3. Method

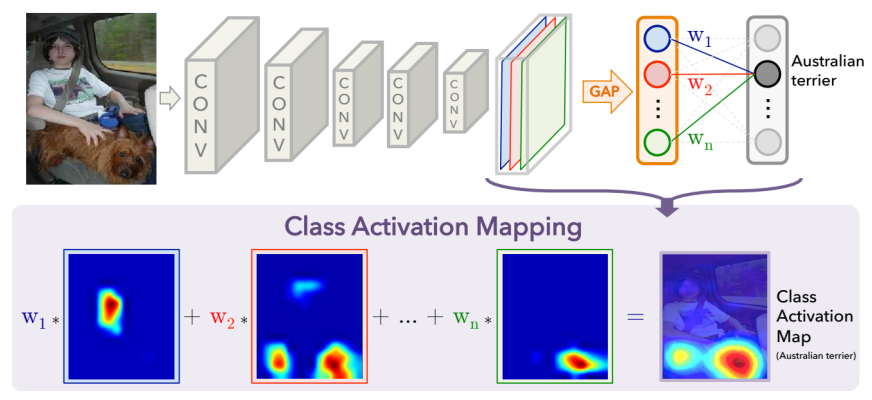
3. x. Class Activation Mapping (CAM)

人工神经网络已经在部分领域有了极大的应用，比如人脸识别，图像分类等，替人们做各种决策。而在现实生活中，有意义的决策是需要解释的。因此为了建立人们对于人工神经网络系统的信任，我们需要解释模型为什么会做出某些预测，是什么让模型做出这样的决策。这被称为模型的透明性。

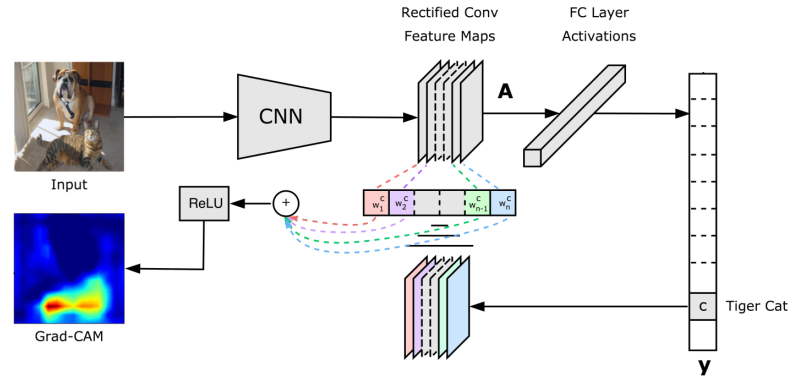
为了在复杂任务中有更好的性能表现，我们往往会使用多层的，复杂的模型。而这样的模型往往具有较差的可解释性。因此，针对复杂模型的可解释性的研究变得越来越重要。所以，我们使用了CAM和Grad-CAM进行可解释性研究。

以往研究表明CNN中的更深层表征捕获了更高层次的视觉结构，并且卷积层自然地保留了在全连接层中丢失的空间信息。因此Grad-CAM使用流入CNN最后一个卷积层的梯度信息，为每个神经元分配一个特定感兴趣的决策的重要性值。

CAM在最终输出层（softmax）之前，对卷积层的输出特征进行全局平均池化，作为全连接层的输入特征。在这个架构中，全连接层的权重与其对应的输出特征中的特征反映了图像某个区域的重要性。这种将输出层的权重投影回卷积层的输出特征的方法被称为CAM。按照全连接层权重将输出特征加权线性求和，最后叠加到原图即可得热力图。



Grad-CAM使用任何目标概念的梯度流入最终的卷积层，生成一个粗糙的定位图，突出显示图像中用于预测概念的重要区域。



Grad-CAM的流程分为两步。

第一步：计算网络对于某一类别预测的分数，然后反向传播计算预测分数相对于最后一个卷积层的输出特征的梯度，最后计算全局平均池化得到每个神经元的重要性权值，该值表示了输出特征中某一特征对于目标类别的重要性。

第二步：使用第一步中得到的权值，计算卷积层输出特征的线性加权和，最后使用Relu激活函数处理得到热力图。应用ReLU是因为我们只对那些对于目标类别有积极影响的特征感兴趣，而负像素可能属于图像中的其他类别。

