ENGN8536 CLab2 Report

Shreya Chawla (u7195872)

2 Task 1: Implement the Class Activation Mapping

2.1 Build the CAM Model

The code for CAM Model is in Appendix A (model.py). The backbone is pre-trained ResNet-18 model upto Layer2 - index 0 block (the first block in Layer 2) using pytorch torchvision. Global average pooling is initialized by nn.AdaptiveAvgPool2d function by passing output_size as (1,1). The classification layer W_{CLS} is a linear layer mapping dim128 input to dim10 to get classification results.
A set of large images is passed through backbone to get feature map (f). The size of all 128 feature map (f_i) is ensured to be (28, 28) using an assert statement. This feature map is then passed to GAP to get the weighting coefficient (w). This coefficient (w) is then flattened and passed to the linear classification layer to get classification scores.

2.2 Train the CAM model

1. The code for CAM Model training is in Appendix B (train.py). The results are in Table 1 below. An improvement in accuracy of about 2.6% is observed when the batch size was reduced from 64 to 16. To train, the classification scores are passed through the cross entropy loss criterion. For all future tasks a batch size of 64 and learning rate of 0.001 are used as they gave the best results under these conditions. The value of the rest of the hyperparameters is set as default.

Table 1: Classification performance of the model in CAM mode

Epochs trained for	Batch size	Learning rate	**Testing Accuracy**	Achieved at epoch:
10	64	0.0001	0.692	10
10	16	0.0001	0.718	10
10	64	0.001	0.760	10

2. The code for this part is in Appendix B (train.py) under the train_CV function where the train data is split into 5 folds and of these 5 models trained, the best model is tested on the test set. The script main.py (Appendix C) was modified to call this function and also an argument in parse_args function called “cv” was added. It uses sklearn library to get 5 folds of the data and then splits the images and their labels accordingly. Then, the original train method is called in a loop for each train-validation set. The model with best accuracy on the validation set is checkpointed. The best model is tested on the test set.

3 Task 2: Weakly Supervised Semantic Segmentation with Class Activation Maps

3.1 Complete the forward path for SEG mode

The code for forward path for SEG mode is in Appendix A (model.py) under the if self.args.mode==’SEG’ statement. Like in the earlier CAM mode, the images of original or large size are forwarded through the network following Backbone → GAP → W_{CLS}. Similarly if img_S is given then it is also fed to the model similarly. If the img_S is provided then it returns the classification score, feature map and the weighing coefficient for both large and small images, else only for the large image.

3.2 Implement the scale equivariant loss

Classification losses (L^{L}_{cls} and L^{S}_{cls}) are computed using the cross entropy loss function used also in CAM. Next, the feature maps and weighting coefficients for both large and small images along with “args” are fed to a helper function “returnCAM_SEG” in train.py which is similar to the “returnCAM” function of visualize.py script (Appendix E).

In “returnCAM_SEG" function, the features from L^L are downscaled to half the size by feeding a scale factor of 0.5 torch.nn.functional.interpolate function in bilinear mode. Softmax of the weighting coefficients are taken using torch.softmax function with dim=1. The CAMs for I^L and I^S are calculated for each batch of images by summing over product of weighting coefficient and reshaped feature maps along dim=1 which is then reshaped into shape of each feature map. These CAMs are then returned.
The segmentation loss (L_{seg}) is calculated using MSELoss in torch.nn. Final loss is calculated as: $(L^L_{cls}+L^S_{cls})/2 + L_{seg}$

3.3 Train the CAM model in SEG mode

The CAM model is trained in SEG mode. The accuracy observed in SEG mode is slightly more than that in CAM mode. This can be attributed to the fundamental difference between them in their model. The accuracy achieved for the best set of hyperparameters (from Task 2.2.1) based on limited experiement is given in Table 2.

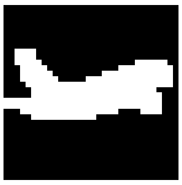
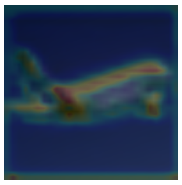
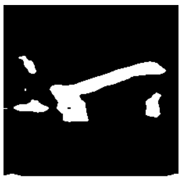
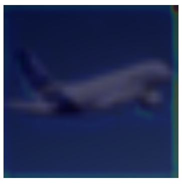
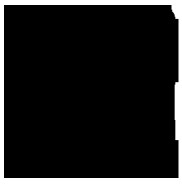
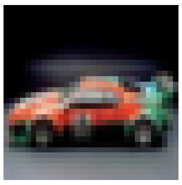
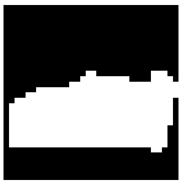
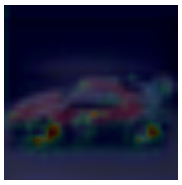
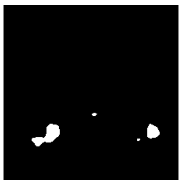
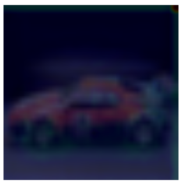
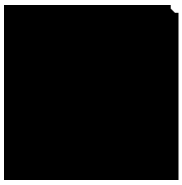
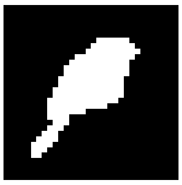
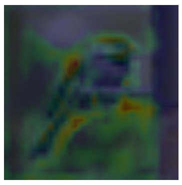
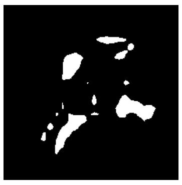
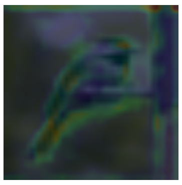
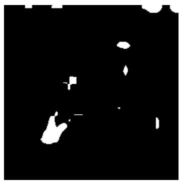
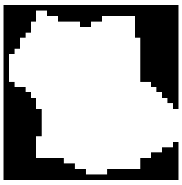
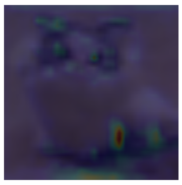
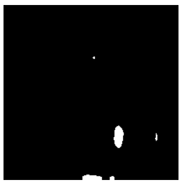
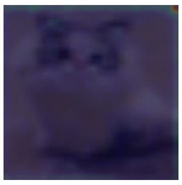
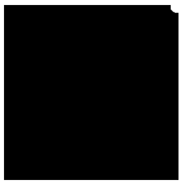
Table 2: Classification performance of the model in SEG mode with the best set of hyperparameters used to train CAM

