Homework 1: Explainability

助教：莫易川

邮箱：mo666666@stu.pku.edu.cn

CONTENTS

CONTENTS

Brief Introduction:

Points

- Coding (20 points: 3+5+5+7)
- Paper reading (5 points: 3+1+1)

Requirements

- Word/pdf is both ok.
- Write a report (at most **8** pages).
- Send your report and code to

  trustworthy_ai@163.com

  Theme: Homework1-name-ID
- In Chinese/ English

Language and wheel

- Python
- PyTorch

Bonus

- The first five student who submits their homework will receive 1 extra bonus point

Due: 4/7 24:00 (+3 days delay is allowed)

Environment:

jupyter in ANACONDA

Not

and

Reference:

Amaconda Installation:
https://blog.csdn.net/qq_42257666/article/details/121383450

The usage of Jupyter:
https://zhuanlan.zhihu.com/p/33105153

The documents of Pytorch:
https://pytorch.org/docs/stable/index.html

If you like, you could also use **Pycharm** and **Vscode** for coding

CONTENTS

02 Coding (Preparation)

Package installation:

Add Code Cell | Add Markdown Cell

In 3

```
# Install other package if neccesary.
!pip install lime==0.1.1.37
```

```
Requirement already satisfied: scikit-learn>=0.18 in /home1/moyichuan/miniconda3/envs/skm/lib/python3
.7/site-packages (from lime==0.1.1.37) (1.0.2)
Requirement already satisfied: matplotlib in /home1/moyichuan/miniconda3/envs/skm/lib/python3
.7/site-packages (from lime==0.1.1.37) (3.5.2)
Requirement already satisfied: scipy in /home1/moyichuan/miniconda3/envs/skm/lib/python3.7/site-packages
(from lime==0.1.1.37) (1.7.3)
Requirement already satisfied: numpy in /home1/moyichuan/miniconda3/envs/skm/lib/python3.7/site-packages
(from lime==0.1.1.37) (1.21.6)
Requirement already satisfied: progressbar in /home1/moyichuan/miniconda3/envs/skm/lib/python3
.7/site-packages (from lime==0.1.1.37) (2.5)
```

Import package

In 4

```
# Import package
import torch
import json
import cv2
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from lime import lime_image
from torchvision.datasets import ImageFolder
import torch.nn as nn
from torchvision.models.resnet import resnet50
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from skimage.segmentation import slic
from torch.autograd import Variable
from typing import Callable, List, Tuple
```

Test whether jupyter works well:

```
# Test whether the jupyter works well.
np.random.seed(10)
img = Image.open('./test.JPEG')
img = img.convert('RGB')
transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
])
img = transform(img)
img_hwc = img.permute(1, 2, 0)
plt.imshow(img_hwc)
plt.axis("off")
plt.show()
img = img.unsqueeze(dim=0)
```

02 Coding (Preparation)

Load model

In 8

```
# Load the model
class Normalize(nn.Module):
    def __init__(self, mean, std):
        super(Normalize, self).__init__()
        self.mean = torch.tensor(mean)
        self.std = torch.tensor(std)

    def forward(self, x):
        return (x - self.mean.type_as(x)[None,:,None,None]) / self.std.type_as(x)[None,:,None,None]
imagenet_mean = (0.485, 0.456, 0.406)
imagenet_std = (0.229, 0.224, 0.225)
net = resnet50(num_classes=1000, pretrained=True)
model = nn.Sequential(Normalize(mean=imagenet_mean, std=imagenet_std), net)
model.eval()
```

Load dataset

In 9

```
# Load the dataset
dataset = ImageFolder(root="./ImageNet_subset/", transform=transform)
dataset_loader = DataLoader(dataset, batch_size=10, num_workers=6)
for image, _ in dataset_loader:
    result = model(image)
    real_label = torch.argmax(result,dim = -1).numpy()


# Print the label
for j, label in enumerate(real_label):
    class_name = json.load(open("imagenet_class_index.json"))[str(label.item())]
    print("The label of image {:d} is :".format(j), class_name[1])


# Show the image
img_indices = [i for i in range(10)]
all_image,_ = next(iter(dataset_loader))
all_image = all_image.mul(255).add_(0.5).clamp_(0, 255).permute(0, 2, 3, 1).to('cpu', torch.uint8).numpy()
fig, axs = plt.subplots(1, len(img_indices), figsize=(15, 8))
for i, img in enumerate(all_image):
  axs[i].imshow(img)
  axs[i].set_xticks([])
  axs[i].set_yticks([])
```

