Assignment 4

r12922054 資工所 邱信瑋

Pseudocodes of Grad-CAM

Algorithm 1 Grad-CAM

1: $A^k_{\text{ij}} \leftarrow$ Target layer feature maps;
   i, j represent the pixel x coordinate and y coordinate of the feature map;
   k indicates which feature map is in this layer
2: $Y^c \leftarrow$ Final score for each class c
3: **procedure** Grad-CAM:
4: Do forwarding and hook the target layer to get A^k_{ij}.
5: $\alpha^c_k \leftarrow \frac{1}{Z} * \sum_i \sum_j * \frac{\partial Y^c}{\partial A^k_{\text{ij}}}$, α represents weights
6: $L^c_{\text{Grad\_CAM}} \leftarrow \text{ReLU}(\sum_k (\alpha^c_k * A^k)$
7: return $L_{\text{Grad\_CAM}}$

Experiment Setting

I. Hardware Specification

- CPU : Intel(R) Core(TM) i7-6700K CPU 4.00GHz
- GPU : NVIDIA GeForce RTX 2070 8GB

II. Package Version

- python 3.10.13
- torch 1.11.0+cu113
- torchvision 0.12.0+cu113
- numpy 1.26.0
- tqdm 4.66.1
- matplotlib 3.8.0
- Pillow 10.0.1

III. Please explain how to use Grad-CAM for visualizing this model; please illustrate your method and visualize the results in the report.

This assignment mainly involves the implementation of Grad-Cam. In addition, it is also necessary to implement Guided Backpropagation, which can increase the resolution after visualization. The implementation steps are as follows:

1. **Guided Backpropagation:**

The definition of Guided backpropagation is that they not only need the value to be positive during forward, but also the value after Backpropagation to be positive, and everything else is 0. The concept is roughly that when doing Backpropagation on a feature map, if a certain position (x, y) of a feature map is more relevant to the category, then the Backpropagation will usually be positive, because The weight of the feature maps of that category needs to be increased. Figure 1 is the implemented code.

```python
class GuidedBackpropReLU_2(Function):
    @staticmethod
    def forward(ctx,input):
        ctx.save_for_backward(input)
        return input.clamp(min=0)

    @staticmethod
    def backward(ctx,grad_output):
        input = ctx.saved_tensors[0]
        grad_input = grad_output.clone()
        grad_input[grad_input<0] = 0
        grad_input[input<0]=0
        return grad_input

# 修改模型中的ReLU層
def modify_ReLU_2(model):
    for name, module in model.named_children():
        if isinstance(module, nn.ReLU):
            module.register_forward_hook(lambda module, input, output: GuidedBackpropReLU_2.apply(output))
        else:
            modify_ReLU_2(module)
```

Figure 1: GuidedBackpropReLU code.

2. **Grad-CAM:**

Before explaining Grad-CAM, I would like to first talk about the implementation process of CAM. The method is to connect the GAP layer after the last convolution layer. After each feature map is converted by GAP, the information of the feature map is compressed into one neuron, so each neuron after GAP conversion corresponds to a certain feature map of the last layer, and the weight connected to the GAP layer can be regarded as the importance of each feature map to the model prediction category. , and finally weight each feature map according to its corresponding weight to obtain the CAM. From the introduction of CAM, we know that in order to be able to implement CAM, the final convolutional layer output must be connected to the GAP Layer. However, Grad-CAM does not have such limitations.

The running process of Grad-CAM for this assignment is as follows. First, specify one of the convolution layers as the target layer, hook it, and perform forwarding to obtain the feature maps of this layer. Second, the score of the prediction result (In practice, we set the correct class to 1 and the others to 0) and perform partial differentiation on the pixels of each feature map of the target layer to obtain the gradient. Third, for each feature map, obtain the average value of its pixel gradient, as the weight of the feature map. Finally, after linear combination of the feature map and its corresponding weight, do a ReLU and resize the image size, we can get the result we want.

```python
def generate_guided_gradcam(self):
    weights = torch.mean(self.gradients, dim=(2, 3), keepdim=True)
    cam = torch.sum(weights * self.feature_maps, dim=1, keepdim=True)
    cam = self.normalize_cam(cam)
    cam = F.relu(cam)
    cam = F.interpolate(cam, size=(28, 28), mode='bilinear', align_corners=False)
    # cam = cam.squeeze(1)
    cam = cam.squeeze().detach().cpu().numpy()
    cam = (cam - 0) / (cam.max())

    return cam
```

Figure 2: Grad-CAM code.