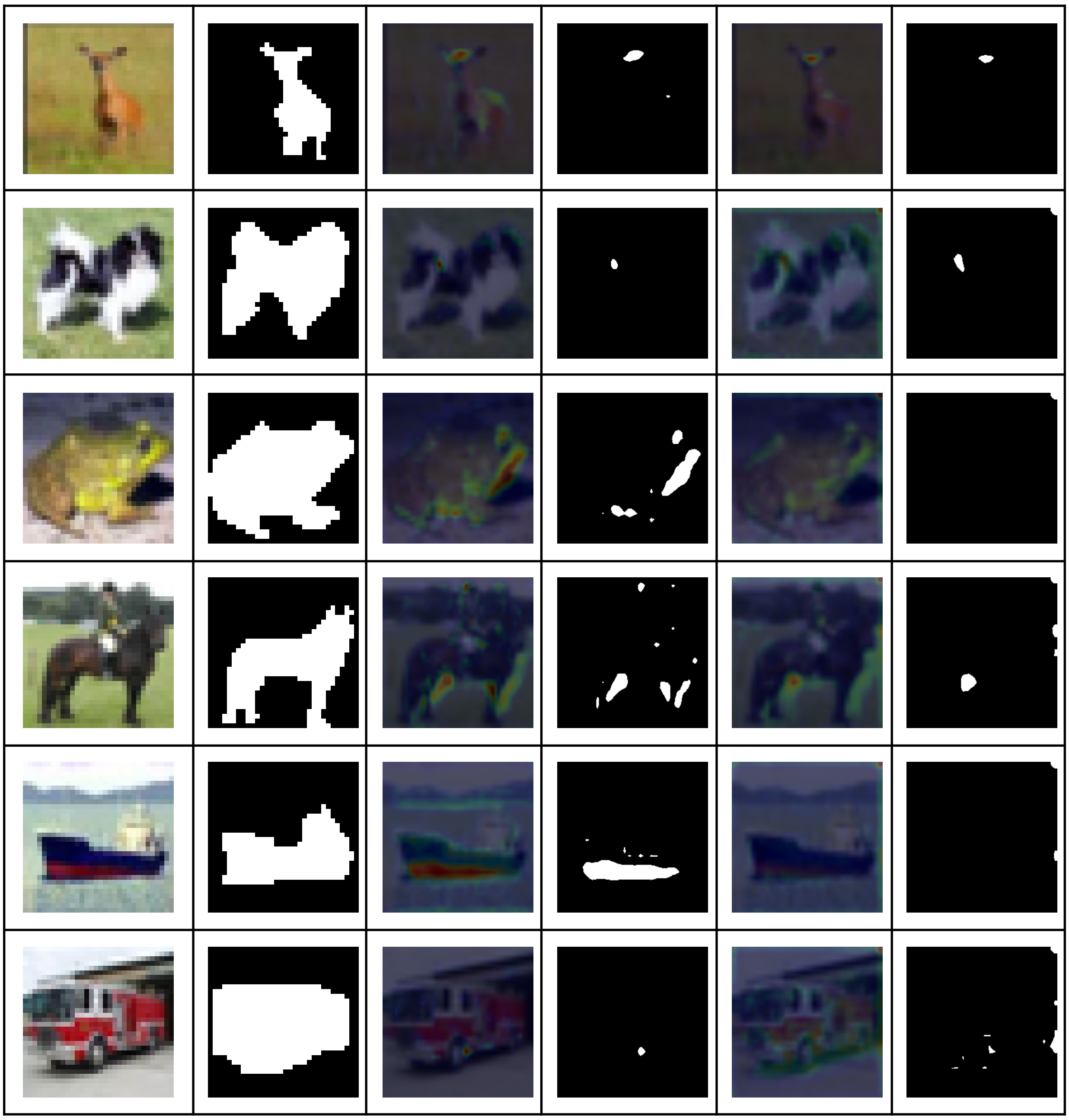
Epochs trained for	Batch size	Learning rate	**Testing Accuracy**	Achieved at epoch:
10	64	0.001	0.789	8

3.4 Visualization and IoU

When ‘visualize.py’ is run in ‘CAM’ and ‘SEG’ mode we get the following results as shown in Table 3. These results are obtained for a threshold of 125. The visualize.py script (Appendix E) was changed to accept the threshold value and the location of the source and ground truth were changed. Also, in the returnCAM function, little changes were made for correct processing of CAMs. It is observed that in general, the result of SEG mode is poorer than CAM. The reason for this is discussed after stating the IoU scores.

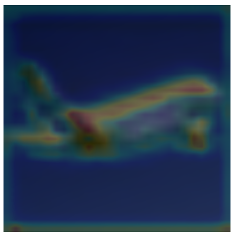
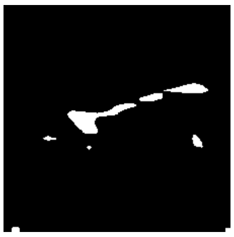
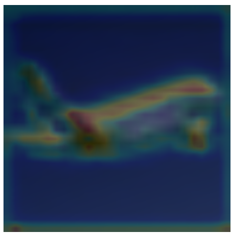
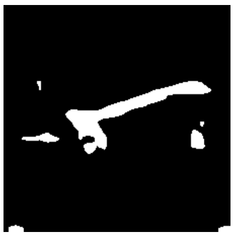
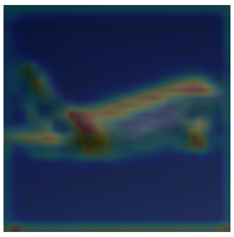
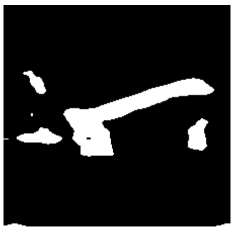
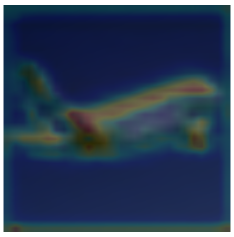
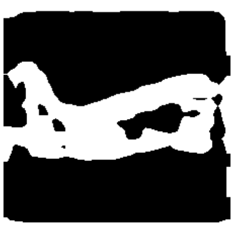
Table 3: Qualitative results for both CAM and SEG mode of the model for threshold = 125

