CAM

Learning Deep Features for Discriminative Localization

1. About The Paper

Problem and proposed Solution

Problem

The ability to localize objects from CNNs is lost in the FC layers

Need to avoid the use of FC layers

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Solution

The ability to localize objects from CNNs is lost in the FC layers

Need to avoid the use of FC layers

Replace multiple FC layers with Global Average Pooling GAP

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
GAP					
FC-1000					
soft-max					

What the paper is about

The ability to localize objects from CNNs is lost in the FC layers

Need to avoid the use of FC layers

Replace multiple FC layers with Global Average Pooling GAP

Applying GAP for accurate discriminative localization

CAM: Class Activation Mapping

	LRN	**conv3-64**	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
GAP					
FC-1000					
soft-max					

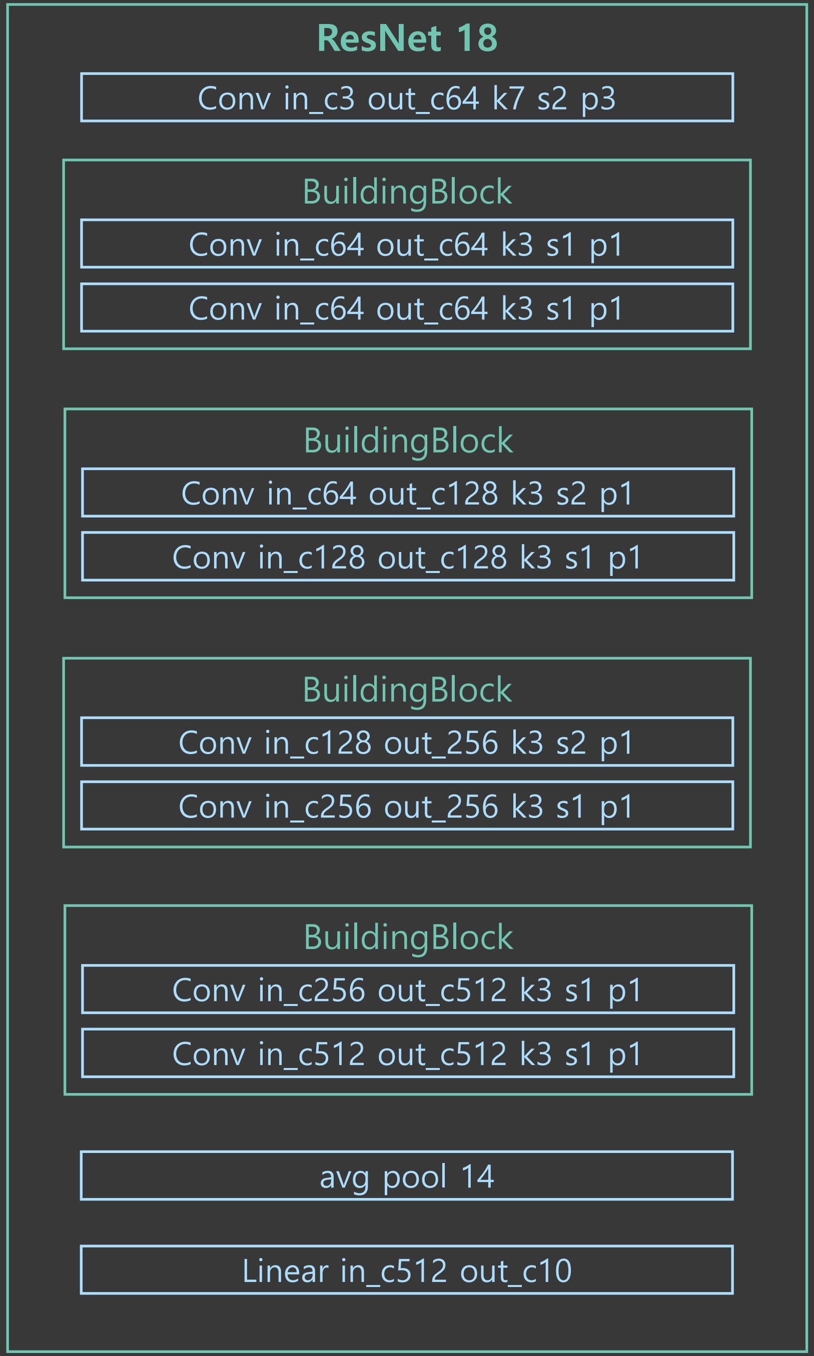
2. Model

Constructing ResNet18B suited for CAM

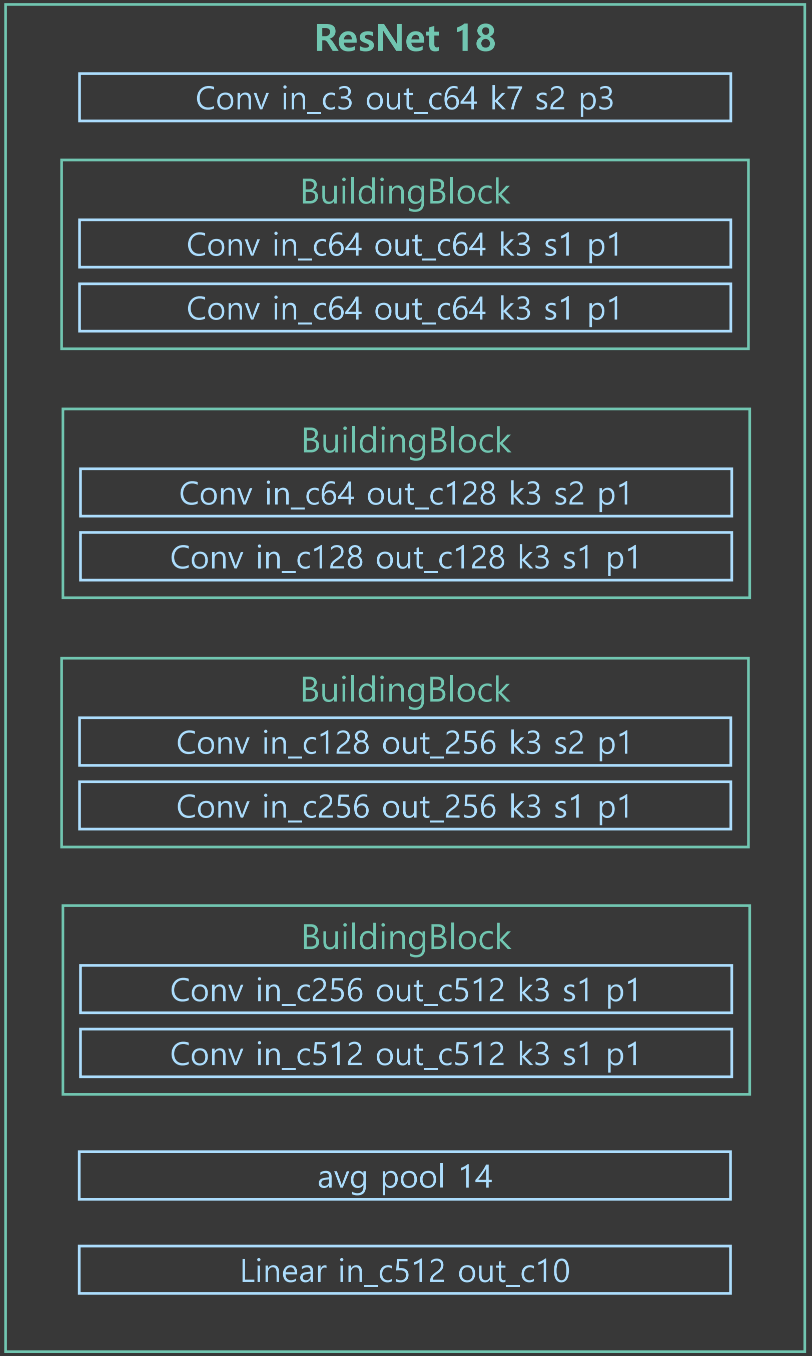
Implementation: Model

Alternation of ResNet18 to make resulting mapping resultion 14 X 14

ResNet already uses GAP so not much to change



```python
class BuildingBlock(nn.Module):

    def __init__(self, in_c, out_c, stride=1, option='B'):
        super(BuildingBlock, self).__init__()

        self.conv1 = nn.Conv2d(in_c, out_c, kernel_size=3, stride=stride, padding=1)
        self.bn1 = nn.BatchNorm2d(out_c)
        self.conv2 = nn.Conv2d(out_c, out_c, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(out_c)

        self.shortcut = nn.Sequential()
        if in_c != out_c:
            if option == 'A':
