Georgia Institute of Technology

ECE 4803: Fundamentamentals of Machine Learning (FunML)

Spring 2022

Homework Assignment # 7

Due: Monday, April 25th, 2022 @8PM

Please read the following instructions carefully.

- The entire homework assignment is to be completed on this `ipython` notebook. It is designed to be used with `Google Colab`, but you may use other tools (e.g., Jupyter Lab) as well.
- Make sure that you execute all cells in a way so their output is printed beneath the corresponding cell. Thus, after successfully executing all cells properly, the resulting notebook has all the questions and your answers.
- Print a PDF copy of the notebook with all its outputs printed and submit the **PDF** on `Canvas` under Assignments.
- Make sure you delete any scratch cells before you export this document as a PDF. Do not change the order of the questions and do not remove any part of the questions. Edit at the indicated places only.
- Rename the PDF according to the format: ***LastName_FirstName_ECE_4803_sp22_assignment_#.pdf***
- It is encouraged for you to discuss homework problems amongst each other, but any copying is strictly prohibited and will be subject to Georgia Tech Honor Code.
- Late homework is not accepted unless arranged otherwise and in advance.
- Comment on your codes.
- Refer to the tutorial and the supplementary/reading materials that are posted on `Canvas` for lectures 22, 23 to help you with this assignment.

- **IMPORTANT:** Start your solution with a **BOLD RED** text that includes the words *solution* and the part of the problem you are working on. For example, start your solution for Part (c) of Problem 2 by having the first line as:
  Solution to Problem 2 Part (c). Failing to do so may result in a *20% penalty* of the total grade.

Assignment Objectives:

- Learn to use different methods for model explainability
- Learn how to use hook feature in `Pytorch`
- Implement Grad-CAM and Contrast-CAM

Guide for Exporting Ipython Notebook to PDF:

Here is a video summarizes how to export Ipythin Notebook into PDF.

- **[Method1: Print to PDF]**
  After you run every cell and get their outputs, you can use **[File] -> [Print]** and then choose **[Save as PDF]** to export this Ipython Notebook to PDF for submission.
  Note: Sometimes figures or texts are splited into different pages. Try to tweak the layout by adding empty lines to avoid this effect as much as you can.
- **[Method2: colab-pdf script]**
  The author of that video provided an alternative method that can generate better layout PDF. However, it only works for Ipythin Notebook without embedded images.
  How to use: Put the script below into cells at the end of your Ipythin Notebook. After you run the fisrt cell, it will ask for google drive permission. Executing the second cell will generate the PDF file in your google drive home directory. Make sure you use the correct path and file name.

```
## this will link colab with your google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
%%capture
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('LastName_FirstName_ECE_4803_sp22_assignment_#.ipynb') ## change path and file name
```

- **[Method3: GoFullPage Chrome Extension] (most recommended)**
  Install the extension and generate PDF file of the Ipython Notebook in the browser.

Note: Georgia Tech provides a student discount for Adobe Acrobat subscription. Further information can be found here.

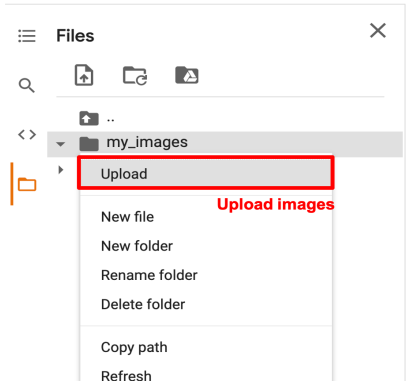
Problem 1: Direct Explainability and Visualization using Grad-CAM (60pts)

In class, we have learnt different approaches for model explainability and visualization such as saliency via occlusion, guided backpropagation, and Grad-CAM etc. In this problem, we are going to guide you implement Grad-CAM method step by step. For each image, Grad-CAM can provide a saliency heatmap based on weighting a convolution layer’s activation maps according to their contributions to the predicting class. This saliency heatmap not only shows the important area in the image, but also provides class-discriminative information. Hence, Grad-CAM helps us understand which area of the image our trained network relys on during classification, and we can also use this information to provide explanations of the decision making in our neural network.

(a) Load Images

Load those images that we used in Assignment 5 Problem 4 and visualize them.

Create a folder named `my_images` using the code below (`!mkdir my_images`) to store all your uploaded images. Then, upload your own images (at least 12 images) to the created `my_images` folder as the step shown below.



Problem 1 (a) Solution

```
# create a folder named `my_images`
!mkdir my_images
```

```
# visualize images

from PIL import Image
import matplotlib.pyplot as plt
from glob import glob
import os
import numpy as np

def imread(img_dir):
  # read the images into a list of `PIL.Image` objects
  images = []
  for f in glob(os.path.join(img_dir, "*")):
    images.append(Image.open(f).convert('RGB'))
  return images

def vis_img_label(image_list, label_list=None):
  # visualize the images w/ labels
  Tot = len(image_list)
  Cols = 4
  Rows = Tot // Cols
  Rows += (Tot % Cols)>0
  if label_list is None:
    label_list = [""]*Tot
  # Create a Position index
  Position = range(1,Tot + 1)
  fig = plt.figure(figsize=(Cols*5, Rows*5))
  for i in range(Tot):
    image = image_list[i]
    # add every single subplot to the figure
    ax = fig.add_subplot(Rows,Cols,Position[i])
    ax.imshow(np.asarray(image))
    ax.set_title(label_list[i])


## Load your uploaded images
img_dir = "/content/my_images"
image_list = imread(img_dir)
## visualize your uploaded images
vis_img_label(image_list)
```

(b) Create Custom Dataset

Create the custom dataset as you did in Assignment 5.