Source	SEG GT	CAM CAM	CAM SEG	SEG CAM	SEG SEG
					
					
					
					



It was observed that for high thresholds (e.g. 175), the CAM SEG and SEG SEG were both very sparse with little or no white spots. Hence, the threshold was lowered. Also, it was observed that as the threshold was lowered (e.g. 75), the segmentation became noisier which segmented parts of the image that did not correspond to their class. Hence, a threshold of 100 is used for all experiments. These results are displayed in Table 4 below. It is observed that there are more activation regions in CAMs relevant to the image than in CAM with SEG. Performing data augmentations on the image before training on it makes the model learn better representations and patterns. Although the classification accuracy for test set under same conditions was higher for SEG, CAM mode is able to generate better segmentations.

Table 4: Comparison of CAMs and SEGs in CAM and SEG mode for different thresholds for class = “plane”

Threshold	CAM CAM	CAM SEG	SEG CAM	SEG SEG
175				
150				
125				
75				

Intersection over Union (**IoU**) is an evaluation metric for segmentation and detection tasks. It takes the common part of the segmented image and GT image, called intersection, over the union of the area in the 2 images. The union works like a binary logical ‘OR’ whereas the intersection is analogous to the binary local operation ‘AND’. The iou.py script implements these concepts using the numpy library’s bitwise operators (Appendix F). These IoU scores are recorded in table 5 below for different threshold values. As was observed in the visualizations of Table 3, the IoU scores are very low for high thresholds and as it is lowered, the IoU value improves. But for some classes this score is still 0. However, IoU does not always indicate the effectiveness of the model and CAM. The visualizations above show that for threshold of 75, the results are very noisy. It is also observed that one threshold does not give best results for all classes, and neither for both the modes. In CAM, the higher threshold gave better IoU for two classes (dog, horse) but for SEG, the IoU is highest for all classes for a low threshold of 75.The IoU for CAM is generally better than SEG for all classes. This could be attributed to different thresholds giving better results for different classes and modes, and could be improved by improving the generalizability of the model for SEG mode as will be seen in 3.5 task. This could also be attributed to the hyperparameters, As the best set of hyperparameters in CAM mode were used for SEG mode as well for a comparative study, Hence, the effect on SEG model by change in hyperparameters was not experimented on. This change could probably enhance its pattern recognition capabilities.

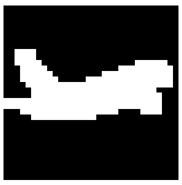
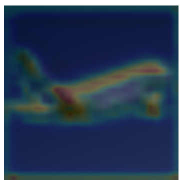
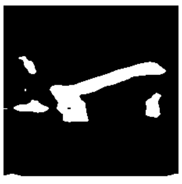
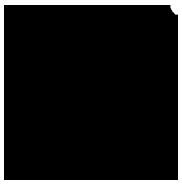
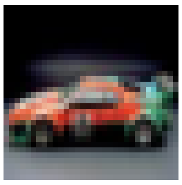
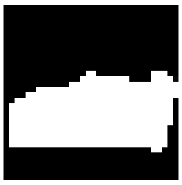
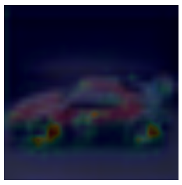
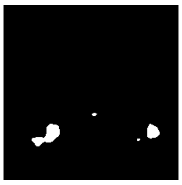
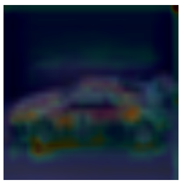
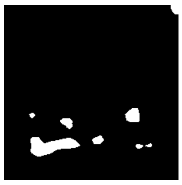
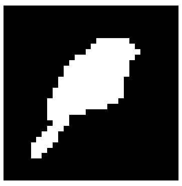
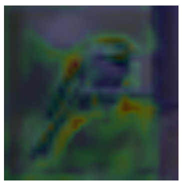
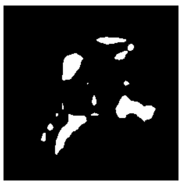
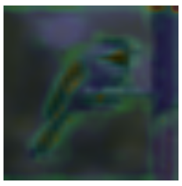
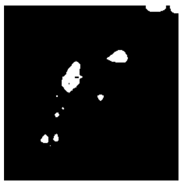
Table 5: IoU metric results for threshold of 175, 150, 125 and 75