IV. Visually show the Grad-CAM map of the given testing images in the report.

The following are the results of my test. Some interesting phenomena can be found. Generally speaking, human intuition will think that in order to correctly judge the results, the model should pay attention to the white areas. This is indeed the case in the results of Figure 3. However, in Figure 4 we can find that the attention area of the graph is in the black part.

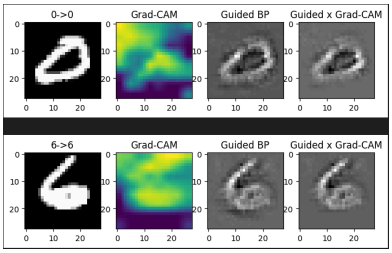

Figure 3: Attention on the white areas.

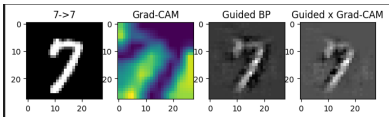

Figure 4: Attention on the black areas.

V.a detailed description of the parameter settings and the implementation in q2.

In this paragraph, I will describe the details of this implementation and some of my attempts, as follows:

1. Each layer in the model is independently declared as an attribute(Figure 5), so that we can select the layer we want through Pytorch's hook function.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.relu1 = nn.ReLU()
        self.avgpooling1 = nn.AvgPool2d(2,2)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.relu2 = nn.ReLU()
        self.avgpooling2 = nn.AvgPool2d(2,2)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.avgpooling1(x)
        x = self.conv2(x)
        x = self.relu2(x)
        x = self.avgpooling2(x)
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

Figure 5: model.

2. In the original model, MaxPooling was used. I changed it to AvgPooling. To compare the results, they are shown in Figure 6 and Figure 7 below.(It proves that Grad-CAM does not need the last layer to do GAP)

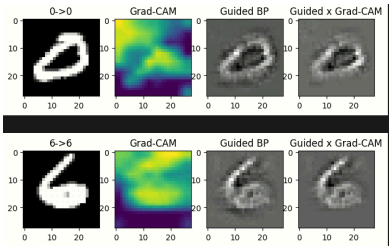

Figure 6: Do maxpooling.

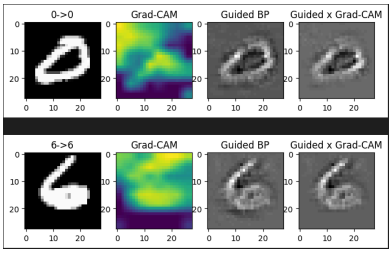

Figure 7: Do avgpooling.

3. In the test data set, I had a piece of data that was originally predicted incorrectly, so I fine-tuned the model until the prediction was correct. The following compares the results before fine-tune(Figure 8) and after fine-tune(Figure 9).

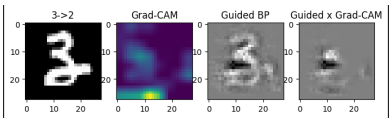

Figure 8: Original result was wrong.

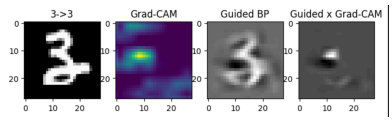

Figure 9: After fine-tune, the result is correct.

4. In the implementation of Grad-CAM this time, after I obtained the linear combination of feature map and weight, I did an additional normalization. This normalization is not a general normalization, but in order to achieve better results in the visualization where to strengthen the model's attention part.The results of comparison are as follows, Figure 10 and Figure 11. (All my previous picture results were normalized.)

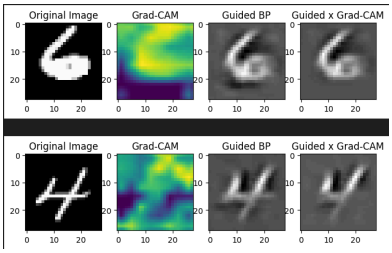

Figure 10: With nomalization.

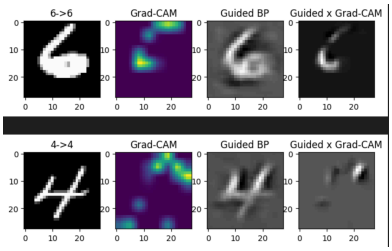

Figure 11: Without nomalization.