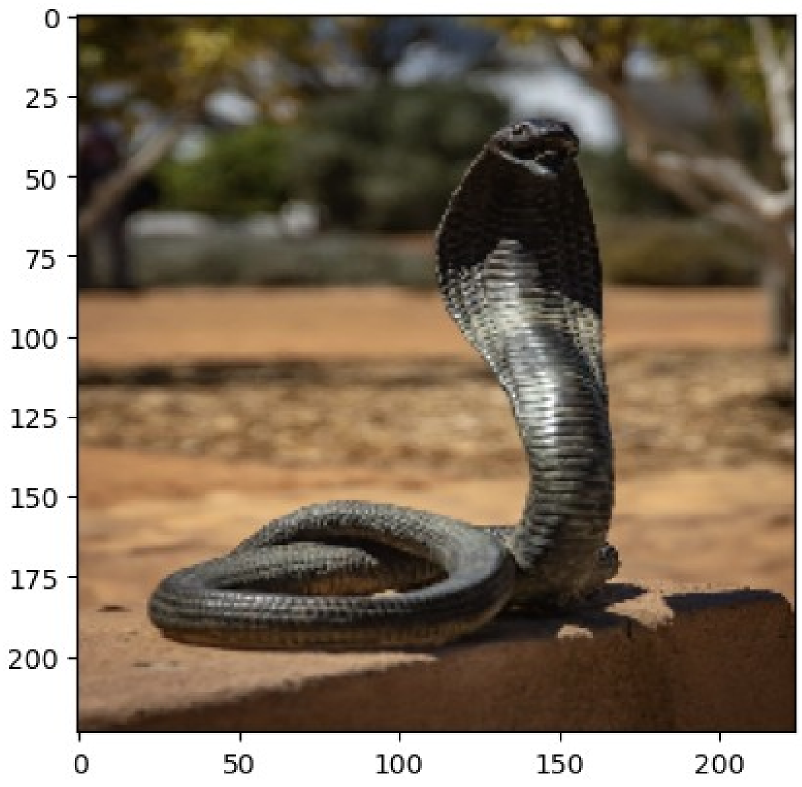
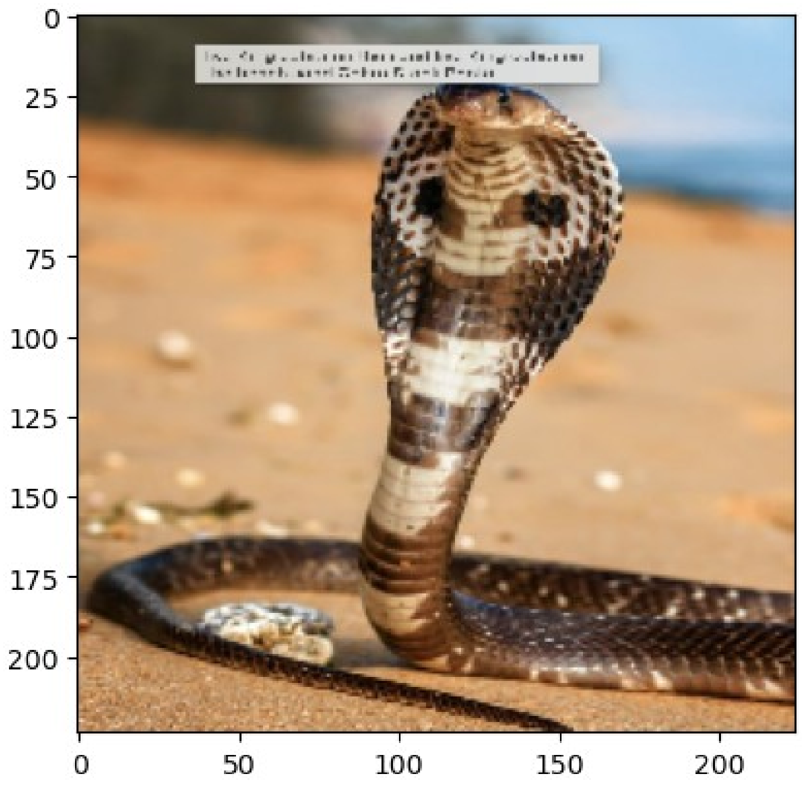
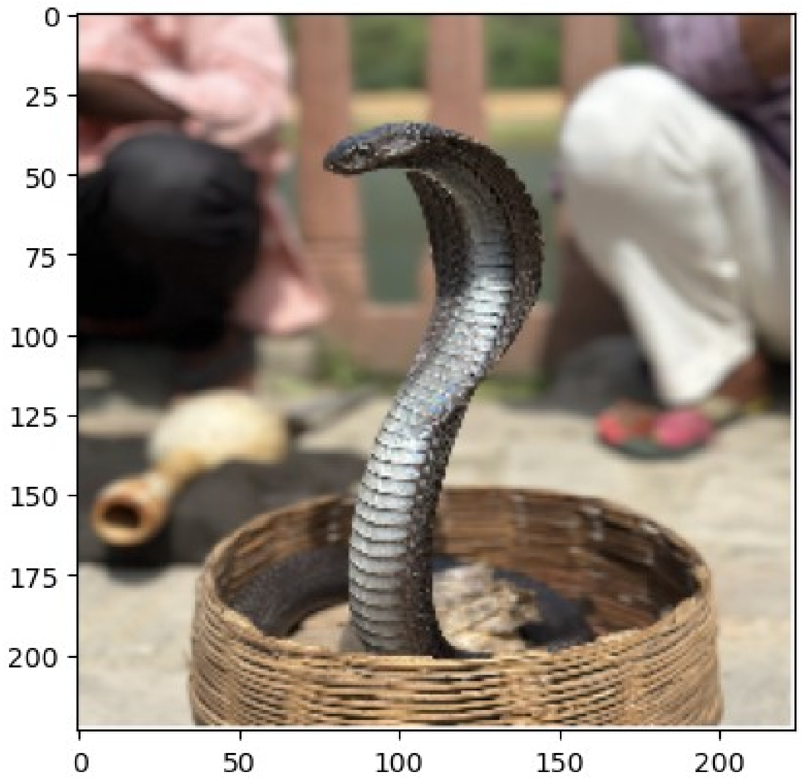
```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.applications.vgg16 import VGG16,
preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image

model = VGG16(weights='imagenet', include_top=True)

image_paths = [
    '/Users/sandeepreddy/Desktop/snake/blackcobra.jpeg',
    '/Users/sandeepreddy/Desktop/snake/whitecobra.jpeg',
    '/Users/sandeepreddy/Desktop/snake/blackmamba.jpeg'
]
for i, img_path in enumerate(image_paths):
    # Load and preprocess the image
    img = image.load_img(img_path, target_size=(224, 224))
    plt.imshow(img)
    plt.show()
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)

# target_class = 207
```