02 Coding (Line) 3point

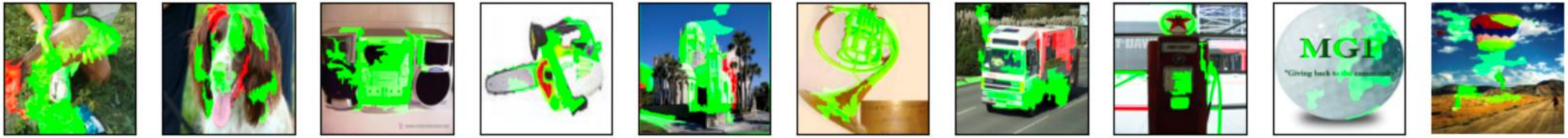
In 10

```python
def predict(input):
    # input: numpy array, (batches, height, width, channels)
    # output: the output of the model.
    # pass
    model.eval()
    input = torch.FloatTensor(input).permute(0, 3, 1, 2)
    output = model(input)
    return output.detach().numpy()
    ###################################
    # write the code here
    # return output.detach().numpy()
    ###################################

def segmentation(input):
    # split the image into 200 pieces with the help of segmentaion from skimage
    # doc: https://scikit-image.org/docs/stable/api/skimage.segmentation.html#slic
    return slic(input, n_segments=200, compactness=1, sigma=1, start_label=1)

img_indices = [i for i in range(10)]
fig, axs = plt.subplots(1, len(img_indices), figsize=(15, 8))


# fix the random seed to make it reproducible
for idx, (image,label) in enumerate(zip(all_image,real_label)):
    x = (image/255).astype(np.double)

    # Refer the doc: https://lime-ml.readthedocs.io/en/latest/lime.html?highlight=explain_instance#lime.lime_image.LimeImageExplainer.explain_instance
    ###################################
    # write the code here
    ###################################
    explainer = lime_image.LimeImageExplainer()
    explaination = explainer.explain_instance(image=x, classifier_fn=predict, segmentation_fn=segmentation)


    # Turn the result from explainer to the image
    # doc: https://lime-ml.readthedocs.io/en/latest/lime.html?highlight=get_image_and_mask#lime.lime_image.ImageExplanation.get_image_and_mask
    lime_img, mask = explaination.get_image_and_mask(label=label.item(),positive_only=False,hide_rest=False,num_features=11,min_weight=0.05)
    axs[idx].imshow(lime_img)
    axs[idx].set_xticks([])
    axs[idx].set_yticks([])

plt.xticks([])
plt.yticks([])
plt.axis('off')
plt.show()
plt.close()
```

2 point

1 point

Results:



02 Coding (Saliency Map) 5 point

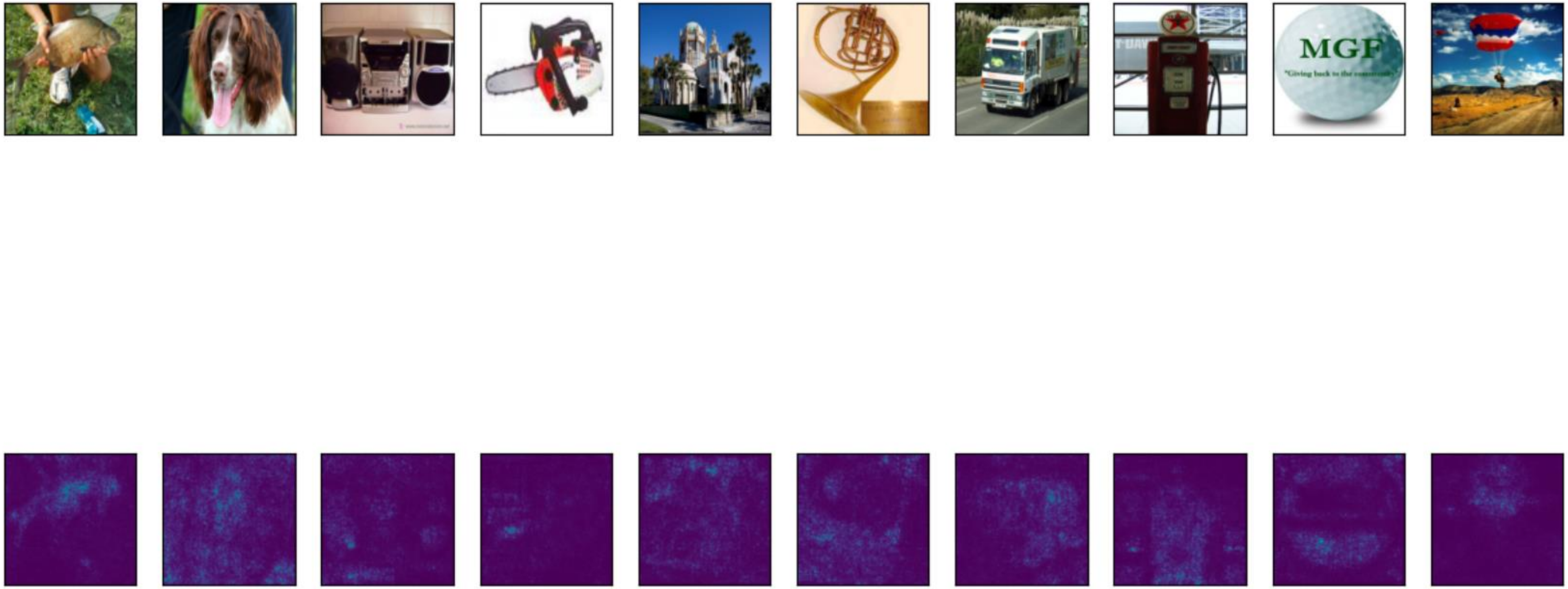
```python
def normalize(image):
  return (image - image.min()) / (image.max() - image.min())

def compute_saliency_maps(x, y, model):
  # input: the input image, the ground truth label, the model
  # output: the saliency maps of the images
  # We need to normalize each image, because their gradients might vary i
  pass
  ###################################
  # write the code here
  # return saliencies
  ###################################

image,_ = next(iter(dataset_loader))
result = model(image)
label = torch.argmax(result,dim = -1)



saliencies = compute_saliency_maps(image, label, model)
# visualize
fig, axs = plt.subplots(2, len(img_indices), figsize=(15, 8))
```

5 point

```python
for row, target in enumerate([image, saliencies]):
  for column, img in enumerate(target):
    if row==0:
      axs[row][column].imshow(img.permute(1, 2, 0).detach().mul(255).add_(0.5).clamp_(0, 255).to('cpu',
       torch.uint8).numpy())
      axs[row][column].set_xticks([])
      axs[row][column].set_yticks([])
    else:
      axs[row][column].imshow(img.numpy())
      axs[row][column].set_xticks([])
      axs[row][column].set_yticks([])

plt.show()
plt.close()
```

02 Coding (Saliency Map) 5 point

Results:



02 Coding (Smooth-grad) 5 point

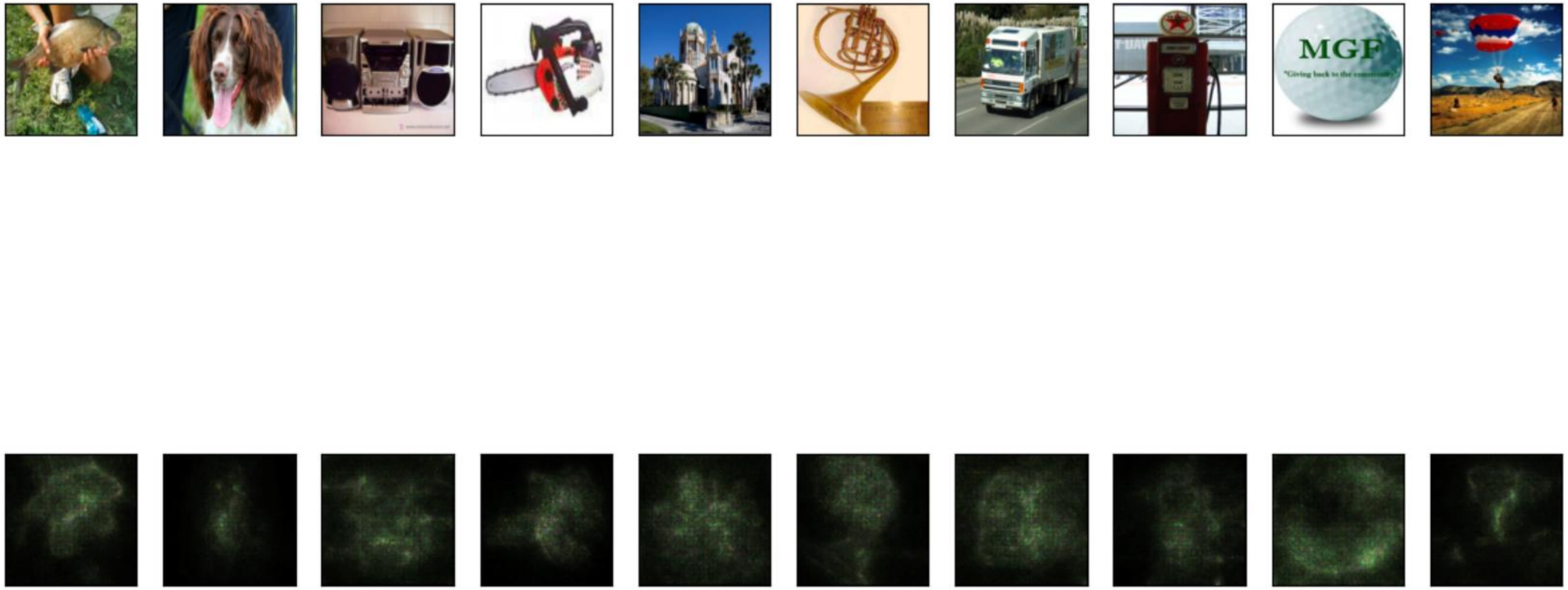
```python
def normalize(image):
  return (image - image.min()) / (image.max() - image.min())

def smooth_grad(x, y, model, epoch, param_sigma_multiplier):
  # input: the input image, the ground truth label, the model,
   calculating std of noise
  # output: the saliency maps of the images
  pass
  ###################################
  # write the code here
  # return smooth
  ###################################


smooth = []
image,_ = next(iter(dataset_loader))
result = model(image)
label = torch.argmax(result,dim = -1)


for i, l in zip(image, label):
  smooth.append(smooth_grad(i, l, model, 10, 0.4))
smooth = np.stack(smooth)
```

5 point

```python
fig, axs = plt.subplots(2, len(img_indices), figsize=(15, 8))
for row, target in enumerate([image, smooth]):
  for column, img in enumerate(target):
    if row==0:
        axs[row][column].imshow(img.permute(1, 2, 0).detach().mul(255).add_(0.5).clamp_(0, 255).to('cpu',
         torch.uint8).numpy())
        axs[row][column].set_xticks([])
        axs[row][column].set_yticks([])
    else:
        axs[row][column].imshow(np.transpose(img.reshape(3,224,224), (1,2,0)))
        axs[row][column].set_xticks([])
        axs[row][column].set_yticks([])
```

Coding (Smooth-grad) 5point

Results:



02 Coding (grad-cam) 7 point

```python
def scale_cam_image(cam, target_size=None):
    result = []
    for img in cam:
        img = img - np.min(img)
        img = img / (1e-7 + np.max(img))
        if target_size is not None:
            img = cv2.resize(img, target_size)
        result.append(img)
    result = np.float32(result)
    return result

class ActivationsAndGradients:
    """ Class for extracting activations and
    registering gradients from targetted intermediate layers """

    def __init__(self, model, target_layers):
        self.model = model
        self.gradients = []
        self.activations = []
        self.handles = []
        for target_layer in target_layers:
            self.handles.append(
                target_layer.register_forward_hook(self.save_activation))
            self.handles.append(
                target_layer.register_forward_hook(self.save_gradient))
```

```python
    def save_activation(self, module, input, output):
        activation = output
        self.activations.append(activation.cpu().detach())

    def save_gradient(self, module, input, output):
        if not hasattr(output, "requires_grad") or not output.requires_grad:
            # You can only register hooks on tensor requires grad.
            return
        # Gradients are computed in reverse order
        def _store_grad(grad):
            self.gradients = [grad.cpu().detach()] + self.gradients
        output.register_hook(_store_grad)

    def __call__(self, x):
        self.gradients = []
        self.activations = []
        return self.model(x)

    def release(self):
        for handle in self.handles:
            handle.remove()
```