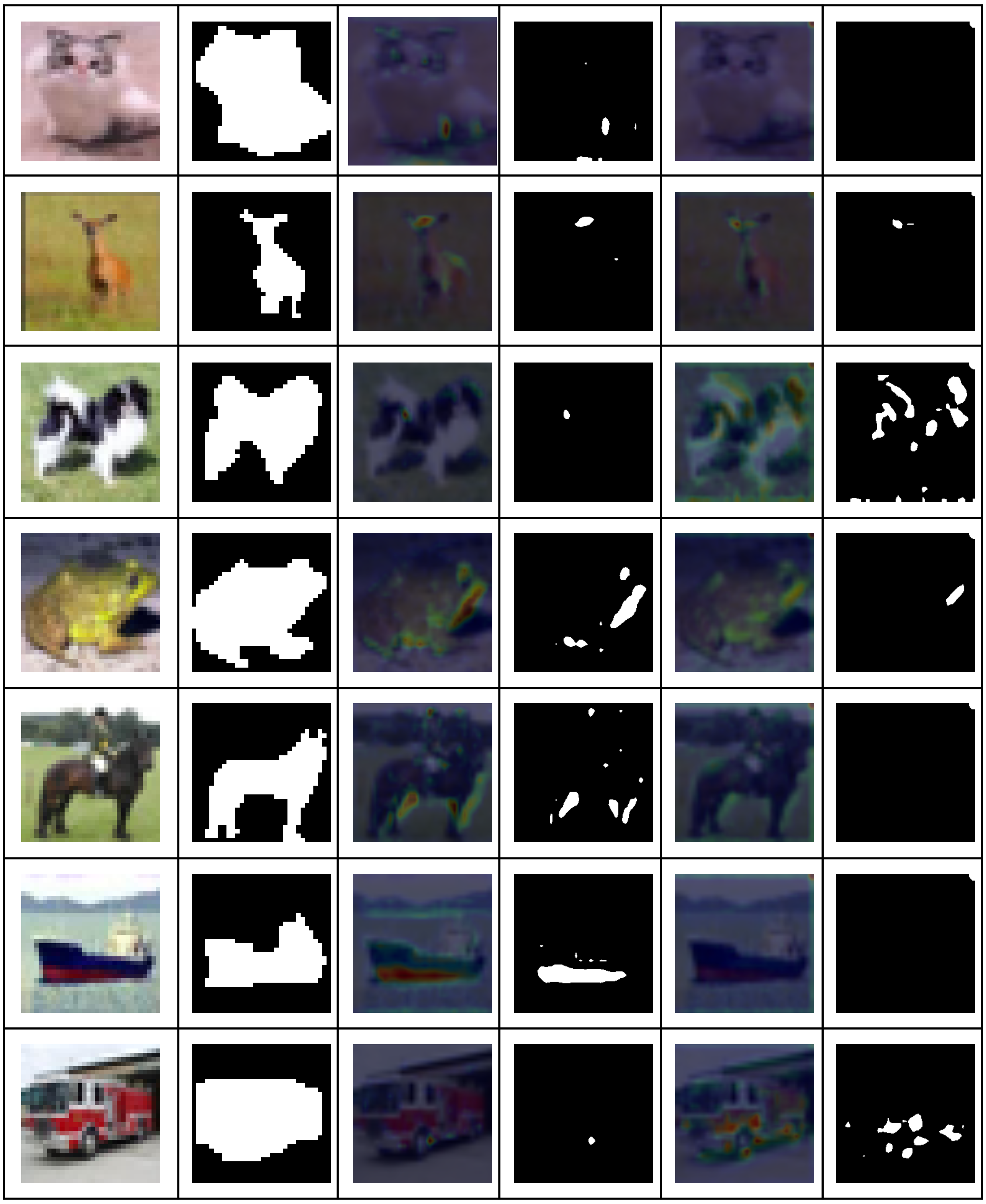
	175		150		125		75	
Class	CAM	SEG	CAM	SEG	CAM	SEG	CAM	SEG
plane	0.000	0.000	0.215	0.000	0.325	0.000	**0.524**	0.000
car	0.000	0.000	0.025	0.000	0.042	0.000	**0.162**	**0.008**
bird	0.248	0.000	0.052	0.027	0.107	0.080	**0.275**	**0.345**
cat	0.000	0.000	0.006	0.000	0.014	0.000	**0.050**	**0.006**
deer	0.030	0.012	0.036	0.012	0.035	0.018	**0.113**	**0.053**
dog	**0.052**	0.000	0.002	0.000	0.007	0.011	0.034	**0.133**
frog	0.002	0.000	0.038	0.000	0.065	0.000	**0.185**	**0.021**
horse	0.013	0.000	**0.014**	0.000	0.033	0.000	0.095	**0.037**
ship	0.000	0.000	0.168	0.000	0.224	0.000	**0.466**	**0.025**
truck	**0.110**	0.000	0.002	0.000	0.006	0.004	0.013	**0.327**

3.5 Explore other transformations

1. The linear transformation of vertical flipping was performed on the data in dataloader.py (Appendix D). The results obtained are as below in Table 6.
   In SEG mode, the segmentation has improved drastically showing that it has hugely benefited by the data augmentation for many classes like car, dog, and truck. However, for some other classes, there is still room for improvement.
   In SEG mode with augmentation of flipping the image along vertical axis (aka mirroring) we get better results than CAM mode for car, dog, and truck. This shows the potential of this algorithm to learn. Perhaps training for more epochs or choosing a different threshold (like 75 - as seen in previous question) might give even better segmentations.

Table 6: Results after training model in SEG mode at threshold of 125 after vertical flip linear transformation

Source	SEG GT	CAM CAM	CAM SEG	SEG CAM	SEG SEG
					
					
					

