```
import cv2
import numpy as np
import tensorflow as tf

def get_last_conv_layer(model):
    '''
    Returns the last conv layer of a given model.

    Returns the last layer of a given model with "conv" in name.

    :param keras_model model: The model in which a conv layer is searched for
    :return: Last concolution layer in model or None if there no conv layer is found
    :rtype: layer or None
    '''

    all_conv_layers = [layer for layer in model.layers if "conv" in layer.name]

    if (len(all_conv_layers) == 0):
        return None

    last_conv_layer = all_conv_layers[-1]

    return last_conv_layer

def grad_cam(model, x, class_index= None):
    '''
    Returns a grad-CAM heat map.
```

```
    Returns a grad-CAM heat map for a given image x in relation to one class (class_index) for a given model.

    :param keras_model model: The model to calculate the heatmap for
    :param np x: The image to calculate the heatmap for
    :param int class_index: The class index to calculate the heat map in relation to
    :return: grad-CAM heat map
    :rtype: np
    '''
    x = np.expand_dims(x, axis=0)
    x = tf.Variable(x, dtype=float)

    last_conv_layer = get_last_conv_layer(model)

    # First, we create a model that maps the input image to the activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [last_conv_layer.output, model.output]
    )

    # Then, we compute the gradient of the top predicted class for our input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(x)
        if class_index is None:
            class_index = tf.argmax(preds[0])
        class_channel = preds[:, class_index]

    # This is the gradient of the output neuron (top predicted or chosen)
```

```
# with regard to the output feature map of the last conv layer
grads = tape.gradient(class_channel, last_conv_layer_output)

# This is a vector where each entry is the mean intensity of the gradient
# over a specific feature map channel
pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

# We multiply each channel in the feature map array
# by "how important this channel is" with regard to the top predicted class
# then sum all the channels to obtain the heatmap class activation
last_conv_layer_output = last_conv_layer_output[0]
heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
heatmap = tf.squeeze(heatmap)

# For visualization purpose, we will also normalize the heatmap between 0 & 1
heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)

# Resize the heatmap to the size of the input image
heatmap = cv2.resize(heatmap.numpy(), (x.shape[2],x.shape[1]))

return heatmap
```