02 Coding (grad-cam) 7 point

```python
def show_cam_on_image(img: np.ndarray,
                      mask: np.ndarray,
                      use_rgb: bool = False,
                      colormap: int = cv2.COLORMAP_JET,
                      image_weight: float = 0.5) -> np.ndarray:
    """ This function overlays the cam mask on the image as an heatmap.
    By default the heatmap is in BGR format.
    :param img: The base image in RGB or BGR format.
    :param mask: The cam mask.
    :param use_rgb: Whether to use an RGB or BGR heatmap, this should be set to True if 'img' is in RGB format.
    :param colormap: The OpenCV colormap to be used.
    :param image_weight: The final result is image_weight * img + (1-image_weight) * mask.
    :returns: The default image with the cam overlay.
    """
    heatmap = cv2.applyColorMap(np.uint8(255 * mask), colormap)
    if use_rgb:
        heatmap = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB)
    heatmap = np.float32(heatmap) / 255

    if np.max(img) > 1:
        raise Exception(
            "The input image should np.float32 in the range [0, 1]")

    if image_weight < 0 or image_weight > 1:
        raise Exception(
            f"image_weight should be in the range [0, 1].\
                Got: {image_weight}")

    cam = (1 - image_weight) * heatmap + image_weight * img
    cam = cam / np.max(cam)
    return np.uint8(255 * cam)
```

```python
class ClassifierOutputTarget:
    def __init__(self, category):
        self.category = category

    def __call__(self, model_output):
        if len(model_output.shape) == 1:
            return model_output[self.category]
        return model_output[:, self.category]


class GradCAM:
    def __init__(self,
                 model: torch.nn.Module,
                 target_layers: List[torch.nn.Module]) -> None:
        self.model = model.eval()
        self.target_layers = target_layers
        self.model = model
        self.activations_and_grads = ActivationsAndGradients(
            self.model, target_layers)

    """ Get a vector of weights for every channel in the target layer.
    Methods that return weights channels,
    will typically need to only implement this function. """

    def get_cam_weights(self,
                        grads: torch.Tensor) -> np.ndarray:
        return np.mean(grads, axis=(2, 3))
```

02 Coding (grad-cam) 7 point

```python
def show_cam_on_image(img: np.ndarray,
                      mask: np.ndarray,
                      use_rgb: bool = False,
                      colormap: int = cv2.COLORMAP_JET,
                      image_weight: float = 0.5) -> np.ndarray:
    """ This function overlays the cam mask on the image as an heatmap.
    By default the heatmap is in BGR format.
    :param img: The base image in RGB or BGR format.
    :param mask: The cam mask.
    :param use_rgb: Whether to use an RGB or BGR heatmap, this should be set to True if 'img' is in RGB format.
    :param colormap: The OpenCV colormap to be used.
    :param image_weight: The final result is image_weight * img + (1-image_weight) * mask.
    :returns: The default image with the cam overlay.
    """
    heatmap = cv2.applyColorMap(np.uint8(255 * mask), colormap)
    if use_rgb:
        heatmap = cv2.cvtColor(heatmap, cv2.COLOR_BGR2RGB)
    heatmap = np.float32(heatmap) / 255

    if np.max(img) > 1:
        raise Exception(
            "The input image should np.float32 in the range [0, 1]")

    if image_weight < 0 or image_weight > 1:
        raise Exception(
            f"image_weight should be in the range [0, 1].\
                Got: {image_weight}")

    cam = (1 - image_weight) * heatmap + image_weight * img
    cam = cam / np.max(cam)
    return np.uint8(255 * cam)
```

```python
class ClassifierOutputTarget:
    def __init__(self, category):
        self.category = category

    def __call__(self, model_output):
        if len(model_output.shape) == 1:
            return model_output[self.category]
        return model_output[:, self.category]


class GradCAM:
    def __init__(self,
                 model: torch.nn.Module,
                 target_layers: List[torch.nn.Module]) -> None:
        self.model = model.eval()
        self.target_layers = target_layers
        self.model = model
        self.activations_and_grads = ActivationsAndGradients(
            self.model, target_layers)

    """ Get a vector of weights for every channel in the target layer.
        Methods that return weights channels,
        will typically need to only implement this function. """

    def get_cam_weights(self,
                        grads: torch.Tensor) -> np.ndarray:
        return np.mean(grads, axis=(2, 3))
```