                self.shortcut = Padding(in_c, out_c)
                # why not concantenate x instead of padding?
                # since dim increase by factor of 2 all the time

            if option == 'B':
                self.shortcut = nn.Sequential(
                    nn.Conv2d(in_c, out_c, kernel_size=1, stride=stride),
                    nn.BatchNorm2d(out_c)
                )
                # i don't like the idea of batchnormalization for projection shortcut
                # should i add BN?

            # additional option I thought of hehe
            if option == 'Mine':
                self.shortcut = Concat(in_c, out_c)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```



```python
class BuildingBlock(nn.Module):

    def __init__(self, in_c, out_c, stride=1, option='B'):
        super(BuildingBlock, self).__init__()

        self.conv1 = nn.Conv2d(in_c, out_c, kernel_size=3, stride=stride, padding=1)
        self.bn1 = nn.BatchNorm2d(out_c)
        self.conv2 = nn.Conv2d(out_c, out_c, kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(out_c)

        self.shortcut = nn.Sequential()
        if in_c != out_c:
            if option == 'A':
                self.shortcut = Padding(in_c, out_c)
                # why not concantenate x instead of padding?
                # since dim increase by factor of 2 all the time

            if option == 'B':
                self.shortcut = nn.Sequential(
                    nn.Conv2d(in_c, out_c, kernel_size=1, stride=stride),
                    nn.BatchNorm2d(out_c)
                )
                # i don't like the idea of batchnormalization for projection shortcut
                # should i add BN?