The class `myDataset` overrides the following methods:

- `__len__` that returns the size of `myDataset` (the length of the `img_list`)
- `__getitem__` to support the indexing such that `myDataset[i]` can be used to get the i-th sample of `img_list`.

Your tasks are:

- Write your code to return the size of `myDataset` in the method `__len__`.
- Write your code to index the i-th sample of `img_list` in the method `__getitem__`.

Problem 1 (b) Solution

```
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms
import torch
import torch.nn as nn
import torch.nn.functional as F

# customized pytorch dataset
class myDataset(Dataset):
  def __init__(self, img_list, data_transform=None):

    self.img_list = img_list   # the list of all uploaded Images
    self.length = len(img_list)
    self.data_transform = data_transform

  def __getitem__(self, index):
    """function extracts a single example from img_list given its index.

    Parameters
    ----------
    index : int
```

```
      index of a single example to be extracted from img_list

    Returns
    -------
    img : torch.tensor, shape(3, 224, 224), type torch.float
      indexed example from img_list reshaped into a RGB channel image of
      size 224 x 224 and float datatype.
    """

    img = self.img_list[index]
    # img = img.type(torch.float)
    # img = torch.reshape(img, (3, 224, 224))
    # img = img.to('cuda')

    # img = ##TODO

    if self.data_transform is not None:
        img =self.data_transform(img)  # apply data transformations
    assert img.shape == (3, 224, 224)

    return img

  def __len__(self):
    """
    Returns
    -------
    length : int
      length of img_list.
    """
    length = len(self.img_list)  ##TODO

    return length

#-----------------Don't change anything below------------------------#

data_transform = transforms.Compose([transforms.Resize((224, 224)),
                                     transforms.ToTensor(),
                                        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                             std=[0.229, 0.224, 0.225])])

my_dataset = myDataset(image_list, data_transform)
my_dataloader = DataLoader(my_dataset, batch_size=1, shuffle=False, num_workers=2)

#----verify the customized dataset `myDataset`----#
print('The size of the dataset: ', len(my_dataset))
print('The dimension of the first image sample after transformations: ', my_dataset[0].shape)
```

```
The size of the dataset:  12
The dimension of the first image sample after transformations:  torch.Size([3, 224, 224])
```

(c) Load Vgg16

The provided code for this part loads the pre-trained Vgg16 using the `torchvision.models` module. The pre-trained model is constructed by passing `pretrained=True`. Execute the code below as is and observe the printed AlexNet architecture. We will use Vgg16 to visualize Grad-CAM result.

Problem 1 (c) Solution

```
import torchvision

#Load the pre-trained AlexNet
vgg16 = torchvision.models.vgg16(pretrained=True) ##TODO
print(vgg16)   # print the model architecture
```

```
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/checkpoints/vgg16-397923af.pth
100%                                          528M/528M [00:04<00:00, 133MB/s]
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
```

```
        (2): Dropout(p=0.5, inplace=False)
        (3): Linear(in_features=4096, out_features=4096, bias=True)
        (4): ReLU(inplace=True)
        (5): Dropout(p=0.5, inplace=False)
```

Now, images and model are ready. Before we start designing our Grad-CAM class, we need to learn how to use `hook` feature in `Pytorch`.

(d) Hook

In Grad-CAM, we need i) the gradient of predicting class y^c toward the last convolution activations A_{ij}^k to calculate the importance weighting α_k^c and ii) the activation saliency maps A^k in the last convolution layer. In order to get these two components, we need to use the `hook` feature in `Pytorch`. Hooks can help us retain particualr intermediate values in forward and backward pass, since `pytorch` won't store any intermediate results on non-leaf variables. There are several good tutorals about how `hook` feature works. Learn the idea of `hook` and complete the cell below.

Problem 1 (d) Solution

```
import torch

class Hook():

  def __init__(self, ):
    self.z_grad = None
    self.y_grad = None

  # hook for z
  def z_backward_hook(self, grad):
    self.z_grad = grad ##TODO

    return None

  # hook for y
  def y_backward_hook(self, grad):
    self.y_grad = grad ##TODO
    return None

  def register(self, y, z):

    # register hooks
    ##TODO
    ##TODO
    z.register_hook(self.z_backward_hook)
    y.register_hook(self.y_backward_hook)

#-----------------Don't change anything below------------------------#

## Create matrices
v1 = torch.tensor([[1],[2]])
v2 = torch.tensor([[-2],[-1]])

## Compute the L2 norm
square_l2_v = torch.sum((v1 - v2)**2)   #  the L2 norm of v1 - v2

# Create tensors and keep track of operations on them
v1 = (v1.float()).requires_grad_()
v2 = (v2.float()).requires_grad_()

z = v1 - v2
y = (torch.norm((z),2))**2


# register hooks
hook = Hook()
hook.register(y, z)

# Perform automatic differentiation of y
y.backward()

print("The gradient w.r.t. v1 : \n" , v1.grad)
print("The gradient w.r.t. v2 : \n" , v2.grad)
print("The gradient w.r.t. z : \n", z.grad)
print("The gradient w.r.t. y : \n", y.grad)
print("The gradient w.r.t. z by hook : \n", hook.z_grad)
print("The gradient w.r.t. y by hook : \n", hook.y_grad)
```

```
The gradient w.r.t. v1 :
 tensor([[6.],
        [6.]])
The gradient w.r.t. v2 :
 tensor([[-6.],
        [-6.]])
The gradient w.r.t. z :
 None
The gradient w.r.t. y :
 None
The gradient w.r.t. z by hook :
 tensor([[6.],
        [6.]])
The gradient w.r.t. y by hook :
 tensor(1.)
/usr/local/lib/python3.7/dist-packages/torch/_tensor.py:1013: UserWarning: The .grad attribute of a Tensor that is not a leaf T
  return self._grad
```