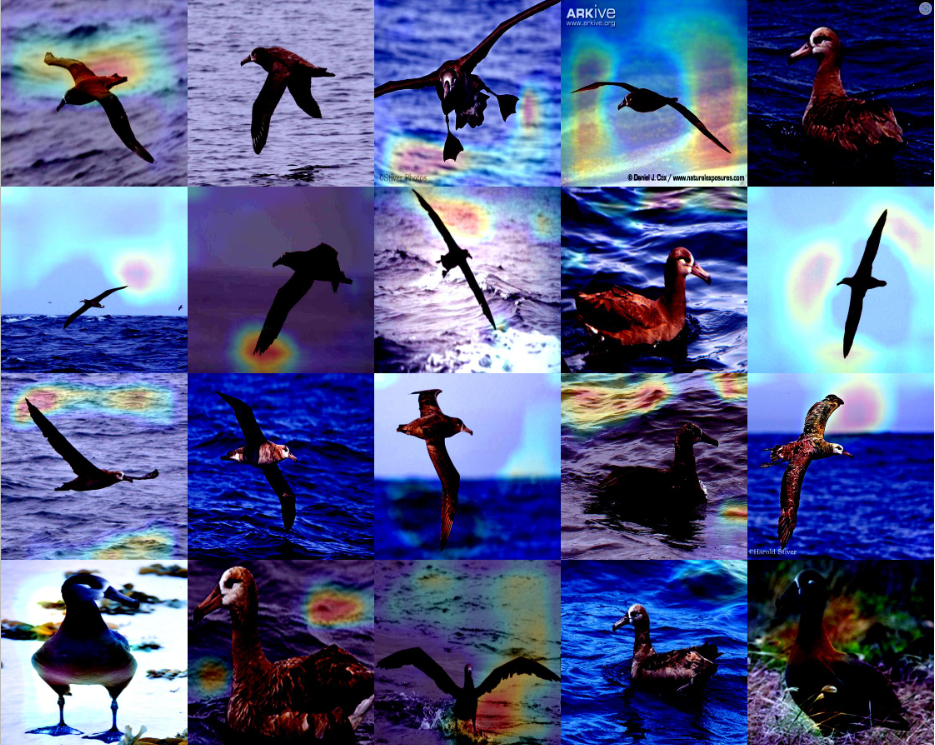
4. Experiment

在项目的main\_interpretability.py中，实现了生成单张图片的热力图的代码逻辑。但在真正生成之前，需要在trainer.py的inference函数中注册向前钩子函数hook\_fn\_forward（同样已在trainer.py中给出）提取保存模型卷积层的输入特征和输出特征。然后就可以读取全局平均池化层的输入特征（即最后一个卷积层的输出特征）和全连接层权重，调用项目提供的函数计算并保存热力图。批量生成热力图只需要在原来的代码的基础上添加遍历和处理逻辑即可。我实现了对于单个类别以及整个测试集的热力图生成逻辑。



生成单张热力图（左），批量生成热力图（右）

生成其他卷积层的输出特征基于CAM的热力图。这里我将hook注册到了最后一个卷积层，然后读取特征的时候读取的是输入特征（即倒数第二个卷积层的输出特征）。因为其大小与全连接层权重的大小并不契合（256和512），所以我使其经过一个卷积操作放大到512后再与权重相乘。可解释性不强。



生成其他卷积层的热力图（左），Grad-CAM生成的热力图（右）

Grad-CAM中的热力图由梯度和输出特征经过处理得到。为了得到梯度信息，需要编写一个向后钩子函数并在inference中注册该函数。然后在inference中执行推断，计算交叉熵损失并反向传播，就可以得到输出特征关于推断分数的梯度信息。得到输出特征和梯度信息后，对梯度的每个通道求均值得到权重，乘以输出特征相应的通道后将特征线性叠加在一起求平均，再用ReLU函数处理去掉负值，然后标准化（缩放到0到1之间）得到相应的热力图，最后改变其大小并叠加在原图后就可以保存下来。