            # additional option I thought of hehe
            if option == 'Mine':
                self.shortcut = Concat(in_c, out_c)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

BuildingBlock

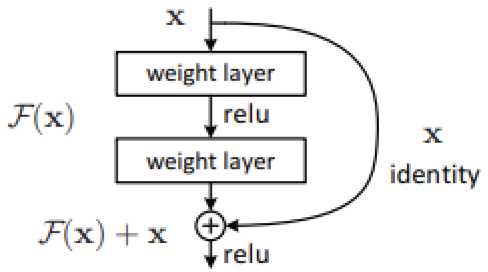
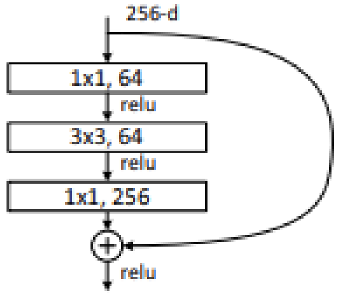


Figure 2. Residual learning: a building block.

Bottleneck

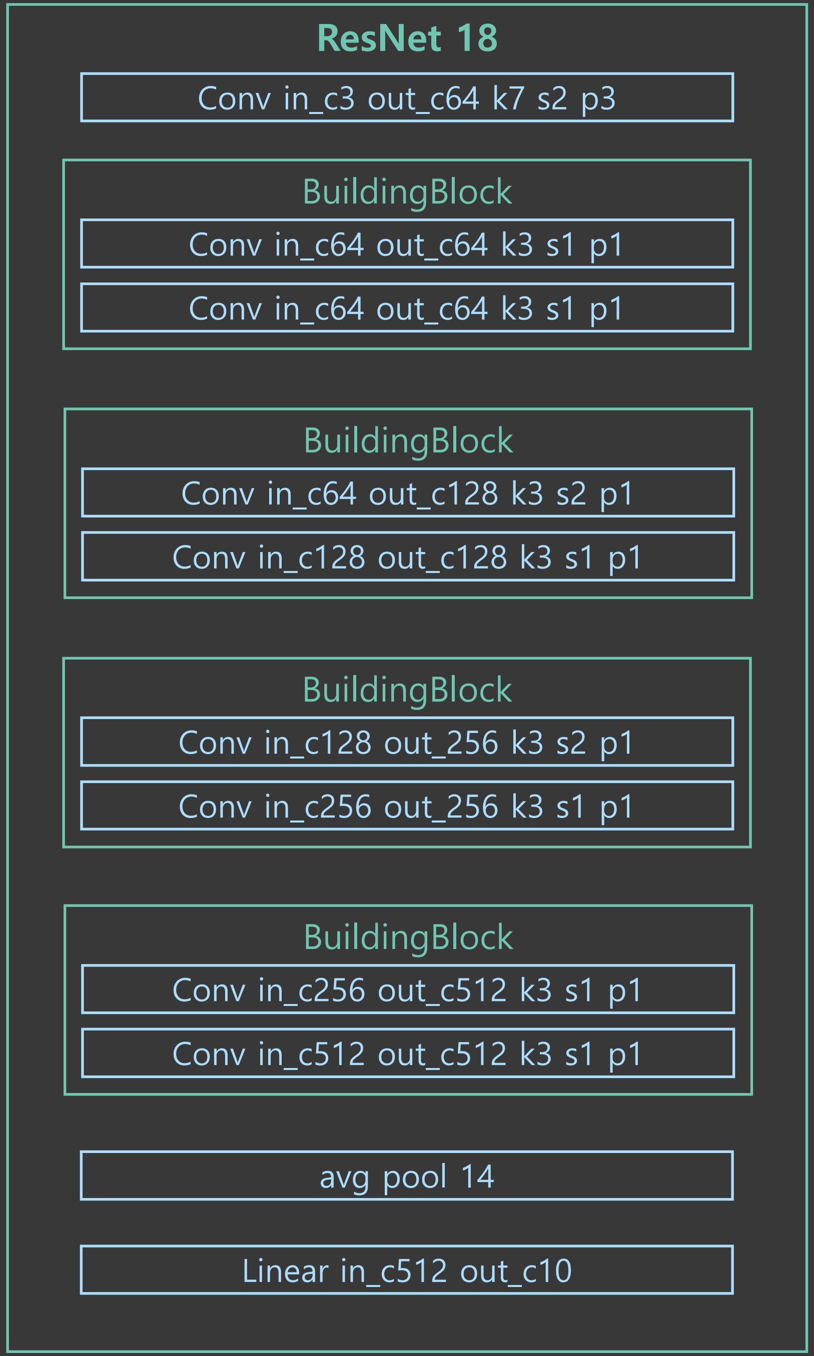


```python
class ResNet18B(nn.Module):
    def __init__(self):
        super(ResNet18B, self).__init__()

        self.gate = Gate()

        self.conv2_1 = BuildingBlock(64, 64, 1, 'B')
        self.conv2_2 = BuildingBlock(64, 64, 1, 'B')

        self.conv3_1 = BuildingBlock(64, 128, 2, 'B')
        self.conv3_2 = BuildingBlock(128, 128, 1, 'B')

        self.conv4_1 = BuildingBlock(128, 256, 2, 'B')
        self.conv4_2 = BuildingBlock(256, 256, 1, 'B')

        self.conv5_1 = BuildingBlock(256, 512, 1, 'B')
        self.conv5_2 = BuildingBlock(512, 512, 1, 'B')

        self.output = nn.Linear(512, 10, bias=False)
```