(e) Grad-CAM class

Now we can start implementing our Grad-CAM class, and we decide to use the last convolution layer activation maps in vgg16 for our Grad-CAM. In `__init__` function, it will load a model and its target layer for Grad-CAM, and register the `hook` in the model for saving particular graidents and activation maps. In `calculate` function, it will calculate the Grad-CAM result using the formula provided in the lecture:

step1) calculate the feature importance weighting from gradients.

$$\alpha_k^c = \frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A_{ij}^k},$$

step2) calculate the rectified weighted linear combination of feature activation maps.

$$L^c_{\text{Grad-CAM}} = ReLU(\sum_k \alpha^c_k A^k).$$

Problem 1 (e) Solution

```
# Grad-CAM class implementation

import torch
import torch.nn.functional as F
from torch.autograd import Variable
from torch import nn
import numpy as np

class GradCAM():

  def __init__(self, model_object, layer_idx):
    """
    Args:
        model_object: model object that we apply Grad-CAM with
        layer_idx: index of the layer which we get activation maps

    """

    self.model = model_object

    self.gradients = dict()
    self.activations = dict()

    # backward hook for gradient
    def backward_hook(module, grad_input, grad_output):
      self.gradients['value'] = grad_output[0]
      return None

    # forward hook for activation map
    def forward_hook(module, input, output):
      self.activations['value'] = output
      return None

    target_layer = self.model.features[layer_idx] ##TODO

    # register hooks
    ##TODO
    ##TODO
    target_layer.register_backward_hook(backward_hook)
    target_layer.register_forward_hook(forward_hook)


  def calculate(self, input):
    """
    Args:
        input: input image with shape of (1, 3, H, W)

    Return:
        saliency_map: saliency map of the same spatial dimension with input
        logit: model output
    """
    b, c, h, w = input.size()
    self.model.eval()
    self.model.cuda()

    # model output
    logit = self.model(input) ##TODO

    # predicting class
    y_c = torch.max(logit) ##TODO

    self.model.zero_grad()
    y_c.backward()

    # get activation maps and gradients
    gradients = self.gradients['value']
    activations = self.activations['value']
    b, k, i, j = gradients.size()
    # print(b)
    # print(k)

    # calculate alpha (step1)
    alpha = torch.sum(gradients, dim=[2,3])/i/j ##TODO

    # calculate Grad-CAM result using rectified weighted linear combination of feature activation maps (step2)
    ##TODO
    # print(alpha.size())
    # print(activations.size())
    alpha = alpha.unsqueeze(2)
    alpha = alpha.unsqueeze(3)

    # print(alpha.size())
    # print(activations.size())

    saliency_map = torch.sum(alpha*activations, dim = 1, keepdim=True)
    saliency_map = F.relu(saliency_map)

    saliency_map = F.interpolate(saliency_map, size=(h, w), mode='bilinear', align_corners=False)
    saliency_map_min, saliency_map_max = saliency_map.min(), saliency_map.max()
    saliency_map = (saliency_map - saliency_map_min).div(saliency_map_max - saliency_map_min).data

    return saliency_map, logit

  def __call__(self, input):
    return self.calculate(input)
```

(f) Grad-CAM Visualization

Now we can apply our Grad-CAM on those images we loaded. Is there any Grad-CAM visualization reasonable with the prediction? Is there any Grad-CAM visualization not align with prediction? Why? Use the Grad-CAM visualizatioj results to answer these questions.

```
## Visualization function
```

```
import cv2
import numpy as np
import torch

def visualize_cam(mask, img):

    heatmap = cv2.applyColorMap(np.uint8(255 * mask), cv2.COLORMAP_JET)
    heatmap = heatmap.astype('float64') / 255
    heatmap = heatmap[:, :, [2, 1, 0]]

    result = heatmap + np.asarray(img).astype('float64') / 255
    result = result.astype('float64') / np.amax(result)

    return heatmap, result
```

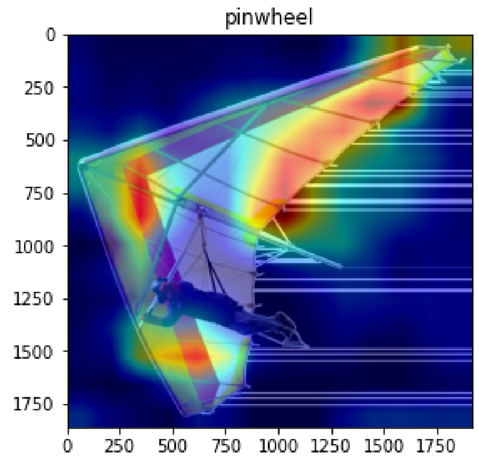
Problem 1 (f) Solution

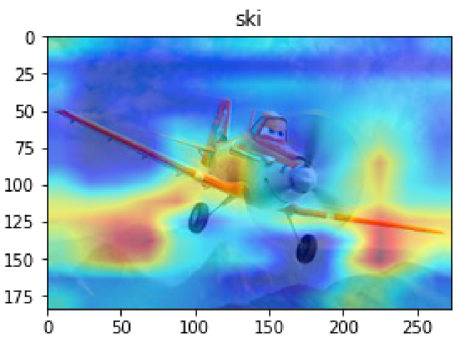
```
# create Grad-CAM object
gradcam = GradCAM(vgg16, 28)   ##TODO

# download label map
!gdown --id 1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn
label_map = torch.load("/content/label_dict.pth")

# Grad-CAM result list
gradcam_list = []

# prediction label list
pred_list = []

for batch_id, img_tensor in enumerate(my_dataloader):

  img_tensor = img_tensor.cuda()
```