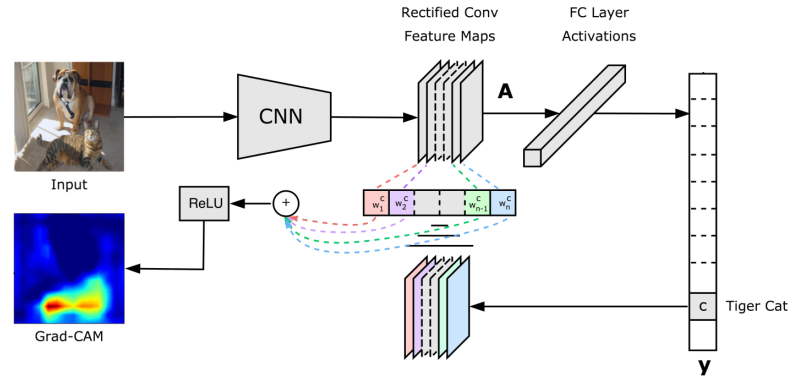
TODO Grad-CAM的效果看起来并不好，探究的什么原因。尝试Guided Grad-CAM

Artificial neural networks have found extensive applications in various fields such as facial recognition, image classification, and decision-making tasks. However, meaningful decisions made by these networks require explanations. Therefore, to establish trust in artificial neural network systems, we need to explain why the model makes certain predictions and what factors influence these decisions. This is known as model transparency.

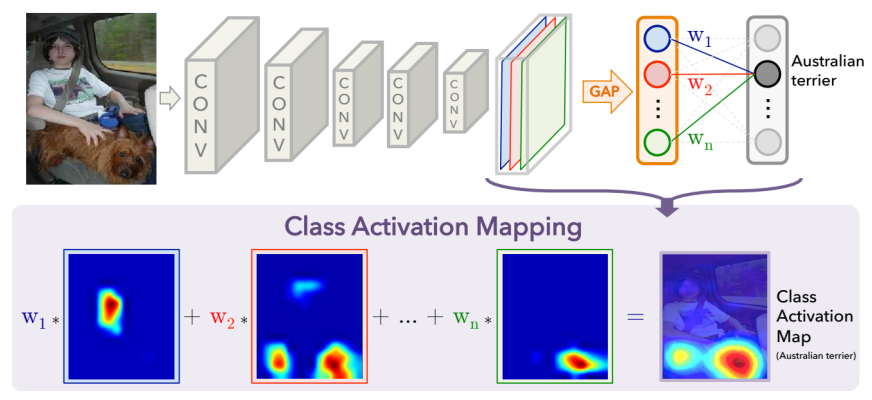
To achieve better performance in complex tasks, we often use multi-layered and complex models. However, such models often lack interpretability. Hence, research on the interpretability of complex models has become increasingly important. Therefore, we employed CAM (Class Activation Mapping) and Grad-CAM (Gradient-weighted Class Activation Mapping) for interpretability studies.

Previous research has shown that deeper representations in CNNs capture higher-level visual structures, and convolutional layers naturally preserve spatial information lost in fully connected layers. Therefore, Grad-CAM uses gradient information flowing into the last convolutional layer of the CNN to assign importance values to each neuron for a specific region of interest in decision-making.

CAM performs global average pooling on the output feature of the convolutional layer before the final output layer (softmax) to obtain input features for the fully connected layer. In this architecture, the weights of the fully connected layer correspond to the output features and reflect the importance of certain regions in the image. This method of projecting weights from the output layer back to the output features of the convolutional layer is called CAM. By linearly summing the output features weighted by the weights of the fully connected layer and overlaying them on the original image, a heatmap can be generated.



Grad-CAM calculates the gradient flowing into the final convolutional layer for any target concept to generate a rough localization map, highlighting the important areas of the image used for predicting the concept.



The process of Grad-CAM is divided into two steps.

1. Compute the network's prediction score for a specific category, then backpropagate to calculate the gradient of the prediction score with respect to the output features of the last convolutional layer. Finally, compute global average pooling to obtain importance weight values for each neuron, indicating the importance of a specific feature in the output features for the target category.
2. Use the obtained weights from step one to compute the linear weighted sum of the output features of the convolutional layer, and then apply the ReLU activation function to generate the heatmap. ReLU is applied because we are only interested in features that positively impact the target category, while negative pixels may belong to other categories in the image.