02 Coding (grad-cam) 7 point

```python
    def get_cam_image(self,
                      activations: torch.Tensor,
                      grads: torch.Tensor) -> np.ndarray:
        # input: the activation, the gradient(4D tensor)
        # output: cam of a specific layer
        pass
        ###################################
        # write the code here
        # return cam
        ###################################

    def forward(self,
                input_tensor: torch.Tensor,
                targets: List[torch.nn.Module]) -> np.ndarray:
        outputs = self.activations_and_grads(input_tensor)
        self.model.zero_grad()
        loss = sum([target(output)
                    for target, output in zip(targets, outputs)])
        loss.backward(retain_graph=True)
        # In most of the saliency attribution papers, the saliency is
        # computed with a single target layer.
        # Commonly it is the last convolutional layer.
        # Here we support passing a list with multiple target layers.
        # It will compute the saliency image for every image,
        # and then aggregate them (with a default mean aggregation).
        # This gives you more flexibility in case you just want to
        # use all conv layers for example, all Batchnorm layers,
        # or something else.
        cam_per_layer = self.compute_cam_per_layer(input_tensor,
                                                   targets)
        return self.aggregate_multi_layers(cam_per_layer)
```

3 point

```python
    def compute_cam_per_layer(
            self,
            input_tensor: torch.Tensor) -> np.ndarray:
        activations_list = [a.cpu().data.numpy()
                            for a in self.activations_and_grads.activations]
        grads_list = [g.cpu().data.numpy()
                      for g in self.activations_and_grads.gradients]
        target_size =  (input_tensor.size(-1), input_tensor.size(-2))
        cam_per_target_layer = []
        # Loop over the saliency image from every layer
        for i in range(len(self.target_layers)):
            target_layer = self.target_layers[i]
            layer_activations = None
            layer_grads = None
            if i < len(activations_list):
                layer_activations = activations_list[i]
            if i < len(grads_list):
                layer_grads = grads_list[i]
            ###################################
            # write the code here
            ###################################
        return cam_per_target_layer

    def aggregate_multi_layers(
            self,
            cam_per_target_layer: np.ndarray) -> np.ndarray:
        cam_per_target_layer = np.concatenate(cam_per_target_layer, axis=1)
        cam_per_target_layer = np.maximum(cam_per_target_layer, 0)
        result = np.mean(cam_per_target_layer, axis=1)
        return scale_cam_image(result)
```