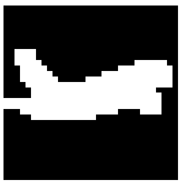
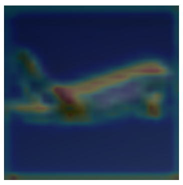
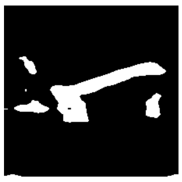
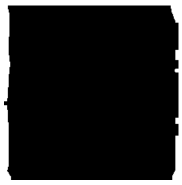
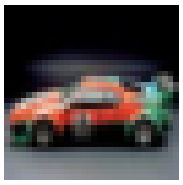
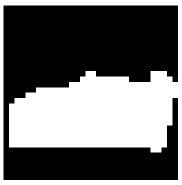
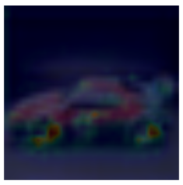
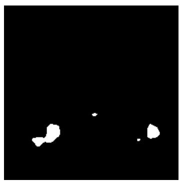
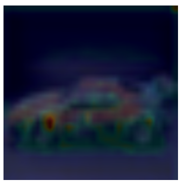
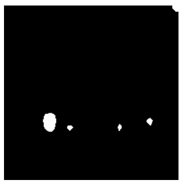
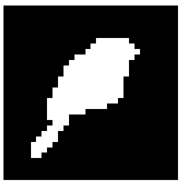
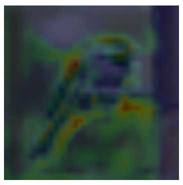
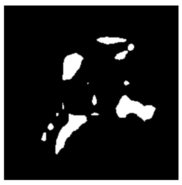
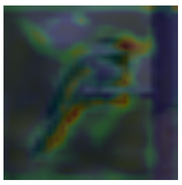
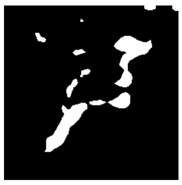
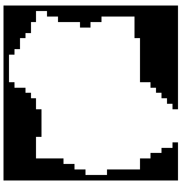
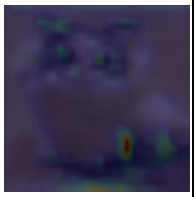
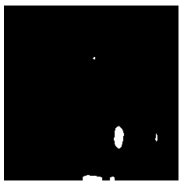
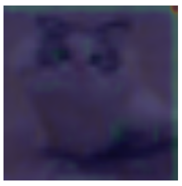
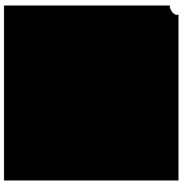
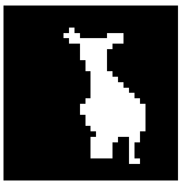
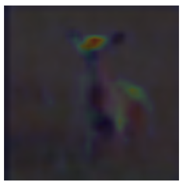
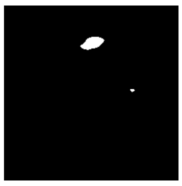
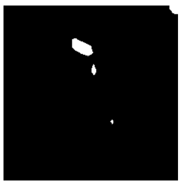
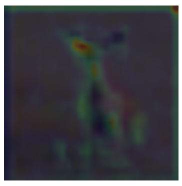
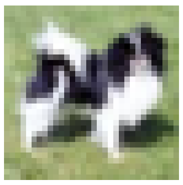
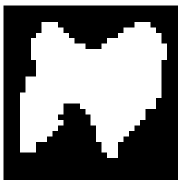
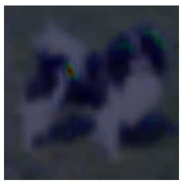
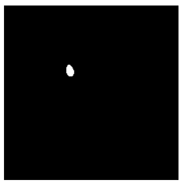
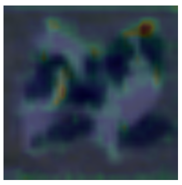
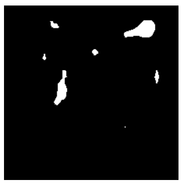
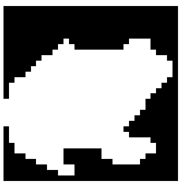
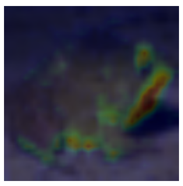
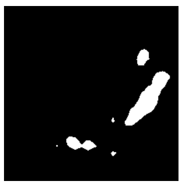
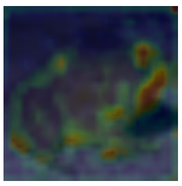
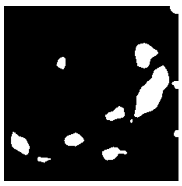


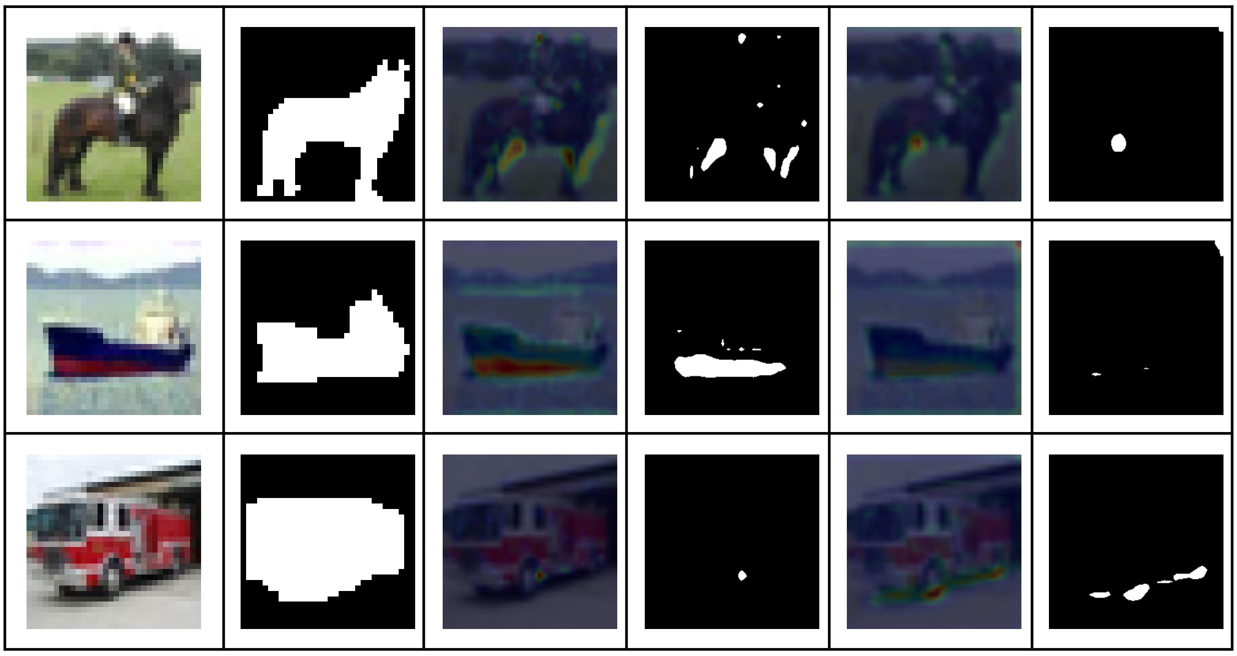
The IoU scores are as below:
Class: plane | CAM IoU: 0.325 | SEG IoU: 0.000
Class: car | CAM IoU: 0.042 | SEG IoU: 0.068
Class: bird | CAM IoU: 0.107 | SEG IoU: 0.126
Class: cat | CAM IoU: 0.014 | SEG IoU: 0.000
Class: deer | CAM IoU: 0.035 | SEG IoU: 0.023
Class: dog | CAM IoU: 0.007 | SEG IoU: 0.136
Class: frog | CAM IoU: 0.065 | SEG IoU: 0.018
Class: horse | CAM IoU: 0.033 | SEG IoU: 0.000
Class: ship | CAM IoU: 0.224 | SEG IoU: 0.000
Class: truck | CAM IoU: 0.006 | SEG IoU: 0.090

Compared to the previous scores, the IoU score has also improved. For example, for class "car", the IoU score was 0.000 without augmentation but with augmentation, it spiked to 0.068. Similarly for the bird, deer, dog, frog and truck. Hence, the qualitative and quantitative results show that data augmentation is very helpful in improving results.