ResNet 18

- Conv in_c3 out_c64 k7 s2 p3
- BuildingBlock
  - Conv in_c64 out_c64 k3 s1 p1
  - Conv in_c64 out_c64 k3 s1 p1
- BuildingBlock
  - Conv in_c64 out_c128 k3 s2 p1
  - Conv in_c128 out_c128 k3 s1 p1
- BuildingBlock
  - Conv in_c128 out_256 k3 s2 p1
  - Conv in_c256 out_256 k3 s1 p1
- BuildingBlock
  - Conv in_c256 out_c512 k3 s1 p1
  - Conv in_c512 out_c512 k3 s1 p1
- avg pool 14
- Linear in_c512 out_c10

```python
    def forward(self, x):
        #print("input", x.shape)
        x = self.gate(x)

        #print("1", x.shape)
        x = self.conv2_1(x)
        x = self.conv2_2(x)
        #print("2", x.shape)

        x = self.conv3_1(x)
        x = self.conv3_2(x)
        #print("3", x.shape)

        x = self.conv4_1(x)
        x = self.conv4_2(x)
        #print("4", x.shape)

        x = self.conv5_1(x)
        x = self.conv5_2(x)
        #print("5", x.shape)

        x = F.avg_pool2d(x, 14)
        #print("avgpool", x.shape)

        x = x.view(-1, 512)
        #print("fc", x.shape)

        x = self.output(x)

        return x
```