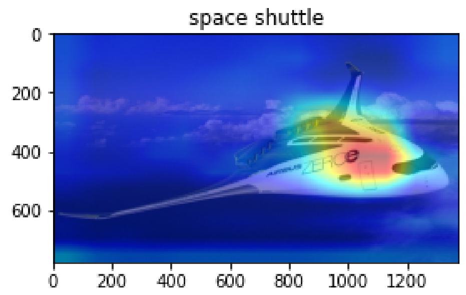
```
    mask_gradcam, logit = gradcam.calculate(img_tensor) ##TODO

    output = F.softmax(logit, dim=1)
    pred_class_idx = output.argmax(dim=1)

    pred_list.append(label_map[pred_class_idx.item()])

    # convert to original size
    mask_gradcam = F.interpolate(mask_gradcam.cpu(), np.asarray(my_dataset.img_list[batch_id]).shape[:2], mode='bilinear', align_corne

    # align heatmap with image
    heatmap, result = visualize_cam(mask_gradcam[0, 0, :, :].numpy(), my_dataset.img_list[batch_id])

    gradcam_list.append(result)
```

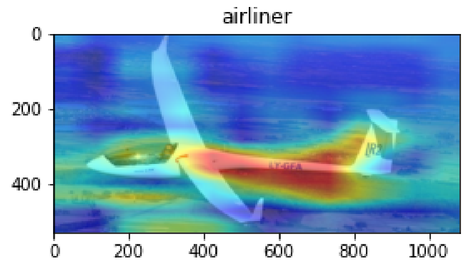
```
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1bDrtvgX-ztIh7A46FQNvROS7bEVuYogn
To: /content/label_dict.pth
100% 33.1k/33.1k [00:00<00:00, 37.4MB/s]
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py:1025: UserWarning: Using a non-full backward hook when the fo
  warnings.warn("Using a non-full backward hook when the forward contains multiple autograd Nodes "
```

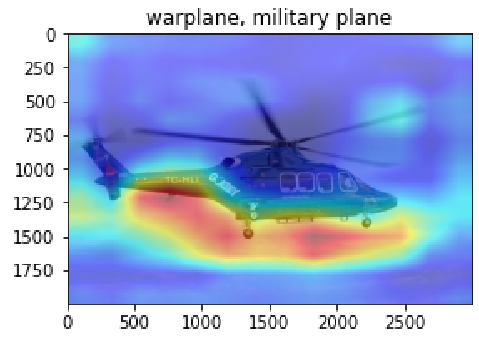
```
# visualize result
vis_img_label(gradcam_list, pred_list)
```