2. The linear transformation of vertical flipping followed by horizontal flipping was performed on the data in dataloader.py (Appendix D). The results obtained are as below in Table 7. The code for these is in main.py and dataloader.py.

Table 7: Results after training model in SEG mode at threshold of 125 after vertical and horizontal flip linear transformation during training

Source	SEG GT	CAM CAM	CAM SEG	SEG CAM	SEG SEG
					
					
					
					
					
					
					



The IoU scores are as below:

Class: plane | CAM IoU: 0.325 | SEG IoU: 0.007
Class: car | CAM IoU: 0.042 | SEG IoU: 0.028
Class: bird | CAM IoU: 0.107 | SEG IoU: 0.200
Class: cat | CAM IoU: 0.014 | SEG IoU: 0.000
Class: deer | CAM IoU: 0.035 | SEG IoU: 0.040
Class: dog | CAM IoU: 0.007 | SEG IoU: 0.045
Class: frog | CAM IoU: 0.065 | SEG IoU: 0.127
Class: horse | CAM IoU: 0.033 | SEG IoU: 0.009
Class: ship | CAM IoU: 0.224 | SEG IoU: 0.010
Class: truck | CAM IoU: 0.006 | SEG IoU: 0.036

Similar to the results obtained for 3.5.1 task, the results both quantitatively an qualitatively have improved in SEG mode after data augmentation during training time. For the classes - plane, car, bird, deer, dog, frog, horse, ship, and truck. Upon comparing results of 3.5.1 with these results, we can see that after a series of linear transformation the segmentation and IoU has further improved compared to a single data transformation only. Thus, the model is forced to learn different representations of the same image by augmenting it. These provide more examples to our model for better generalizability. These results are comparable and even better than CAM mode for some classes (e.g. bird). Hence, we could conclude that in SEG mode, the model can generalize better than in CAM mode after a series of data augmentations.

Appendix A – “model.py”

```
import torch
import torch.nn as nn
import torchvision
from collections import OrderedDict


class CAMModel(nn.Module):
    def __init__(self, args):
        super(CAMModel, self).__init__()
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.args = args

        resnet18 = torchvision.models.resnet18(pretrained=True)
        layer2 = [module for module in resnet18.layer2.modules() if not
isinstance(module, nn.Sequential)]
        self.backbone = nn.Sequential(OrderedDict([
            ('conv1', resnet18.conv1),
            ('bn1', resnet18.bn1),
            ('relu', resnet18.relu),
            ('maxpool', resnet18.maxpool),
            ('layer1', resnet18.layer1),
            ('layer2', layer2[0])]))

        self.gap = nn.AdaptiveMaxPool2d((1, 1))
        self.cls = nn.Linear(128, 10)

    def forward(self, img_L, img_S=None):

        if self.args.mode == 'CAM':
            # FORWARD PATH FOR CAM
            f = self.backbone(img_L)
            assert list(f.shape)[1:] == [128, 28, 28]  # Sanity check
            w = self.gap(f)
            w_flat = torch.flatten(w, 1)
            out = self.cls(w_flat)
            return out, f, w_flat

        elif self.args.mode == 'SEG':
            # FORWARD PATH FOR SEG
            f_l = self.backbone(img_L)
            w_l = self.gap(f_l)
            w_l_flat = torch.flatten(w_l, 1)
            out_l = self.cls(w_l_flat)
            if img_S is not None:
                f_s = self.backbone(img_S)
                assert list(f_s.shape)[-2:] == [14, 14]  # Sanity check
                w_s = self.gap(f_s)
                w_s_flat = torch.flatten(w_s, 1)
                out_s = self.cls(w_s_flat)
                return out_l, f_l, w_l_flat, out_s, f_s, w_s_flat
            return out_l, f_l, w_l_flat

        else:
            NotImplementedError
```

Appendix B – “train.py”

```
import os
import PIL.Image as Image

import torch
from torch.utils.data import Dataset
from torchvision import transforms

ANNOTATION_PATH_TRAIN = 'engn8536/Datasets/cifar-10-cam-seg-data/train_cls.txt'
ANNOTATION_PATH_TEST = 'engn8536/Datasets/cifar-10-cam-seg-data/test_cls.txt'
IMG_PATH_TRAIN = 'engn8536/Datasets/cifar-10-cam-seg-data/train'
IMG_PATH_TEST = 'engn8536/Datasets/cifar-10-cam-seg-data/test'
SEG_PATH = 'engn8536/Datasets/cifar-10-cam-seg-data/test_seg'

Transform_L = transforms.Compose([transforms.Resize((224,224)),
                                  transforms.ToTensor(),
                                  transforms.Normalize((0.485, 0.456, 0.406),
(0.229, 0.224, 0.225))])
Transform_S = transforms.Compose([transforms.Resize((112,112)),
                                  transforms.ToTensor(),
                                  transforms.Normalize((0.485, 0.456, 0.406),
(0.229, 0.224, 0.225))])
Transform_Seg = transforms.Compose([transforms.Resize((112,112)),
                                    transforms.ToTensor()])

class Cifar10Loader(Dataset):
    # Cifar-10 Dataset
    def __init__(self, args, split='train'):
        assert split in ['train', 'test']
        self.dict = {}
        self.split = split

        if self.split == 'test': # train or test mode
            self.img_path = IMG_PATH_TEST
            self.seg_path = SEG_PATH
            self.anno_path = ANNOTATION_PATH_TEST
        else:
            self.img_path = IMG_PATH_TRAIN
            self.anno_path = ANNOTATION_PATH_TRAIN

        with open(self.anno_path, 'r') as file:
            data = file.readlines()
            for idx, line in enumerate(data):
                img_name = line.split(' ')[0]
                img_label = line.split(' ')[1]
                self.dict[idx] = (img_name, int(img_label))

        self.Transform_L = Transform_L
        self.Transform_S = Transform_S
        if args.augmentation1:
            self.Transform_Seg = transforms.Compose([transforms.Resize((112,
112)),

transforms.RandomVerticalFlip(p=1),
                                                     transforms.ToTensor()])
        elif args.augmentation2:
            self.Transform_Seg = transforms.Compose([transforms.Resize((112,
112)),

transforms.RandomVerticalFlip(p=1),

transforms.RandomHorizontalFlip(p=1),
                                                     transforms.ToTensor()])
        else:
            self.Transform_Seg = Transform_Seg

    def __len__(self):
        return len(self.dict)

    def __getitem__(self, idx):
        img_name, img_label = self.dict[idx]

        img = Image.open(os.path.join(self.img_path, img_name))
        img_L = self.Transform_L(img)

        if self.split == 'test':
            sample = {'img_L': img_L, 'label': torch.tensor(img_label)}
            return sample
        else:
            img_S = self.Transform_S(img)
            sample = {'img_L': img_L, 'img_S': img_S, 'label':
torch.tensor(img_label)}
            return sample
```