3. Class Activation Map

How it works and implementation

CAM: Class Activation Mapping

Last Conv Layer

$$f_k(x, y)$$

GAP

$$F_k = \sum_{x,y} f_k(x, y)$$

FC - softmax

$$S_c = \sum_k w_k^c \sum_{x,y} f_k(x, y)$$

CAM: Class Activation Mapping

Last Conv Layer

$$f_k(x, y)$$

GAP

$$F_k = \sum_{x,y} f_k(x, y)$$

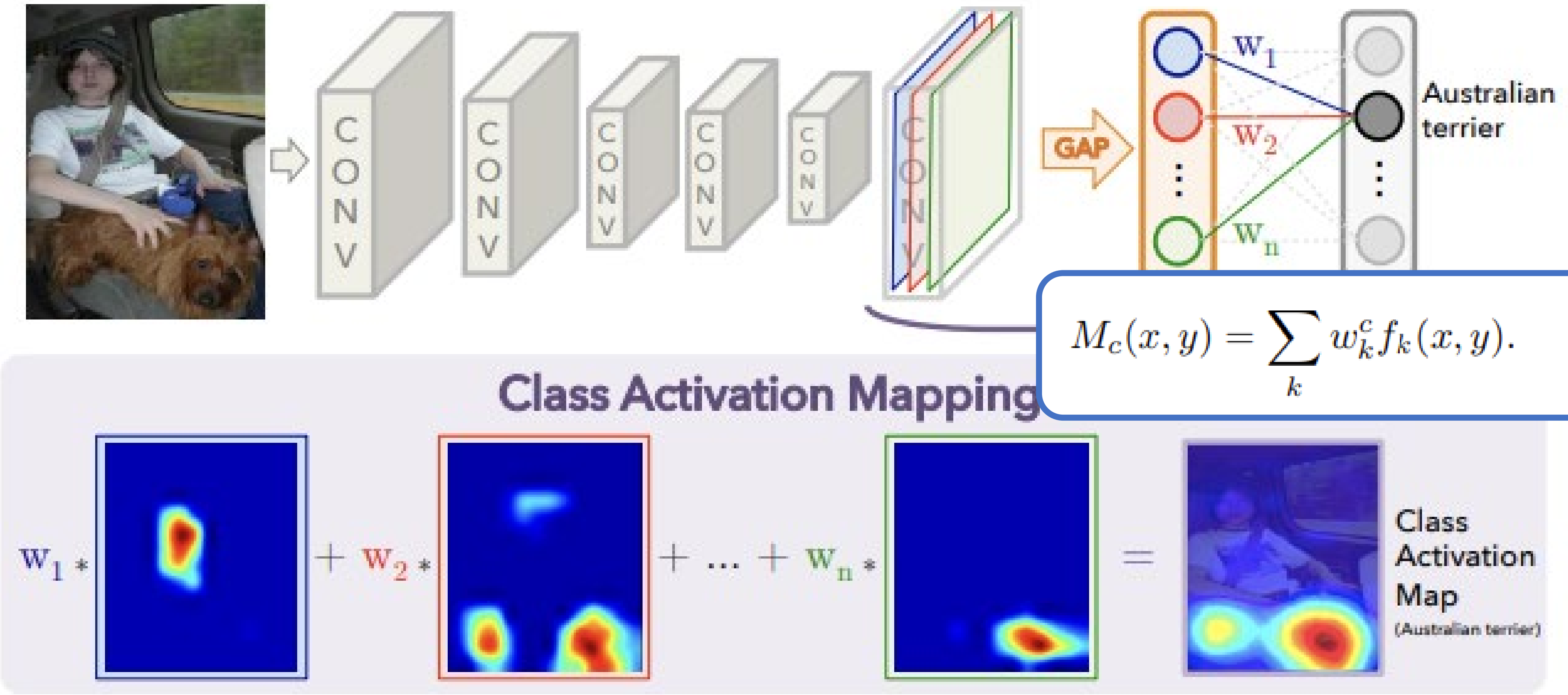
FC - softmax

$$S_c = \sum_k w_k^c \sum_{x,y} f_k(x, y)$$

$$\begin{aligned} S_c &= \sum_k w_k^c \sum_{x,y} f_k(x, y) \\ &= \sum_{x,y} \sum_k w_k^c f_k(x, y). \end{aligned}$$