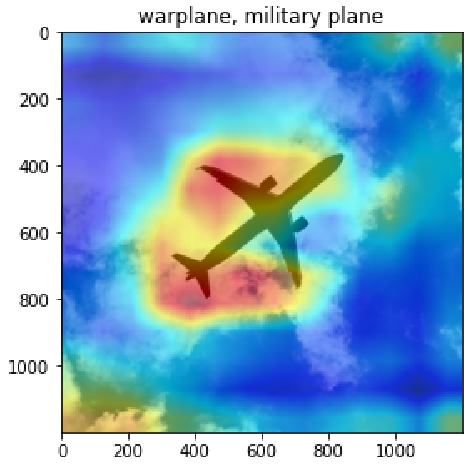

pinwheel

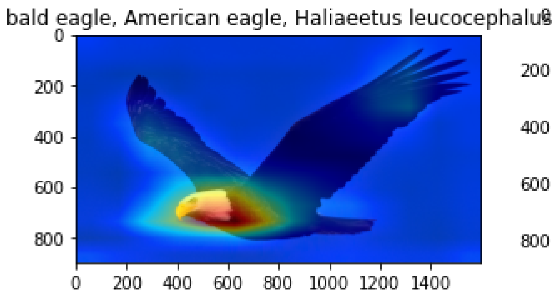

ski

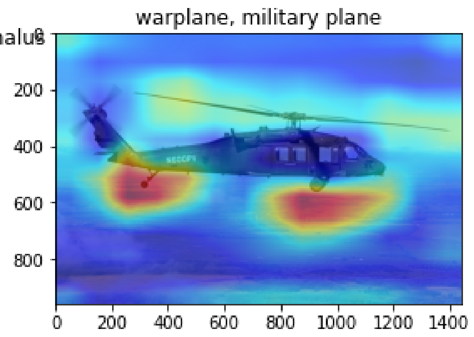

space shuttle

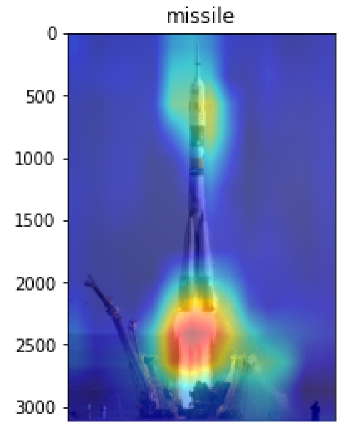

airliner

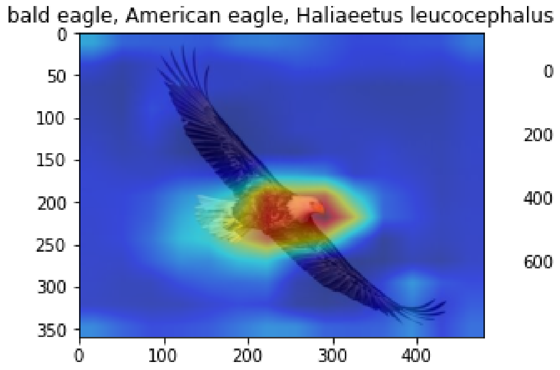

warplane, military plane

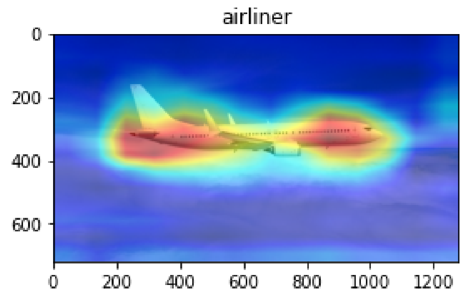

warplane, military plane

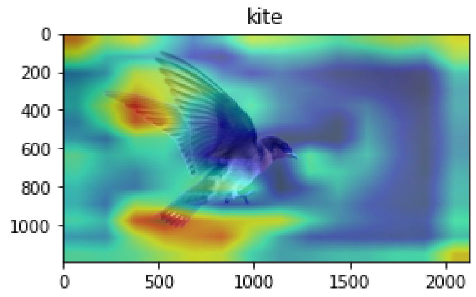

bald eagle, American eagle, Haliaeetus leucocephalus


warplane, military plane


missile


bald eagle, American eagle, Haliaeetus leucocephalus


airliner


kite

Problem 1 (f) Solution

- Most of my prediction look reasonable because the GrabCAM recognizes parts of the object such as the plane body and the head of eagle.
- The incorrect prediction will occur when the GrabCAM focus on other thing than the object itself when the algorithm mistaken the background as part of the predicted object. For example, the background of the last image of the bird have some background noise.

Problem 2: Contrast-CAM (40pts)

In Lec 23, we have learnt that Contract-CAM can help us answer a different kind of explainability question such as "Why P, rather than Q?". In this problem, we will instead use Contract-CAM to analyze examples from Imagenet.

(a) Load Imagenet Images

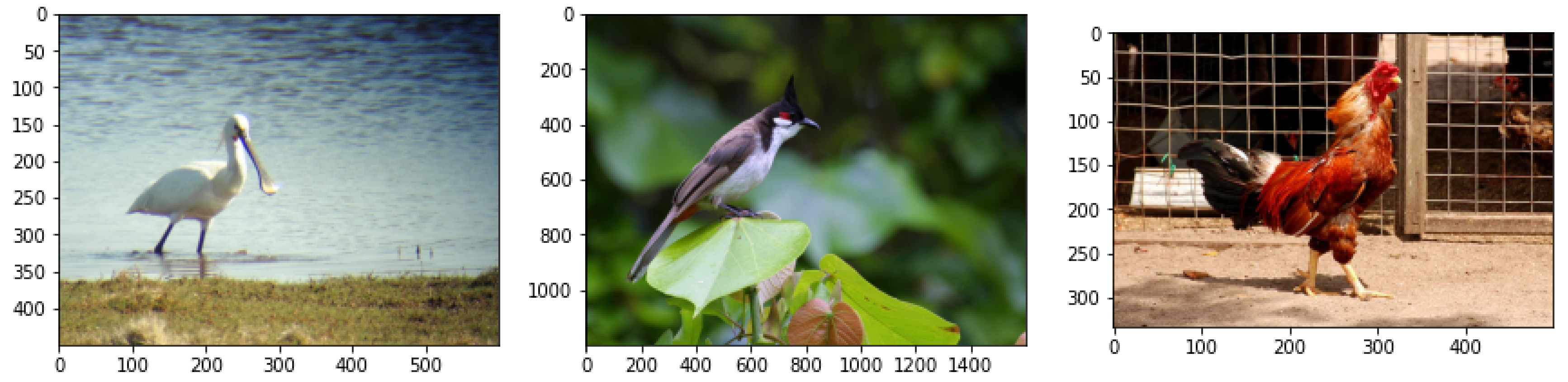
Load provided `Imagenet` images and visualize them.
Use the code in the cell below to create a folder named `Imagenet_images` and store the `Imagenet` images we provide.

