# lab4

## Grad Cam

我尝试自己复现了一下Grad Cam

首先是瞅一眼Alexnet的网络结构

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

在feature最后一层也就是最后一个卷积层打上hook函数

```
grad_block = [] # 存放grad图
feaure_block = []   # 存放特征图

# 获取梯度的函数
def backward_hook(module, grad_in, grad_out):
    grad_block.append(grad_out[0].detach())

# 获取特征层的函数
def farward_hook(module, input, output):
```

```
    feaure_block.append(output)


# layer_name=model.features[18][1]
model.features.register_forward_hook(farward_hook)
model.features.register_full_backward_hook(backward_hook)
```

然后向前向后传播

```
# forward
# 在前向推理时，会生成特征图和预测值
output = model(img)
max_idx = np.argmax(output.data.numpy())

# backward
model.zero_grad()
# 取最大类别的值作为loss，这样计算的结果是模型对该类最感兴趣的cam图
class_loss = output[0, max_idx]
class_loss.backward()   # 反向梯度，得到梯度图
```

利用得到的原图，梯度，特征图来计算热图

```
def cam_show_img(img, feature_map, grads):
    cam = np.zeros(feature_map.shape[1:], dtype=np.float32)
    grads = grads.reshape([grads.shape[0], -1])
    # 梯度图中，每个通道计算均值得到一个值，作为对应特征图通道的权重
    weights = np.mean(grads, axis=1)
    for i, w in enumerate(weights):
        cam += w * feature_map[i, :, :] # 特征图加权和
    cam = np.maximum(cam, 0)
    cam = cam / cam.max()
    cam = cv2.resize(cam, (W, H))
    heatmap = cv2.applyColorMap(np.uint8(255 * cam), cv2.COLO

    # print(heatmap.shape)
    # print(img.size)

    cam_img = 0.3 * heatmap + 0.7 * img
```

```
    cv2.imwrite("./out/"+path, cam_img)
```

## 实验结果



# Layer Cam

我采用的是作者开源的代码，并尝试将其移植到Alexnet上

我选择可视化的层有 0 3 6 9 12层，然后使用LayerCAM类绘制Layercam，最后调用作者写的basic_visualize来进行打印。

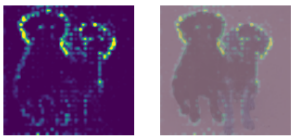
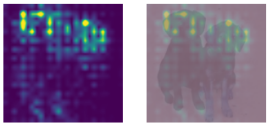
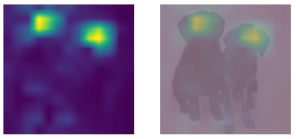
```
def get_arguments():
    parser = argparse.ArgumentParser(description='The Pytorch
    parser.add_argument("--img_path", type=str, default='both
    parser.add_argument("--layer_id", type=list, default=[0,3
    return parser.parse_args()


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

    path = args.img_path
    raw_img = cv2.imread("./images/"+path)
    preprocess = transforms.ToTensor()
    img = preprocess(raw_img)
    img = torch.unsqueeze(img, 0)
```

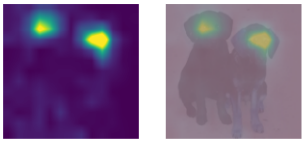
```
    model = torch.load('torch_alex.pth')

    for i in range(len(args.layer_id)):
        layer_name = 'features_' + str(args.layer_id[i])
        model_dict = dict(type='alexnet', arch=model, layer_n
        layercam = LayerCAM(model_dict)
        predicted_class = model(img).max(1)[-1]

        layercam_map = layercam(img)
        basic_visualize(img.detach(), layercam_map.type(torch
```

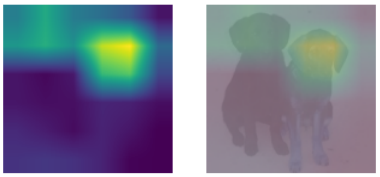
## 实验结果

左边代表的是梯度，右边是将梯度与原图结合起来一起看，可视化的层分别为 0 3 6 9 12层

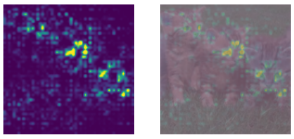
### dog

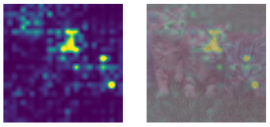




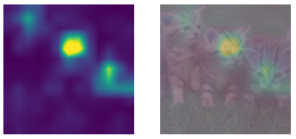


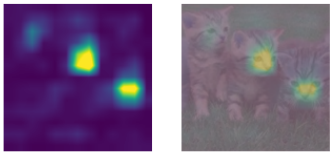


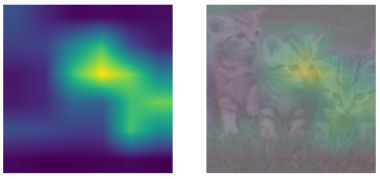
**cat**











## both