dans notre base de données locale, et crée les étiquettes en se basant sur les noms des dossiers dans lesquels se trouvent les images, ainsi que notre modèle.

```
X_TRAIN = []
Y_TRAIN = []

IMG_SIZE = 100
for category in LABELS:
    path = os.path.join(DATADIR, category)
    class_num = LABELS.index(category)
    for img in os.listdir(path):
        try:
            img_array = cv.imread(os.path.join(path, img))
            new_array = cv.resize(img_array, (IMG_SIZE, IMG_SIZE))
            X_TRAIN.append(new_array)
            Y_TRAIN.append(class_num)
        except Exception as e:
            pass

X_TRAIN = np.array(X_TRAIN).reshape(-1, IMG_SIZE, IMG_SIZE,3)
Y_TRAIN = np.array(Y_TRAIN)
print('-----------------------------------------------------')
print(X_TRAIN.shape)
print('-----------------------------------------------------')
X_TRAIN = X_TRAIN/255
```

FIGURE 6 – Lecture automatique des fichiers de la BDD.

```
model = Sequential()
model.add(Conv2D(64, (5,5), input_shape = X_TRAIN.shape[1:]))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (5,5)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (5,5)))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(5))
model.add(Activation("softmax"))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(X_TRAIN,Y_TRAIN, epochs=10, validation_split=0.1)

model.summary()
```

FIGURE 7 – couches du Réseau de Neurones Convolutifs.

Dans le cadre de notre projet (en 2D), il faut simuler l'attention visuelle sur les pages web statiques. Nous avons pensé que la base de données doit contenir des images screenshots des pages web statique ; contenant des boutons, des formulaires, etc.. ainsi nous pourrons faire la classification.

Et pour bien entraîner le modèle, il faut avoir une très grande base de données des images, et malheureusement ça prend énormément de temps pour la créer.

5.4 Comportement du Modèle

Une image testée par notre modèle, passe par une série de suite de convolution et pooling, et à chaque itération, une région/pixels contribue plus ou moins pour activer la classe.

on peut visualiser ce comportement dans la figure ci-dessous :

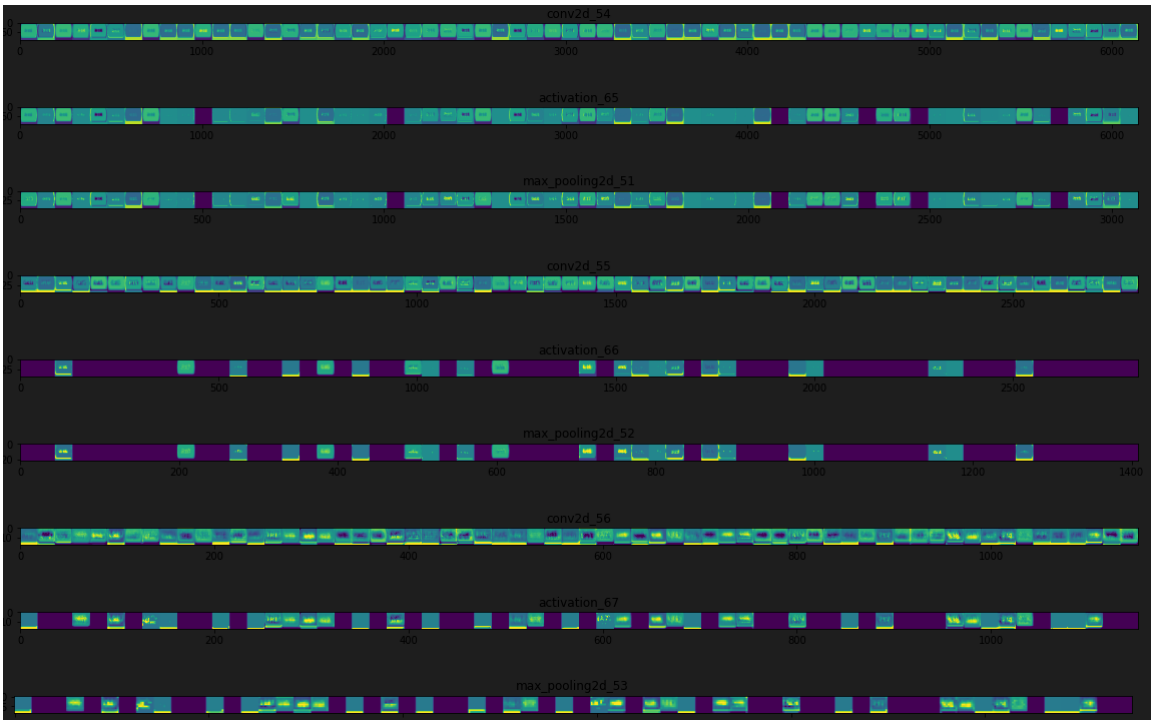

FIGURE 8 – Visualisation du comportement du CNN sur une image.

5.5 Interface

Nous avons penser également à introduire une interface utilisateur rudimentaire pour visualiser l'attention visuelle. L'interface utilise les mêmes principes vus dans les sections d'avant ; un modèle entraîné, et on propose de choisir la CAM que l'utilisateur veut utiliser.

FIGURE 9 – Interface graphique rudimentaire : les méthodes CAMs.

Résultats On observe que les résultats ne sont pas très pertinent pour le moment parce qu'on utilise un modèle qui a été entraîné sur une petite base de données.

NB : les méthodes ScoreCAM, SSCAM et ISCAM ne sont pas implémenté pour le moment.

6 Conclusions

Pour simuler l'attention visuelle, nous avons implémenté un modèle CNN, que nous passons sur un carte d'activation de classes. Le modèle doit être entraîné avec une bonne base de donnée.

Le travail est à améliorer, malheureusement nous n'avons pas trouver (et c'était dure de créer) une base de donné efficace pour ce projet. Une fois la BDD trouvée, nous pourrions travailler à passer à notre modèle des images 360 (image omnidirectionnelle) pour générer des cartes de saillance pour des images 3D.

Au niveau personnel, ce travail m'a permit d'améliorer mon esprit de recherche et de critique.

7 Bibliography

[**1**] Mohamed Amine KERKOURI, Marouane TLIBA, Aladine CHETOUANI, Mohamed SAYEH : Saly-Path360 : Saliency and Scanpath Prediction Framework for Omnidirectional Images.

[**2**] Valentina Alto : Class Activation Maps in Deep Learning

[**3**] Vidhuran : Convolutional Neural Networks (CNN)

[**4**] F-G Fernandez : TorchCam