Appendix C – "main.py"

```
import argparse
from model import CAMModel
from dataloader import Cifar10Loader
from train import train, resume, evaluate, train_CV
import os

import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--exp_id', type=str, default='exp_CAM')
    parser.add_argument('--mode', type=str, default='CAM', help='CAM or SEG')

    parser.add_argument('--resume', type=int, default=0, help='resume the trained model')
    parser.add_argument('--test', type=int, default=0, help='test with trained model')

    parser.add_argument('--epochs', type=int, default=10, help='number of training epochs')

    parser.add_argument('--batch_size', type=int, default=64)
    parser.add_argument('--lr', type=float, default=1e-4, help='learning rate')

    parser.add_argument('--seed', type=int, default=1, help='random seed')

    parser.add_argument('--cv', type=int, default=0, help='cross validation for cam')

    parser.add_argument('--augmentation1', type=int, default=0, help='vertically flip image - data augmentation')
    parser.add_argument('--augmentation2', type=int, default=0,
              help='vertically flip and horizontally flip image - data augmentation')

    args = parser.parse_args()
    return args

if __name__ == '__main__':
    args = parse_args()
    torch.manual_seed(args.seed)
    torch.cuda.manual_seed(args.seed)

    # dataloaders
    trainloader = DataLoader(Cifar10Loader(args, split='train'),
        batch_size=args.batch_size, shuffle=True, num_workers=2)
    testloader = DataLoader(Cifar10Loader(args, split='test'),
        batch_size=args.batch_size, shuffle=False, num_workers=2)
    dataloaders = (trainloader, testloader)

    classes = ('plane', 'car', 'bird', 'cat',
               'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

    # network
    model = CAMModel(args).to(device)
    # optimizer
    optimizer = torch.optim.Adam(model.parameters(), lr=args.lr, betas=(0.9,0.999))

    # resume the trained model
    if args.resume:
        model, optimizer = resume(args, model, optimizer)

    if args.test == 1: # test mode
        testing_accuracy = evaluate(args, model, testloader)
        print('testing finished, accuracy: {:.3f}'.format(testing_accuracy))
    else: # train mode, train the network from scratch
        if args.cv:
            train_CV(args, model, optimizer, dataloaders)
        else:
            train(args, model, optimizer, dataloaders)
        print('training finished')
```

Appendix D - “dataloader.py”

```
import os
import PIL.Image as Image

import torch
from torch.utils.data import Dataset
from torchvision import transforms

ANNOTATION_PATH_TRAIN = 'engn8536/Datasets/cifar-10-cam-seg-data/train_cls.txt'
ANNOTATION_PATH_TEST = 'engn8536/Datasets/cifar-10-cam-seg-data/test_cls.txt'
IMG_PATH_TRAIN = 'engn8536/Datasets/cifar-10-cam-seg-data/train'
IMG_PATH_TEST = 'engn8536/Datasets/cifar-10-cam-seg-data/test'
SEG_PATH = 'engn8536/Datasets/cifar-10-cam-seg-data/test_seg'

Transform_L = transforms.Compose([transforms.Resize((224,224)),
                                  transforms.ToTensor(),
                                  transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])
Transform_S = transforms.Compose([transforms.Resize((112,112)),
                                  transforms.ToTensor(),
                                  transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])
Transform_Seg = transforms.Compose([transforms.Resize((112,112)),
                                    transforms.ToTensor()])

class Cifar10Loader(Dataset):
    # Cifar-10 Dataset
    def __init__(self, args, split='train'):
        assert split in ['train', 'test']
        self.dict = {}
        self.split = split

        if self.split == 'test': # train or test mode
            self.img_path = IMG_PATH_TEST
            self.seg_path = SEG_PATH
            self.anno_path = ANNOTATION_PATH_TEST
        else:
            self.img_path = IMG_PATH_TRAIN
            self.anno_path = ANNOTATION_PATH_TRAIN

        with open(self.anno_path, 'r') as file:
            data = file.readlines()
            for idx, line in enumerate(data):
                img_name = line.split(' ')[0]
                img_label = line.split(' ')[1]
                self.dict[idx] = (img_name, int(img_label))

        self.Transform_L = Transform_L
        self.Transform_S = Transform_S
        if args.augmentation1:
            self.Transform_Seg = transforms.Compose([transforms.Resize((112, 112)),
                                                     transforms.RandomVerticalFlip(p=1),
                                                     transforms.ToTensor()])
        elif args.augmentation2:
            self.Transform_Seg = transforms.Compose([transforms.Resize((112, 112)),
                                                     transforms.RandomVerticalFlip(p=1),
                                                     transforms.RandomHorizontalFlip(p=1),
                                                     transforms.ToTensor()])
        else:
            self.Transform_Seg = Transform_Seg

    def __len__(self):
        return len(self.dict)

    def __getitem__(self, idx):
        img_name, img_label = self.dict[idx]

        img = Image.open(os.path.join(self.img_path, img_name))
        img_L = self.Transform_L(img)

        if self.split == 'test':
            sample = {'img_L': img_L, 'label': torch.tensor(img_label)}
            return sample
        else:
            img_S = self.Transform_S(img)
            sample = {'img_L': img_L, 'img_S': img_S, 'label': torch.tensor(img_label)}
            return sample
```