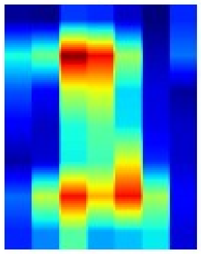




```
last_conv_layer = model.get_layer('block5_pool')
```

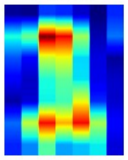
```
target_class = 207
grad_model = tf.keras.models.Model([model.inputs], [model.output,
last_conv_layer.output])
with tf.GradientTape() as tape:
    preds, conv_outputs = grad_model(x)
    class_output = preds[:, target_class]

grads = tape.gradient(class_output, conv_outputs)[0]

weights = tf.reduce_mean(grads, axis=(1, 2))

# Compute the weighted sum of the convolutional layer's output
weights = tf.expand_dims(tf.expand_dims(weights, axis=-1), axis=-1)
cam = tf.reduce_sum(tf.multiply(weights, conv_outputs), axis=-1)

cam_results =[]
cam = tf.maximum(cam, 0)
cam /= tf.reduce_max(cam)
cam = tf.image.resize(cam, (224, 224))
cam = np.asarray(cam)

cam_results.append(cam)
# print(cam_results)
# cam = tf.image.rgb_to_grayscale(cam)

# import scipy.ndimage
# # enlarged_cam = scipy.ndimage.zoom(cam[0], (2, 2), order=1)  # Zoom
factor of 2 for a 2x enlargement
enlarged_cam = np.repeat(cam[0], 25, axis=1)
# cam_results.append(enlarged_cam)

    plt.subplot(3, 2, 2*i+1)
    plt.imshow(img)
    plt.axis("off")

    plt.subplot(3, 2, 2*i+2)
    plt.title(f"Image {i + 1}")
    plt.imshow(enlarged_cam, cmap='jet')
    plt.axis("off")

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

Image 3



```
plt.figure(figsize=(18, 6))
for i, img_path in enumerate(image_paths):
    plt.subplot(3, len(image_paths), i + 1)
    img = image.load_img(img_path, target_size=(224, 224))
    plt.imshow(img)
    plt.axis("off")

    # Check if the index is within the range of cam_results
    if i < len(cam_results):
        plt.subplot(3, len(image_paths), len(image_paths) + i + 1)
        plt.title(f"Image {i + 1}")
        enlarged_cam = np.repeat(cam[0], 25, axis=1)
        plt.imshow(enlarged_cam, cmap='jet')
        plt.axis("off")

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

Image 1



Grad-CAM (Gradient-weighted Class Activation Mapping) is a popular technique in the field of explainable AI that provides insights into the decision-making process of deep learning models, particularly convolutional neural networks (CNNs). Grad-CAM helps answer the question: "Why did the model make a certain prediction?" It visualizes the regions within an input image that the model focuses on when making a prediction, making it easier for humans to understand and trust AI systems.

How the Grad CAM Works?

First we need to select a pre-trained deep learning model, that perform the classification on Images. We need to understand the target class for which we need to understand the model Prediction.Need to modify the model by adding a hook (a small modification) to capture the gradients during backpropagation. This hook is essential for computing gradients with respect to the final convolutional layer's output.

Pass the input image through the modified model. During the forward pass, activations from the chosen convolutional layer are collected. During the backward pass, gradients are computed specifically for the target class.

Multiply the feature maps from the final convolutional layer by the computed gradients. This step assigns importance scores to each feature map based on their contribution to the target class

Calculate the global average of the weighted feature maps. This step aggregates the importance scores and produces a single value for each feature map.

Apply a ReLU activation to focus on positive influences. These aggregated scores are used to generate the heatmap.

```
The main idea is to calculate the gradient loss with respect to the
input images. We generally calculate by below formal.
Sc = wT c + bc

where w is the derivate of Sc, with respect to Image(I).

And we visualize the gradient and we highlight the contribution of
each pixels as positive and negative.


In conclusion, the models thinks an image is y based on the
contribution of each pixel to map input image to class y.
```