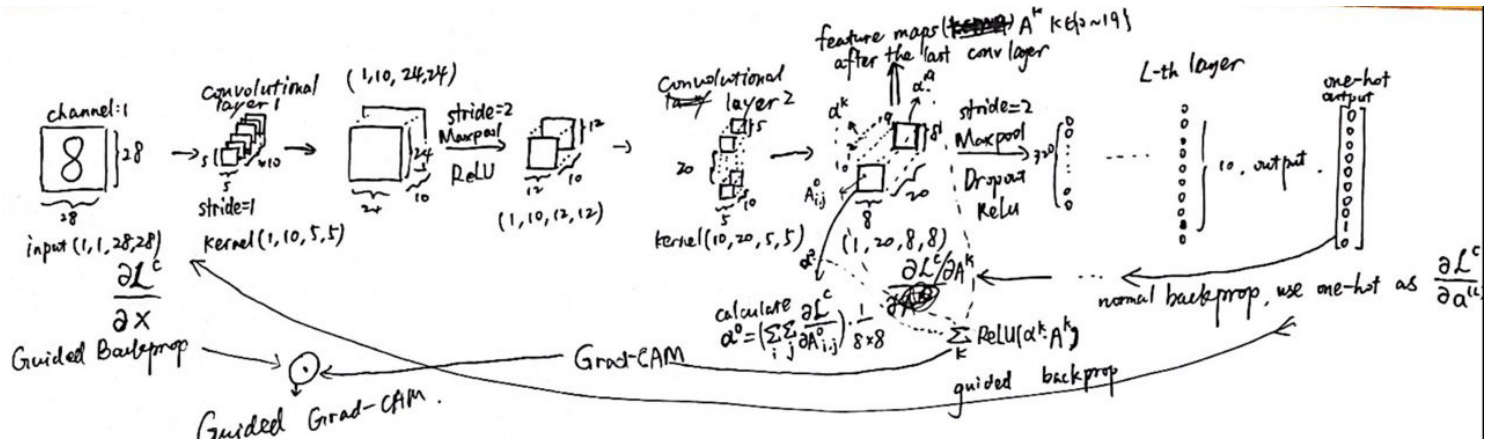
1 Experiment settings

1.1 Hardware Specification & Package version

- computation hardware: (cpu) intel i5

Package	Version
torch	2.0.1
• torchvison	0.15.2
matplotlib	3.8.0
python	3.10.2

1.2 How to use Grad-CAM for visualizing this model



1. Grad-CAM: Expose feature maps in forward function in the given class. Then input each testing image to model to perform forward pass. After that, produce a one-hot vector as the partial derivatives of output to do the backprop. Finally, you can get gradients and activations from feature maps, with which you can perform operations to get Grad-CAM.

2. GuidedBackprop: After gaining all the Grad-CAMs of testing images, it's time to register backward pre-hook on all the layers of the model, in which you can manually modify the gradient before each layer perform calculation during backprop. In this context, we need to apply ReLU. Then after setting testing image to require gradients, input it to model and do the backprop. This time the backprop would automatically be guided. Lastly, collect the gradient from the image tensor.

3. Guided Grad-CAM: For each testing image,after collecting its grad-CAM and guided gradient, simply element-wise multiply them.

1.3 Visually show the Grad-CAM of the given testing images

Since I have 8 testing images, and each of them would be plotted 4 times(Raw, Grad-MAP, Guided Backprop, Guided Grad-MAP), so instead of profile, I choose a landscape layout for my output.

1.4 Description of the parameter settings and the implementation

- parameter settings
  - . model parameters: as given
    - ∗ input shape: (N, C, H, W) = (1, 1, 28, 28)
    - ∗ after first convolutional layer and a maxpool layer: (N, C, H, W) = (1, 10, 12, 12)