Appendix E - “visualize.py”

```
# author: Yicong Hong (yicong.hong@anu.edu.au) for Lab4, 2020, ENGN8536 @ANU

import argparse
from train import resume
from model import CAMModel
import os
import glob
import cv2
import numpy as np
import PIL.Image as Image
import torch
import torch.nn.functional as F
from torchvision import transforms

''' Refer to CAM visualization code by BoleiZhou (bzhou@csail.mit.edu),
    the author of paper Learning Deep Features for Discriminative Localization,
    see https://github.com/zhoubolei/CAM/blob/master/pytorch_CAM.py '''

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--exp_id', type=str, default='exp_CAM')
    parser.add_argument('--mode', type=str, default='CAM', help='CAM or SEG')
    parser.add_argument('--thresh', type=int, default=175, help='threshold value')
    args = parser.parse_args()
    return args

def returnCAM(feature_conv, weights):
    # generate the class activation maps upsample to 224x224
    size_upsample = (224, 224)
    nc, h, w = feature_conv.shape

    #weight_softmax = F.softmax(weights, dim=0)
    #cam = weight_softmax.unsqueeze(-1)*(feature_conv.reshape((nc, h*w)))
    #cam = cam.sum(0).cpu().numpy()

    weight_softmax = F.softmax(weights, dim=1)
    cam = weight_softmax.unsqueeze(-1)*(feature_conv.reshape((nc, h*w)))
    cam = cam.sum(1).cpu().detach().numpy()

    print(cam.shape, h, w)
    cam = cam.reshape(h, w)
    cam = cam - np.min(cam)
    cam_img = cam / np.max(cam)
    cam_img = np.uint8(255 * cam_img)
    output_cam = cv2.resize(cam_img, size_upsample)
    return output_cam

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

if __name__ == '__main__':
    args = parse_args()

    # network
    model = CAMModel(args).to(device)
    model.eval()
    # optimizer (useless)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.00, betas=(0.9,0.999))
    # resume the trained model (assume trained CAM/SEG models exist)
    model, optimizer = resume(args, model, optimizer)

    # source images
    img_list = glob.glob('engn8536/Datasets/cifar-10-cam-seg-data/test_seg_source/*')
    for img_path in img_list:
        img = Image.open(img_path)
        img_tensor = preprocess(img).unsqueeze(0).cuda()

        with torch.no_grad():
            _, feature_conv, weights = model(img_tensor)

        # render the CAM and output
        # feature_conv, weights = cam_source
        output_cam = returnCAM(feature_conv.squeeze(), weights)

        out_img = cv2.imread(img_path)
        out_img = cv2.resize(out_img,(224, 224))
        heatmap = cv2.applyColorMap(output_cam, cv2.COLORMAP_JET)
        result = heatmap * 0.15 + out_img * 0.3
        img_name = img_path.split('/')[-1]

        # threshold the heatmap, use it as the segmentation map
        ret, thresh_img = cv2.threshold(output_cam, args.thresh, 255, cv2.THRESH_BINARY)

        if args.mode == 'CAM':
            cv2.imwrite('./visualize/CAM/'+img_name, result)
            thresh_img_name = img_name.split('.')[0] + '_seg.png'
            cv2.imwrite('./visualize/CAM/'+thresh_img_name, thresh_img)
        elif args.mode == 'SEG':
            cv2.imwrite('./visualize/SEG/'+img_name, result)
            thresh_img_name = img_name.split('.')[0] + '_seg.png'
            cv2.imwrite('./visualize/SEG/'+thresh_img_name, thresh_img)

    else:
        NotImplementedError
```

Appendix F - “iou.py”

```
import cv2
import numpy as np

def compute_iou(seg_path_gt, seg_path_cam, seg_path_seg):
    im_gt = cv2.imread(seg_path_gt)
    im_cam = cv2.resize(cv2.imread(seg_path_cam), dsize=(im_gt.shape[0:2]))
    im_seg = cv2.resize(cv2.imread(seg_path_seg), dsize=(im_gt.shape[0:2]))

    inter_px_cam, intersection_cam = np.unique(np.bitwise_and(im_gt, im_cam), return_counts=True)
    inter_px_seg, intersection_seg = np.unique(np.bitwise_and(im_gt, im_seg), return_counts=True)
    union_px_cam, union_cam = np.unique(np.bitwise_or(im_gt, im_cam), return_counts=True)
    union_px_seg, union_seg = np.unique(np.bitwise_or(im_gt, im_seg), return_counts=True)

    iou_CAM = sum(intersection_cam*inter_px_cam) / sum(union_px_cam*union_cam)
    iou_SEG = sum(intersection_seg*inter_px_seg) / sum(union_px_seg*union_seg)

    return iou_CAM, iou_SEG


if __name__ == '__main__':
    classes = ('plane', 'car', 'bird', 'cat',
               'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

    for cls in classes:
        seg_path_gt = './data/test_seg/{}.png'.format(cls)       # ground-truth seg map
        seg_path_cam = './visualize/CAM/{}_seg.png'.format(cls)  # output seg map from CAM
        seg_path_seg = './visualize/SEG/{}_seg.png'.format(cls)  # output seg map from SEG

        iou_CAM, iou_SEG = compute_iou(seg_path_gt, seg_path_cam, seg_path_seg)

        print('Class: {} | CAM IoU: {:.3f} | SEG IoU: {:.3f}'.format(cls, iou_CAM, iou_SEG))
```