Problem 2 (a) Solution

```
# create a folder named `Imagenet_images`
!mkdir Imagenet_images

# download Imagenet images
!gdown --id 196BaTh-mJqj9Y4Zn0MWWIWm9oiYnebff --output '/content/Imagenet_images/image1.jpeg'
!gdown --id 1OofwrL3xClMJukXMMpD3kLefIvPFdu10 --output '/content/Imagenet_images/image2.jpeg'
!gdown --id 1bmJ-98dSz2JTgfRseJXYcVNkec55Vng4 --output '/content/Imagenet_images/image3.jpeg'
```

```
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=196BaTh-mJqj9Y4Zn0MWWIWm9oiYnebff
To: /content/Imagenet_images/image1.jpeg
100% 96.3k/96.3k [00:00<00:00, 71.9MB/s]
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1OofwrL3xClMJukXMMpD3kLefIvPFdu10
To: /content/Imagenet_images/image2.jpeg
100% 921k/921k [00:00<00:00, 130MB/s]
/usr/local/lib/python3.7/dist-packages/gdown/cli.py:131: FutureWarning: Option `--id` was deprecated in version 4.3.1 and will
  category=FutureWarning,
Downloading...
From: https://drive.google.com/uc?id=1bmJ-98dSz2JTgfRseJXYcVNkec55Vng4
To: /content/Imagenet_images/image3.jpeg
100% 131k/131k [00:00<00:00, 78.6MB/s]
```

```
# visualize images

from PIL import Image
import matplotlib.pyplot as plt
from glob import glob
import os
import numpy as np

def imread(img_dir):
  # read the images into a list of `PIL.Image` objects
  images = []
  for f in sorted(glob(os.path.join(img_dir, "*"))):
    images.append(Image.open(f).convert('RGB'))

  return images

def vis_img_label(image_list, label_list=None):
  # visualize the images w/ labels
  Tot = len(image_list)
  Cols = 4
  Rows = Tot // Cols
  Rows += (Tot % Cols)>0
  if label_list is None:
    label_list = [""]*Tot
  # Create a Position index
  Position = range(1,Tot + 1)
  fig = plt.figure(figsize=(Cols*5, Rows*5))
  for i in range(Tot):
    image = image_list[i]
    # add every single subplot to the figure
    ax = fig.add_subplot(Rows,Cols,Position[i])
    ax.imshow(np.asarray(image))
    ax.set_title(label_list[i])


## Load Imagenet images
img_dir = "/content/Imagenet_images"
Imagenet_image_list = imread(img_dir)
## visualize Imagenet images
vis_img_label(Imagenet_image_list)
```



(b) Create Imagenet Dataset

Create the Imagenet dataset by re-using `myDataset`.

Problem 2 (b) Solution

```
# reuse myDataset class
Imagenet_dataset = myDataset(Imagenet_image_list, data_transform) ##TODO
Imagenet_dataloader = DataLoader(Imagenet_dataset, batch_size=1, shuffle=False, num_workers=2)

#----verify the Imagenet dataset `myDataset`----#
print('The size of the dataset: ', len(Imagenet_dataset))
print('The dimension of the first image sample after transformations: ', Imagenet_dataset[0].shape)
```

```
The size of the dataset:  3
The dimension of the first image sample after transformations:  torch.Size([3, 224, 224])
```

(c) Contrast-CAM class

Now, we are going to implement Contrast-CAM class. The only difference between Grad-CAM and Contrast-CAM is that Contrast-CAM utilizes the loss between predicted class P and some contrast class Q to get gradients:

$$\alpha_k^c = \frac{1}{Z}\sum_i\sum_j \frac{\partial J(P,Q)}{\partial A_{ij}^k}.$$

Hence, you can reuse most of your implementation from problem 1 (e) to complete this part.

Problem 2 (c) Solution

```
# Contrast-CAM class implementation

import torch
import torch.nn.functional as F
from torch.autograd import Variable
from torch import nn
import numpy as np

class ContrastCAM():

  def __init__(self, model_object, layer_idx):
    """
    Args:
        model_object: model object that we apply Grad-CAM with
        layer_idx: index of the layer which we get activation maps

    """

    self.model = model_object

    self.gradients = dict()
    self.activations = dict()

    # backward hook for gradient
    def backward_hook(module, grad_input, grad_output):
      self.gradients['value'] = grad_output[0]
      return None

    # forward hook for activation map
    def forward_hook(module, input, output):
      self.activations['value'] = output
      return None

    target_layer = self.model.features[layer_idx] ##TODO

    # register hooks
    ##TODO
    ##TODO
    target_layer.register_backward_hook(backward_hook)
    target_layer.register_forward_hook(forward_hook)

  def calculate(self, input, Q):
    """
    Args:
        input: input image with shape of (1, 3, H, W)
        Q: integer, class index for calculating Contrast-CAM

    Return:
        saliency_map: saliency map of the same spatial dimension with input
        logit: model output
    """
    b, c, h, w = input.size()
    self.model.eval()
    self.model.cuda()

    # model output
    logit = self.model(input) ##TODO
    # print(logit)

    # predicting class
    y_c = torch.max(logit) ##TODO

    # cross entropy loss function
    ce_loss = torch.nn.CrossEntropyLoss()  ##TODO
    # ce_loss.cuda()

    # calculate loss
    ##TODO
    # print(logit.size())
    Q_arr = torch.zeros(logit.size()[0], logit.size()[1])
    Q_arr[0,Q] = 1
    Q_arr.to('cuda')

    # print(Q_arr.size())
    # print(Q_arr)
    pred_loss = ce_loss(logit.cuda(), Q_arr.cuda())
    # print(pred_loss.shape)
    # pred_loss.cuda()

    self.model.zero_grad()
    pred_loss.backward()

    # get activation maps and gradients
    gradients = self.gradients['value']
    activations = self.activations['value']
    b, k, i, j = gradients.size()

    # calculate alpha (step1)
    alpha = torch.sum(gradients, dim=[2,3])/i/j ##TODO

    alpha = alpha.unsqueeze(2)
    alpha = alpha.unsqueeze(3)

    # calculate Grad-CAM result using rectified weighted linear combination of feature activation maps (step2)
    ##TODO
    saliency_map = torch.sum(alpha*activations, dim = 1, keepdim=True)
    saliency_map = F.relu(saliency_map)

    saliency_map = F.interpolate(saliency_map, size=(h, w), mode='bilinear', align_corners=False)
    saliency_map_min, saliency_map_max = saliency_map.min(), saliency_map.max()
    saliency_map = (saliency_map - saliency_map_min).div(saliency_map_max - saliency_map_min).data

    return saliency_map, logit

  def __call__(self, input, Q):
    return self.calculate(input, Q)
```

(d) Contrast-CAM: Why P, rather than Q?