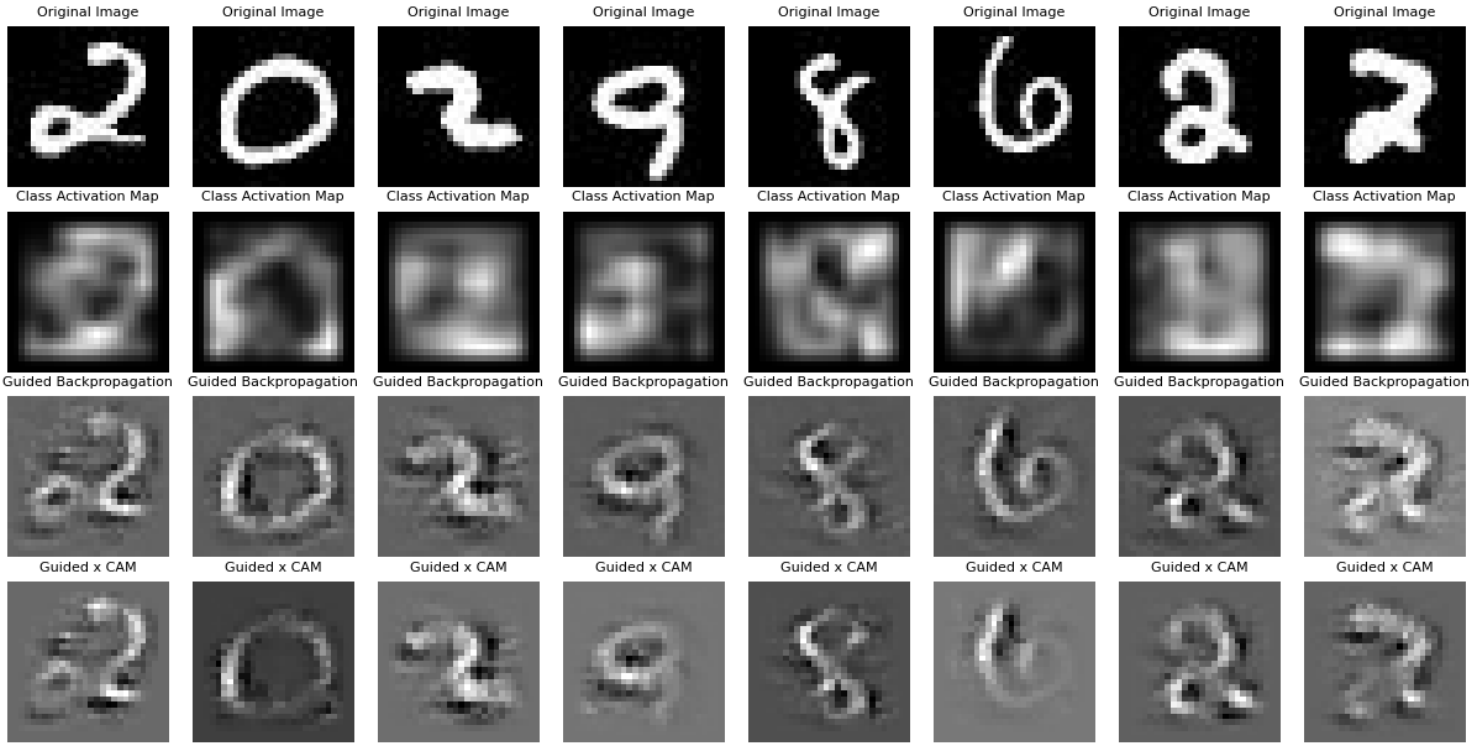


Figure 1: Output from T12902101_a4.py

  - after the second (also the last) convolutional layer: (N, C, H, W) = (1, 10, 8, 8), so the feature map resolution is 8x8

. padding size after obtaining 8x8 Grad-MAP: for each side(1, 1, 1, 1)

. Interpolate mode after padding: Bilinear(same as the given paper[1])

. trivial parameters when plotting are ignored

- implementation

NOTE Due to the absence of padding in the convolutional layers, at a resolution of 8x8, GradCAM corresponds to the central portion of the original image excluding the corners. Therefore, each side of the GradCAM would be padded by one unit before being applied to bilinear interpolation. This operation would give the final outcome a positive impact.

2 Before executing `T12902101_a4.py`

Please note that I changed the file paths to make my assignments directory organized. Change them before you run the code.

```
# line 43
model.load_state_dict(torch.load("../mnist_model.pth", map_location="cpu"))
...
    raw_image = Image.open(f"../data/my_images/image_{i}.jpg")  # line 69
```

By the way, images of `"../data/my_images/image_i.jpg"` are created after executing this code, which would transform my raw testing data `47_data.json` into grey images and save them to that path. For convenience, I packed my testing images together at `./my_images`.

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

Algorithm 1 Guided Grad-CAM Generation Algorithm

$model \leftarrow$Initialize a model of provided neural network class, load pre-trained model weights, and set its mode to evaluate
$out\_channels \leftarrow$ Output channel count of the last convolutional layer in the neural network
$feature\_maps \leftarrow$ Activations of the last convolutional layer as raw feature maps
$u, v \leftarrow$ Height, width of the output from last convolutional layer
$h, w \leftarrow$ Height, width of the testing image

procedure ForthBack(img)
    $output \leftarrow$ Forward($model, img$) ▷ Perform forward pass through the model
    $onehot \leftarrow$ GetOneHotOutput($output$)
    Backward($output$, $onehot$) ▷ Calculate gradients with interest
end procedure

procedure GenerateGradCAM(img)
    ForthBack(img) ▷ Forward pass and backpropagation
    $\nabla_{feature\_maps} \leftarrow$ GetGradients($feature\_maps$)
    for $k = 0$ to $out\_channels - 1$ **do**
        $\nabla^k \leftarrow \nabla^k_{feature\_maps}$ ▷ Gradients for the k-th feature map
        $\alpha_k \leftarrow \frac{1}{u \cdot v} \sum_{i=0}^{u-1} \sum_{j=0}^{v-1} \nabla^k_{i,j}$ ▷ Calculate weight (importance) for the k-th feature map
    end for
    $grad\_CAM \leftarrow \sum_{k=0}^{out\_channels-1} \text{ReLU}(\alpha_k \cdot feature\_maps_k)$
    $grad\_CAM \leftarrow$ Pad($grad\_CAM, 1$)
    $grad\_CAM \leftarrow$ BilinearInterpolate($grad\_CAM, h, w$)
    ZeroOutGradients(model)
    return $grad\_CAM$
end procedure

procedure BackpropGiude($grad_{input}, \dots$)
    $grad_{positive} \leftarrow$ ReLU($grad_{input}$)
    return $grad_{positive}$
end procedure

procedure GuidedBackprop(img)
    Enable gradients computation for the input image
    ForthBack(img) ▷ Forward pass and Execute guided backpropagation
    $guided\_grad \leftarrow$ GetGradients(img)
    return $guided\_grad$
end procedure

procedure GenerateGuidedGradCAM($image$)
    $grad\_CAM \leftarrow$ GenerateGradCAM($image$)
    for $layer \in$ children($model$) **do**
        RegisterBackwardPreHook($layer, BackpropGuide$)
    end for
    $grad\_grad \leftarrow$ GuidedBackprop($image$)
    $guided\_CAM \leftarrow guided\_grad \odot grad\_CAM$ ▷ Element-wise multiplication
    return $guided\_CAM$
end procedure