4 point

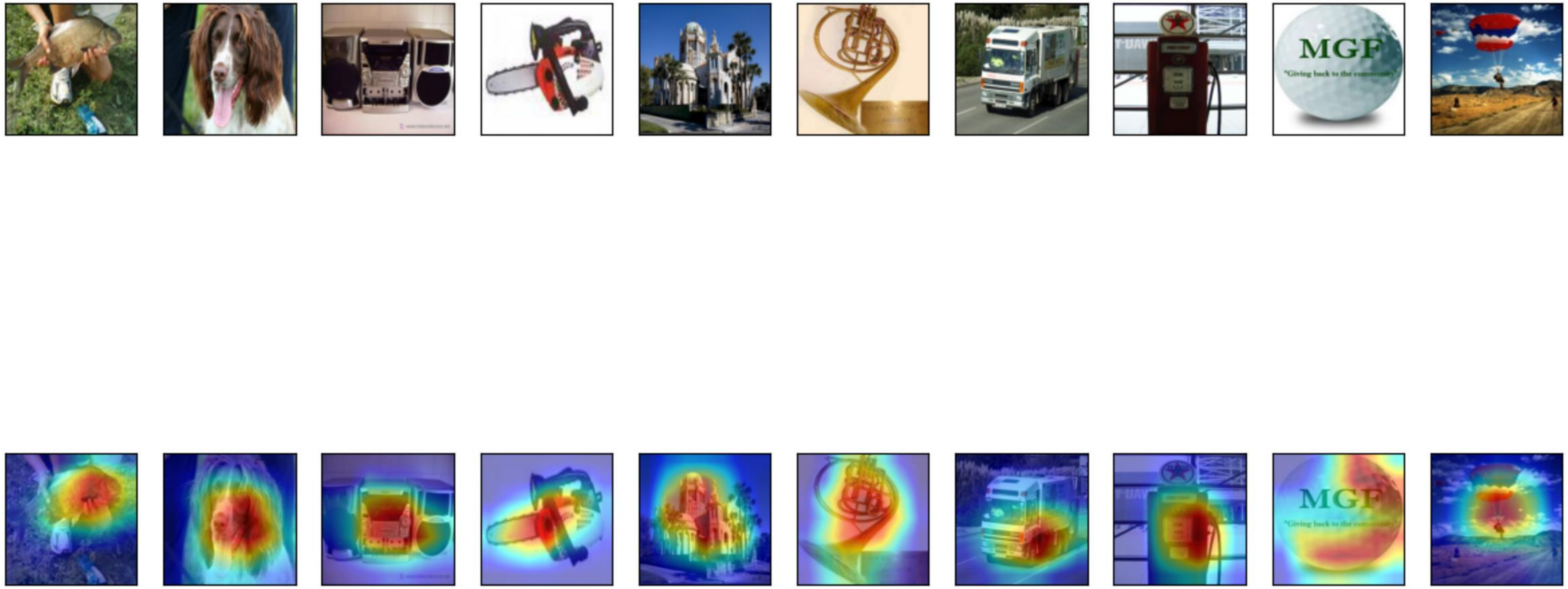
02 Coding (grad-cam) 7 point

```python
def __call__(self,
             input_tensor: torch.Tensor,
             targets: List[torch.nn.Module] = None) -> np.ndarray:
    return self.forward(input_tensor,
                        targets)

def __del__(self):
    self.activations_and_grads.release()

def __enter__(self):
    return self

def __exit__(self, exc_type, exc_value, exc_tb):
    self.activations_and_grads.release()
    if isinstance(exc_value, IndexError):
        # Handle IndexError here...
        print(
            f"An exception occurred in CAM with block: {exc_type}. Message: {exc_value}")
        return True
```

```python
model = resnet50(pretrained=True)
label = torch.argmax(result,dim = -1).numpy()
target_layers = [model.layer4[-1]]
image,_ = next(iter(dataset_loader))
cam = GradCAM(model=model, target_layers=target_layers)
targets = [ClassifierOutputTarget(i) for i in label]
grayscale_cam = cam(input_tensor=image, targets=targets)
fig, axs = plt.subplots(2, len(img_indices), figsize=(15, 8))
for column, single_image in enumerate(image):
    axs[0][column].imshow(single_image.permute(1, 2, 0).detach().mul(255).add_(0.5).clamp_(0, 255).to('cpu',
     torch.uint8).numpy())
    axs[0][column].set_xticks([])
    axs[0][column].set_yticks([])
    axs[1][column].imshow(show_cam_on_image(single_image.permute(1, 2, 0).detach().to('cpu').numpy(),
     grayscale_cam[column],use_rgb=True))
    axs[1][column].set_xticks([])
    axs[1][column].set_yticks([])
```

02 Coding (grad-cam) 5point

Results:



CONTENTS

03 Paper reading

Requirement:

- The brief summary(<120 words) 1point
- The strength(<120 words) 1point
- The weakness(<120 words) 1point
- Quality 1point
- Quantity (>5 papers) 1point

Reference:

https://openreview.net/

You can get 4 points if you just read one paper!

论文列表

[1] Learning Deep Features for Discriminative Localization

[2] Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization

[3] Grad-CAM++: Improved Visual Explanations for Deep Convolutional Networks

[4] Smooth Grad-CAM++: An Enhanced Inference Level Visualization Technique for Deep Convolutional Neural Network Models

[5] Score-CAM: Score-Weighted Visual Explanations for Convolutional Neural Networks

[6] SS-CAM: Smoothed Score-CAM for Sharper Visual Feature Localization

[7] IS-CAM: Integrated Score-CAM for axiomatic-based explanations

[8] Axiom-based Grad-CAM: Towards Accurate Visualization and Explanation of CNNs

论文列表

[9] SmoothGrad: removing noise by adding noise.

[10] Quantifying Attention Flow in Transformers

[11] Transformer Interpretability Beyond Attention Visualization

[12] Explaining deep neural networks and beyond: A review of methods and applications

[13] Machine learning interpretability: A survey on methods and metrics

[14] Towards explainable artificial intelligence

[15] Explaining explanations: An overview of interpretability of machine learning

Q&A

Thanks