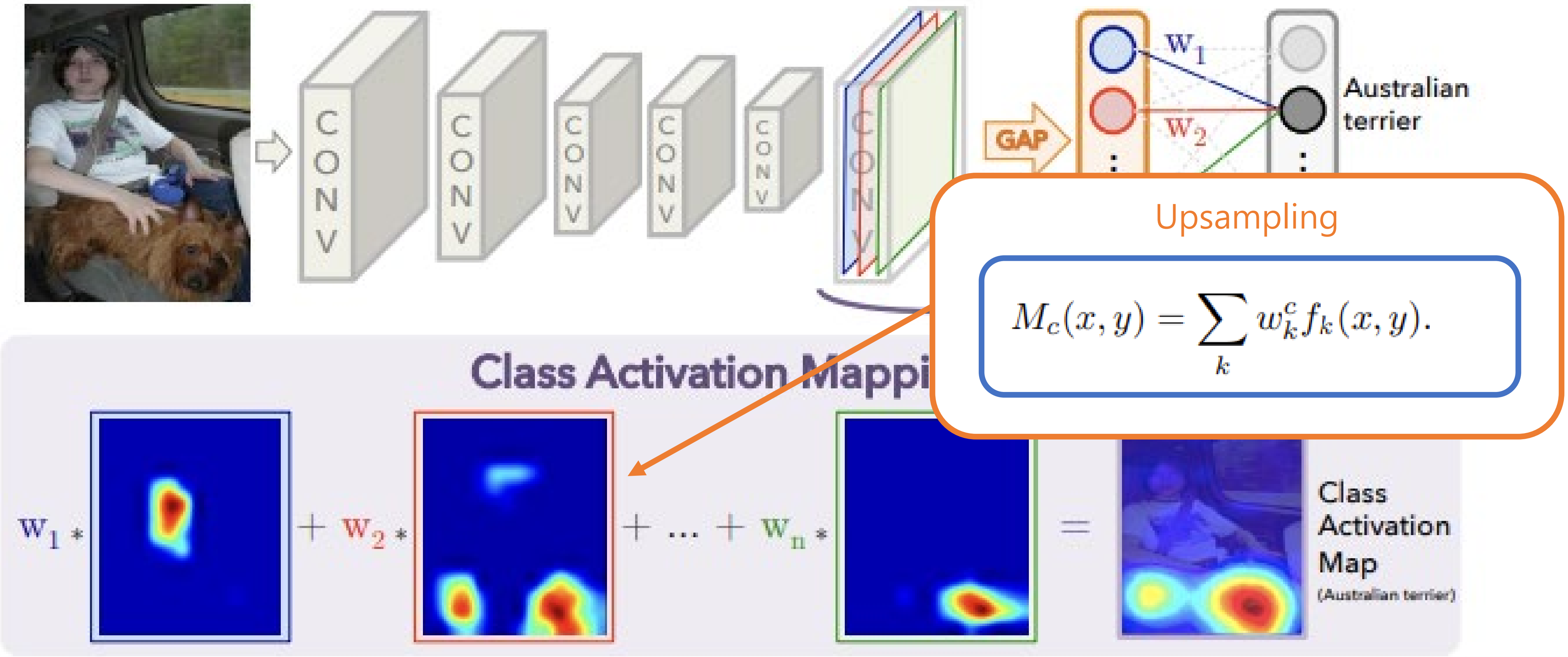
$$M_c(x, y) = \sum_k w_k^c f_k(x, y).$$

CAM: Class Activation Mapping



$$M_c(x,y) = \sum_k w_k^c f_k(x,y).$$

CAM: Class Activation Mapping



Upsampling

$$M_c(x, y) = \sum_k w_k^c f_k(x, y).$$

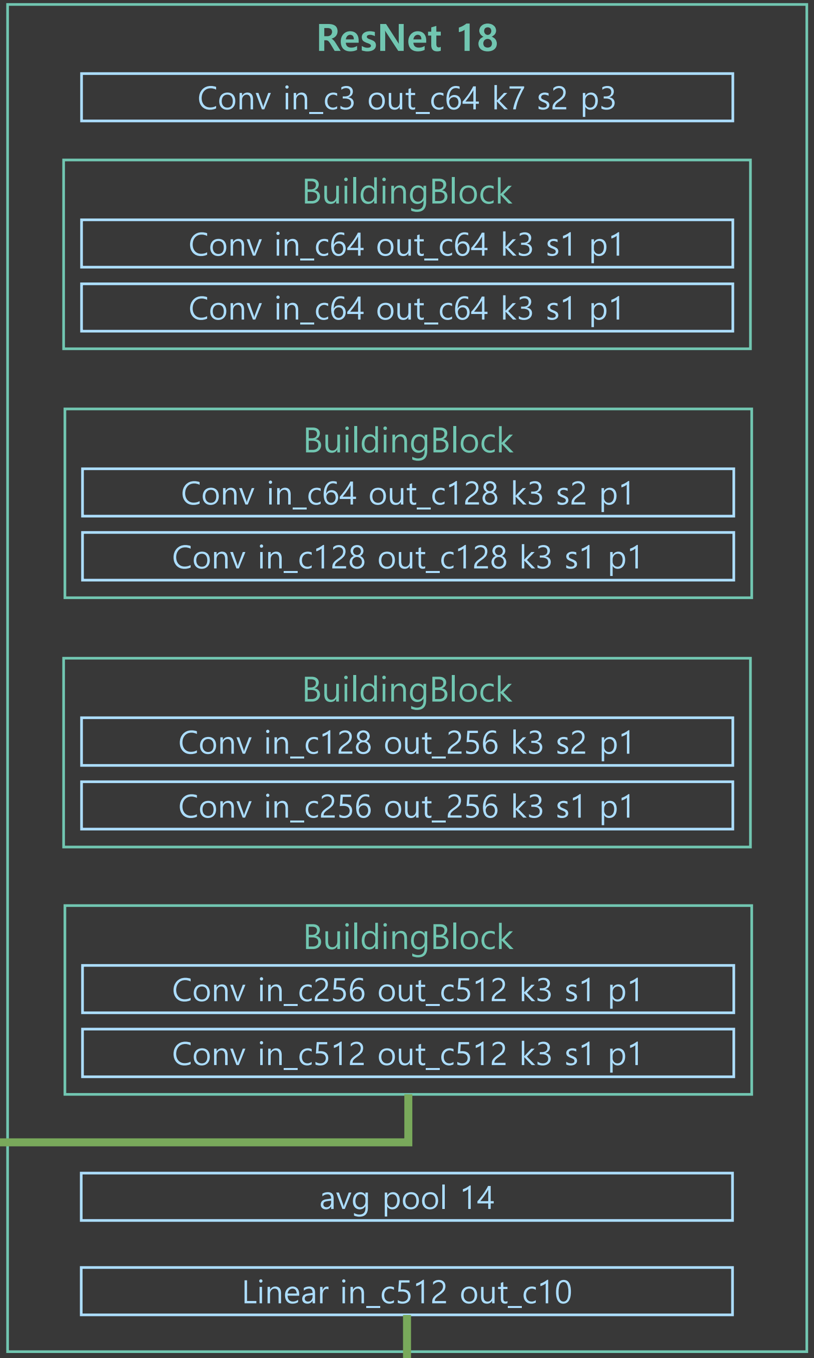
Implementation: CAM

$$M_c(x,y) = \sum_k w_k^c f_k(x,y)$$

1. feature map 512 x 14 x 14
2d matrix 512 x 196

k = 196

2. Weight that leads to classified label
1 x 512



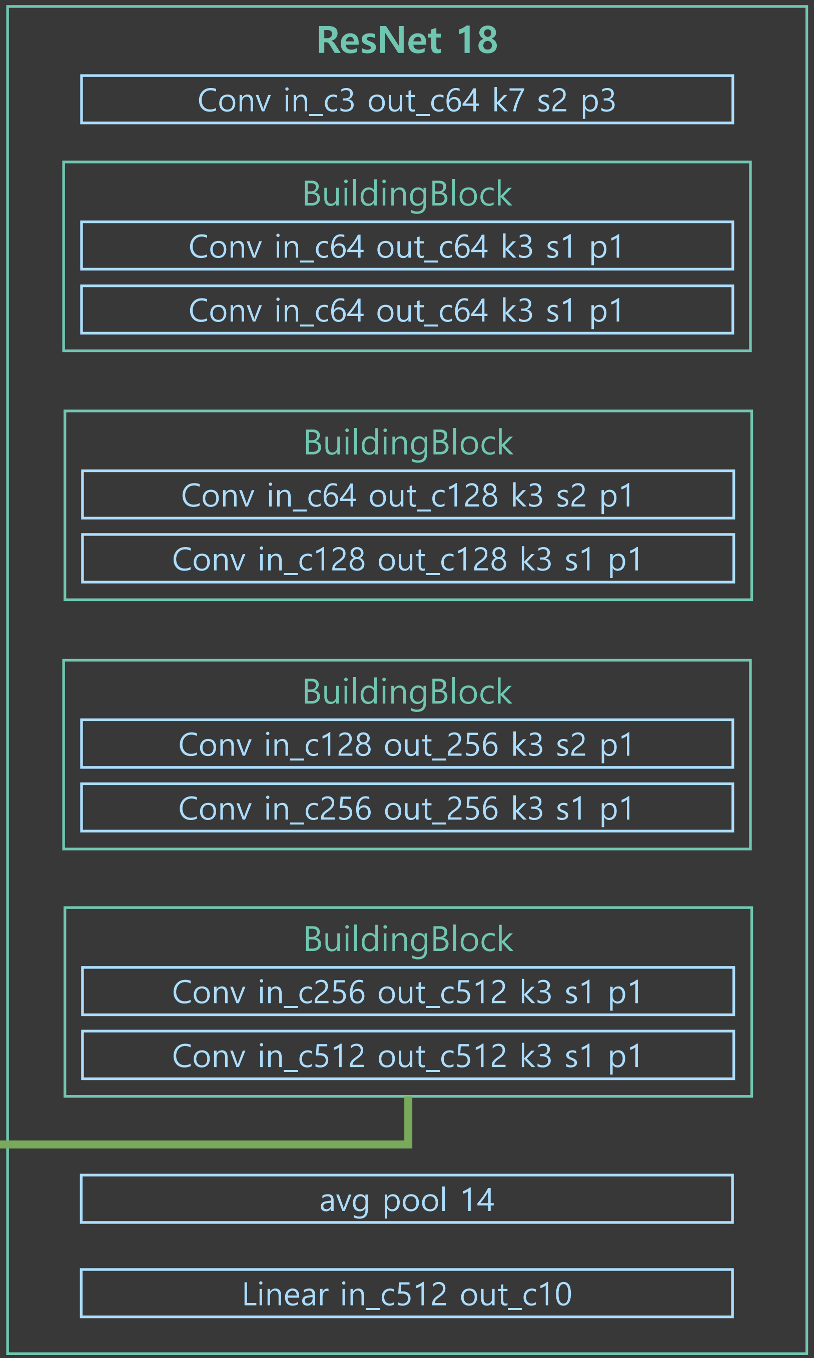
1. Retrieving feature map 512 x 14 x 14

```
[ ]  resnet_front = nn.Sequential(
         resnet.gate,
         resnet.conv2_1,
         resnet.conv2_2,
         resnet.conv3_1,
         resnet.conv3_2,
         resnet.conv4_1,
         resnet.conv4_2,
         resnet.conv5_1,
         resnet.conv5_2
     )
```

```
feature_map = resnet_front(image)
```

```
bz, nc, h, w = feature_map.shape
#print(bz, nc, h, w)

image_matrix_2d = feature_map.reshape((nc, h*w))
# preparing for matrix mul
```

2d matrix 512 x 196
k = 196



2. Retrieve weight that leads to classified label

```
[17]  params = list(ResNet18B().parameters())
      weight = np.squeeze(params[-2].data.numpy())
```