Experiment

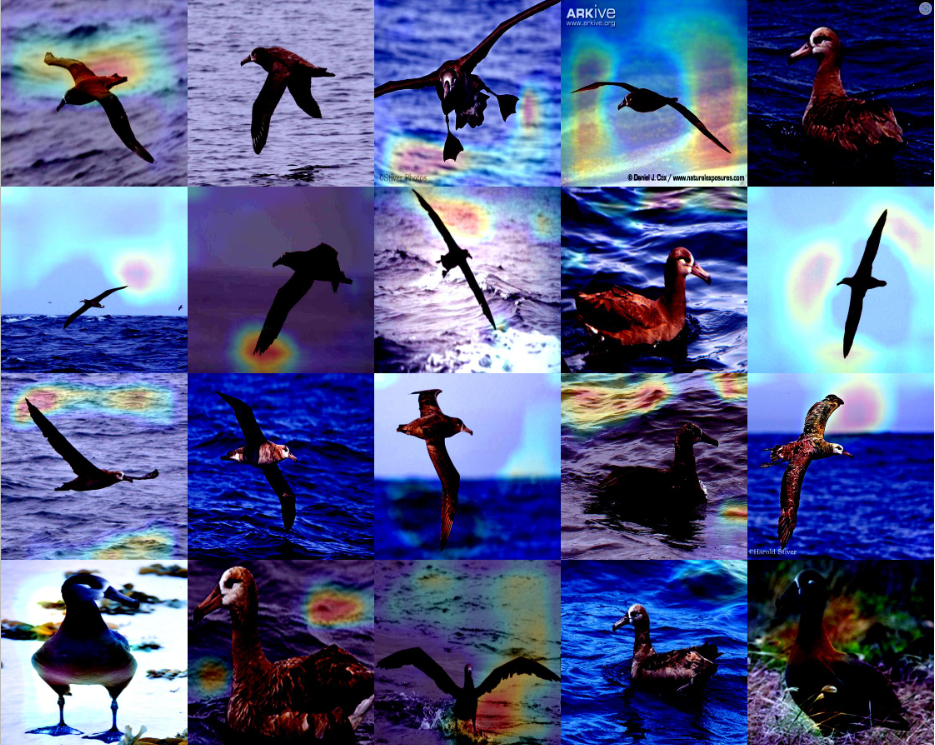
In the file main\_interpretability.py of the project, the code logic for generating a heatmap for a single image has been implemented. However, before actual generation, it's necessary to register a forward hook function hook\_fn\_forward (also provided in trainer.py) in the inference function of trainer.py to extract and save the input and output features of the model's convolutional layers. Then, you can read the input features of the global average pooling layer (i.e., the output features of the last convolutional layer) and the weights of the fully connected layer, and call the provided function in the project to compute and save the heatmap. Batch generation of heatmaps simply requires adding traversal and processing logic on top of the existing code. I implemented the logic for generating heatmaps for individual categories as well as the entire test set.



生成单张热力图（左），批量生成热力图（右）

Heatmap for Other Convolutional Layers (Left), Heatmap Generated by Grad-CAM (Right)

Generating heatmaps for other convolutional layers is based on CAM's heatmap. Here, I registered the hook to the last convolutional layer and then read the input features (i.e., the output features of the second-to-last convolutional layer) when extracting features. Because their sizes do not match (256 and 512), I resized the input features through a convolution operation to match the size of 512 before multiplying them with the weights. However, this method doesn't provide strong interpretability.



生成其他卷积层的热力图（左），Grad-CAM生成的热力图（右）

Heatmap for Other Convolutional Layers (Left), Heatmap Generated by Grad-CAM (Right)

The heatmap in Grad-CAM is derived from gradients and output features through processing. To obtain gradient information, a backward hook function needs to be written and registered in the inference function. Then, inference is executed in the inference function to calculate the cross-entropy loss and perform backpropagation, resulting in gradient information regarding the output features with respect to the inference score. After obtaining the output features and gradient information, weights are calculated by averaging the gradients across channels. These weights are then multiplied by the corresponding channels of the output features, linearly combined, averaged, passed through the ReLU function to remove negative values, normalized (scaled to between 0 and 1), and resized. Finally, they are overlaid on the original image to generate the heatmap, which can then be saved.