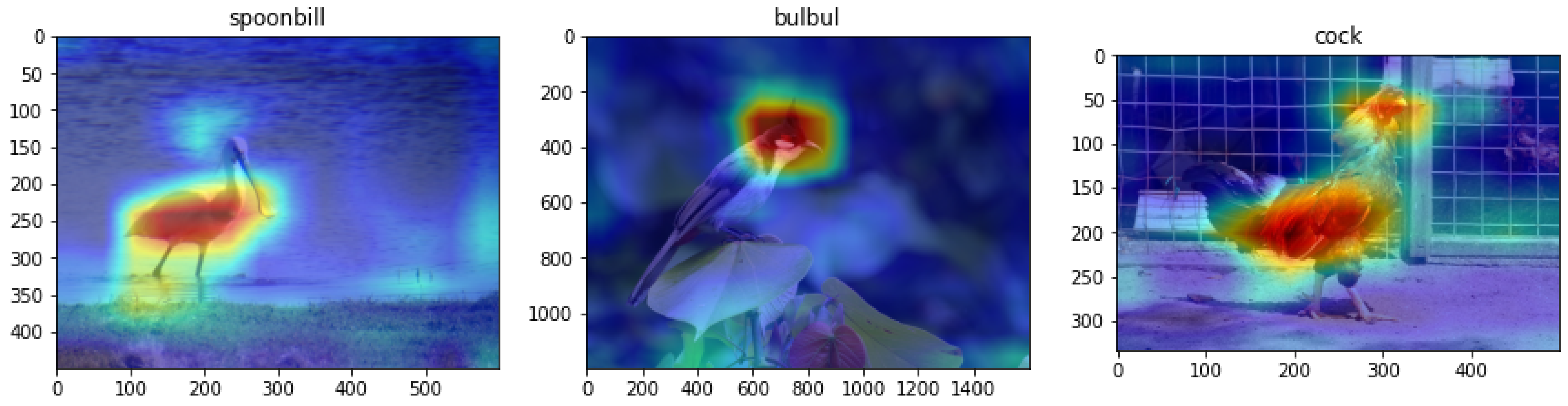
Use the Contrast-CAM visualization result to give your explaination for the "Why P, rather than Q?" question below. Suppose Q is flamingo bird here.

Problem 2 (d) Solution

```
# Why P, rather than Q?

# create Contrast-CAM object
contrastcam = ContrastCAM(vgg16, 28) ##TODO

# Contrast-CAM result list
contrastcam_list = []

# prediction label list
imagenet_pred_list = []

# Q list
Q_list = [130, 130, 130]

# Q_list = torch.Tensor(Q_list)
# Q_list = Q_list.type(torch.int)

# Q_list = Q_list.to('cuda')

for batch_id, img_tensor in enumerate(Imagenet_dataloader):

  img_tensor = img_tensor.cuda()
  # ind = Q_list[batch_id]
  # print(ind.item())
  # ind = ind.to('cuda')
  mask_contrastcam, logit = contrastcam.calculate(img_tensor, Q_list[batch_id]) ##TODO

  output = F.softmax(logit, dim=1)
  pred_class_idx = output.argmax(dim=1)

  imagenet_pred_list.append(label_map[pred_class_idx.item()])

  # convert to original size
  mask_contrastcam = F.interpolate(mask_contrastcam.cpu(), np.asarray(Imagenet_dataset.img_list[batch_id]).shape[:2], mode='bilinear

  # align heatmap with image
  heatmap, result = visualize_cam(mask_contrastcam[0, 0, :, :].numpy(), Imagenet_dataset.img_list[batch_id])

  contrastcam_list.append(result)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py:1025: UserWarning: Using a non-full backward hook when the fo
  warnings.warn("Using a non-full backward hook when the forward contains multiple autograd Nodes "
```

```
# visualize result
vis_img_label(contrastcam_list, imagenet_pred_list)
```



According to the Contrast-CAM result, the bulbul bird is not classified as flamingo because of its head shape. Spoonbill is not classified as flamingo because of its body part. Cock is not classified as flamingo because of its head and body.

(e) Contrast-CAM: Why P, rather than P?

Use the Contrast-CAM visualization result to give your explaination for the "Why P, rather than P?" question below.

