RISE

```
import numpy as np
import cv2
import tensorflow as tf
import matplotlib.pyplot as plt

model_builder = tf.keras.applications.resnet_v2.ResNet50V2
preprocess_input = tf.keras.applications.resnet_v2.preprocess_input
decode_predictions =
tf.keras.applications.resnet_v2.decode_predictions
model = model_builder(weights="imagenet",
classifier_activation="softmax")

def generate_masks_randomly(m, input_size, p, resolution):
    low_res_size = (input_size[0] // resolution, input_size[1] //
resolution)
    masks = np.random.binomial(1, p, size=(m,
*low_res_size)).astype(np.float32)

    upscaled_masks = np.array([cv2.resize(mask, (input_size[1],
input_size[0]), interpolation=cv2.INTER_LINEAR) for mask in masks])

    upscaled_masks = upscaled_masks[..., np.newaxis]

    return upscaled_masks

def perturbed_image(masks, image):
    repeated_image = np.repeat(image[np.newaxis, ...], masks.shape[0],
axis=0)
    m_images = masks * repeated_image
    return m_images

def local_explanation(masks, mImages, classImage, model):
    size_image = mImages.shape[1:3]
    saliency_map = np.zeros(size_image, dtype=np.float32)

    # Stack images for batch processing
    preds = model.predict(mImages)

    # Get predicted indices and top labels
    top_labels = decode_predictions(preds, top=3)

    # Extract scores for the target class using a vectorized approach
    scores = np.array([next((label[2] for label in top_labels[i] if
label[1] == classImage), 0) for i in range(len(top_labels))])

    # Calculate saliency map using vectorized operations
```

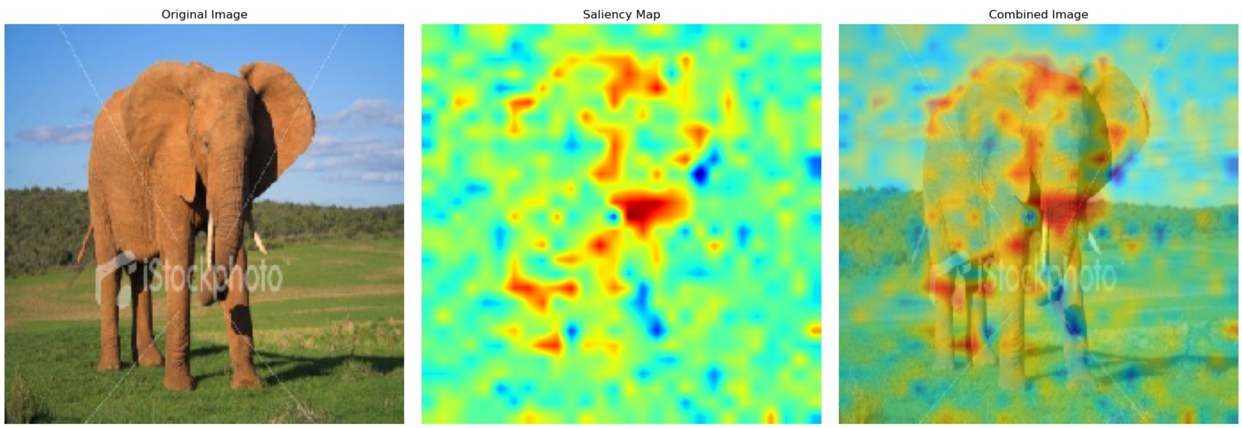
```
    saliency_map = np.sum(masks * scores[:, np.newaxis, np.newaxis,
np.newaxis], axis=0)

    return saliency_map /len(masks)

def rise_saliency(image, model, target_class, m=1000, p=0.5,
resolution=8):
    masks = generate_masks_randomly(m, image.shape, p, resolution)
    perturbed_images = perturbed_image(masks, image)
    saliency_map = local_explanation(masks, perturbed_images,
target_class, model)

    return saliency_map
```

```
img_path = "./data/African_elephant/ILSVRC2012_val_00048781.JPEG"
image = tf.keras.preprocessing.image.load_img(img_path,
target_size=(224, 224))
image = tf.keras.preprocessing.image.img_to_array(image)
image = tf.keras.applications.xception.preprocess_input(image)

target_class = 'African_elephant'
saliency_map = rise_saliency(image, model, target_class, m=7000)
```

```
219/219 ━━━━━━━━━━━━━━━━━━━━ 390s 2s/step
```

```
originalImage = cv2.resize(image/ 2 + 0.5, (224, 224))

saliency_map_normalized = (saliency_map - np.min(saliency_map)) /
(np.max(saliency_map) - np.min(saliency_map))

alpha = 0.5
saliency_map_color = cv2.applyColorMap((saliency_map_normalized *
255).astype(np.uint8), cv2.COLORMAP_JET)
saliency_map_color = cv2.cvtColor(saliency_map_color,
cv2.COLOR_BGR2RGB) / 255.0

combined_image = (1 - alpha) * originalImage + alpha *
saliency_map_color

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 6))

# Affichage de l'image originale
ax1.imshow(originalImage)
ax1.set_title("Original Image")
ax1.axis("off")

# Affichage de la carte de saillance
ax2.imshow(saliency_map_normalized, cmap='jet')
ax2.set_title("Saliency Map")
ax2.axis("off")
```

```
# Affichage de la combinaison de l'image originale et de la carte de saillance
ax3.imshow(combined_image, cmap='jet')
ax3.set_title("Combined Image")
ax3.axis("off")

plt.tight_layout()
plt.show()
```



Nous pouvons observer que l'augmentation du nombre de masques entraîne une réduction des zones rouges incohérentes dans la carte de saillance. Cela peut s'expliquer par la loi des grands nombres,les fluctuations aléatoires tendent à se lisser, ce qui permet d'obtenir une carte de saillance plus stable et fiable.

De plus, il est intéressant de noter que les zones qui ressortent dans la carte de saillance correspondent principalement aux contours et aux détails significatifs de l'image. Cela m que le modèle RISE réussit à identifier des caractéristiques essentielles.