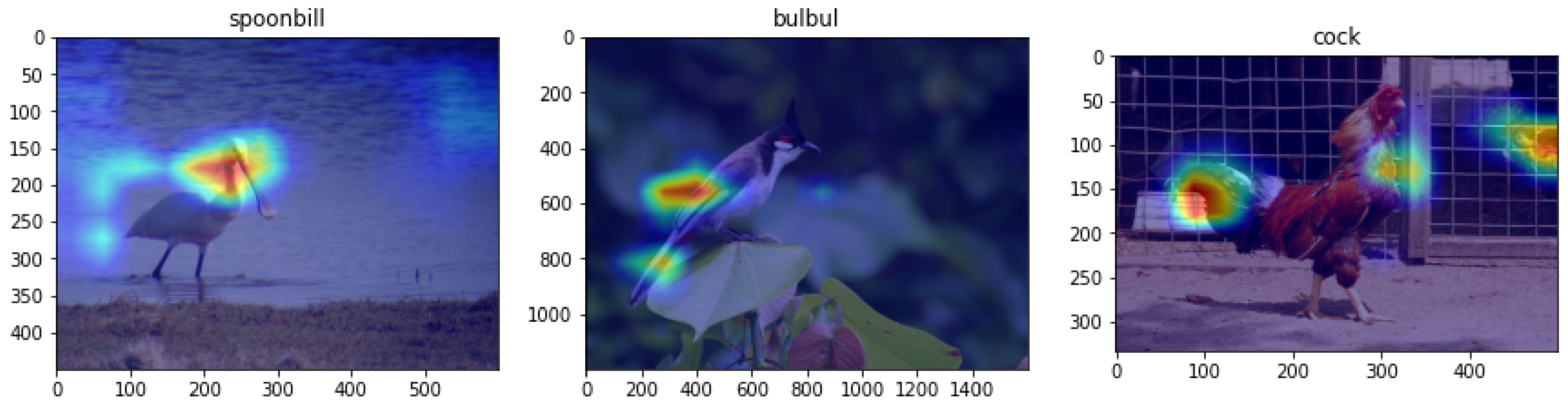
Problem 2 (e) Solution

```
# Why P, rather than P?

# Contrast-CAM result list
contrastcam_list = []

# P list
P_list = [129, 16, 7]

for batch_id, img_tensor in enumerate(Imagenet_dataloader):

  img_tensor = img_tensor.cuda()
  mask_contrastcam, logit = contrastcam.calculate(img_tensor, P_list[batch_id]) ##TODO

  output = F.softmax(logit, dim=1)
  pred_class_idx = output.argmax(dim=1)

  pred_list.append(label_map[pred_class_idx.item()])

  # convert to original size
  mask_contrastcam = F.interpolate(mask_contrastcam.cpu(), np.asarray(Imagenet_dataset.img_list[batch_id]).shape[:2], mode='bilinear

  # align heatmap with image
  heatmap, result = visualize_cam(mask_contrastcam[0, 0, :, :].numpy(), Imagenet_dataset.img_list[batch_id])

  contrastcam_list.append(result)
```

```
    /usr/local/lib/python3.7/dist-packages/torch/nn/modules/module.py:1025: UserWarning: Using a non-full backward hook when the fc
```

```
  warnings.warn("Using a non-full backward hook when the forward contains multiple autograd Nodes "
```

```
# visualize result
vis_img_label(contrastcam_list, imagenet_pred_list)
```



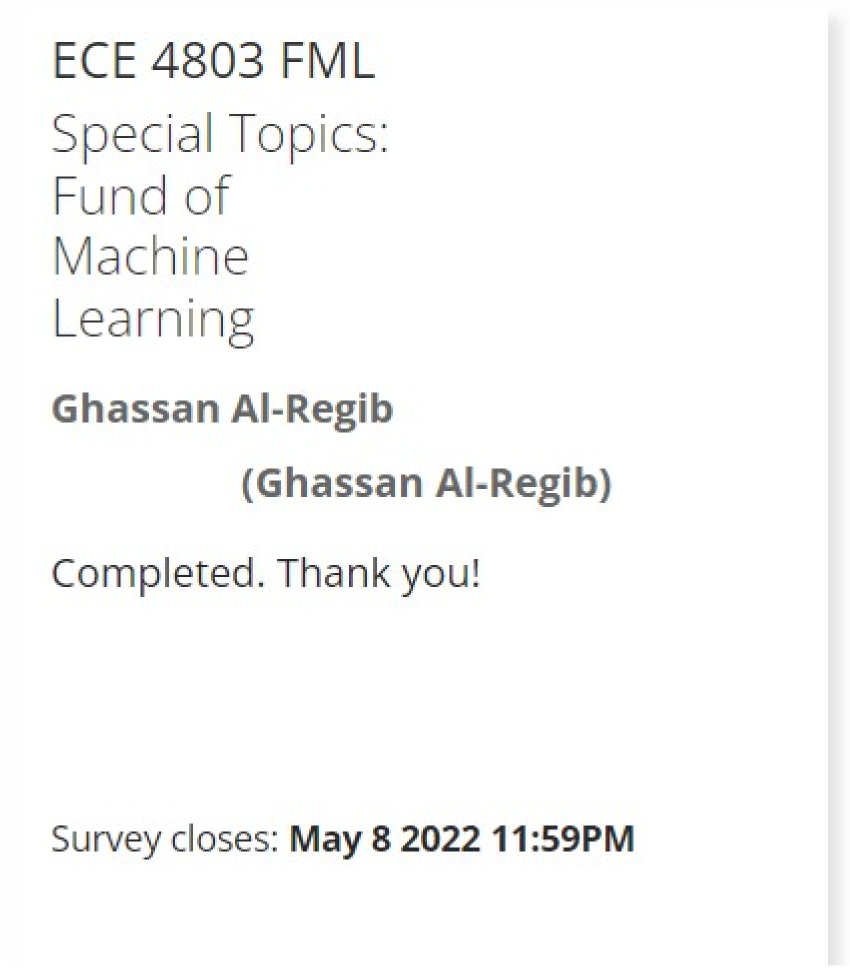
- Backpropagating this loss highlights confusing regions for a network from.
- From the result above, we can see that the ContrastCAM is not 100% confidence because of the head of spoonbill, the tail of bulbul, and the tail and background noise of the cock picture.

Problem 3: CIOS Survey (Bonus)

If you include a proof (e.g., a screenshot without revealing your choices) that you completed the CIOS survey for the course, you will receive a bonus of 1% for the overall grade. This bonus opportunity will be available with your take-home final exam as well in case you do not have time to complete the survey before the due date for hw#7. You will receive the bonus points once, either in hw#7 or in the final exam. More than that, if 90% of the class submits, everyone gets another 1%.

Problem 3 Solution

This is my screenshot:



ECE 4803 FML

Special Topics: Fund of Machine Learning

Ghassan Al-Regib

(Ghassan Al-Regib)

Completed. Thank you!

Survey closes: **May 8 2022 11:59PM**

✓ 1s completed at 12:58 PM