params[-1] = bias for last linear layer

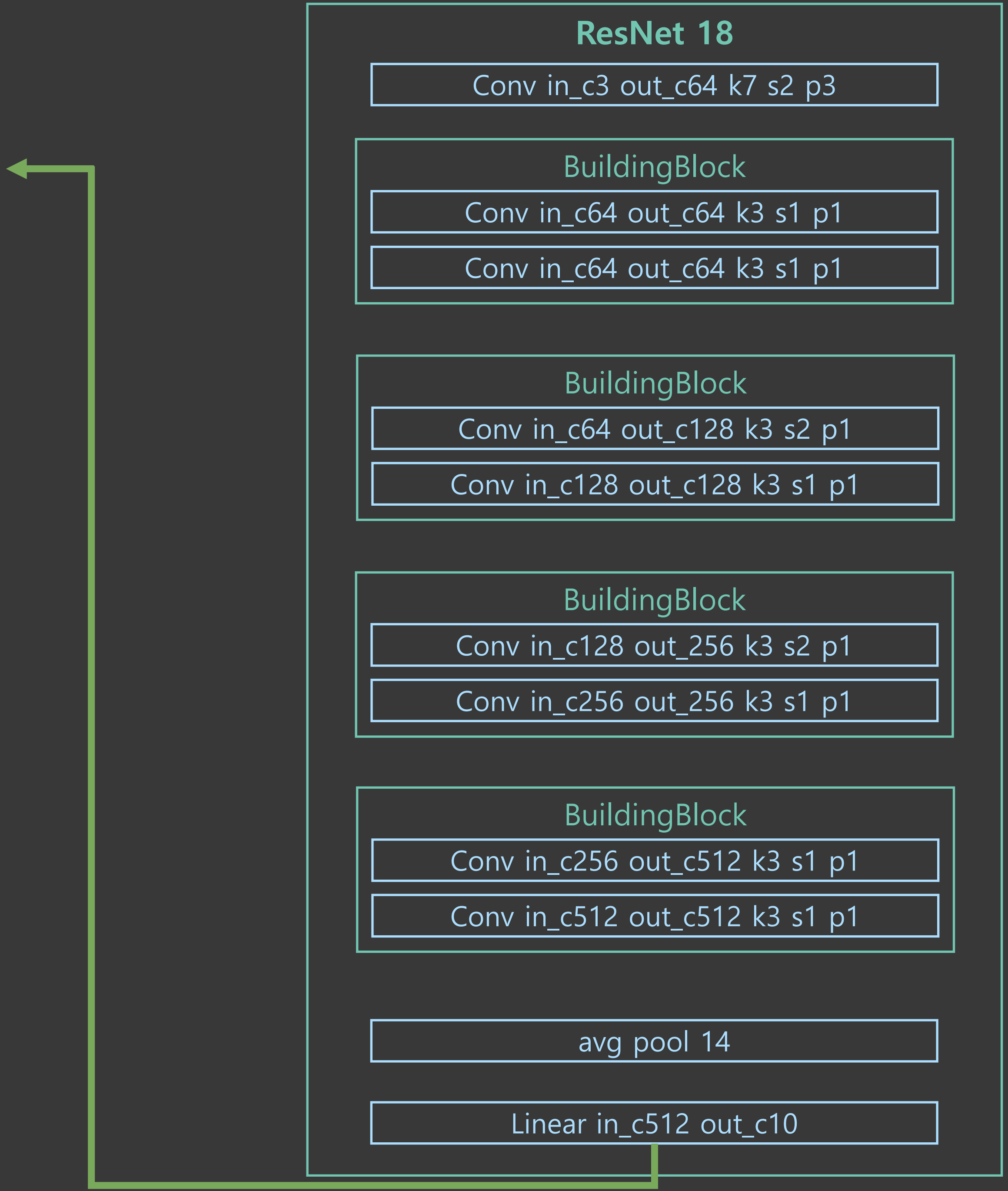
3. CAM calculation

```
cam = np.matmul(weight[idx], image_matrix_2d)
# matrix mul
# 1 512 512 196 = 1 196
```

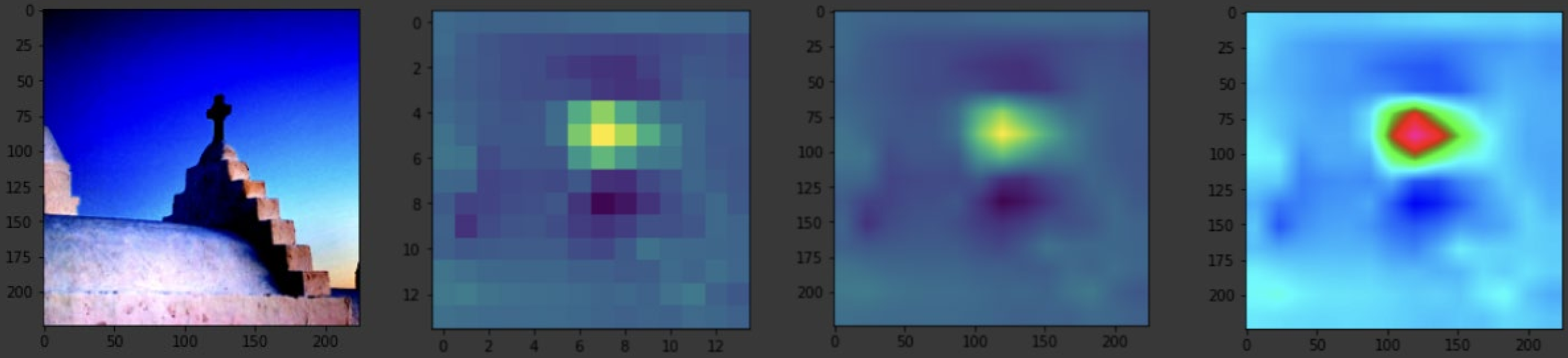
```
# normalization
cam = cam - cam.min()
cam = cam / cam.max()
```

upsample

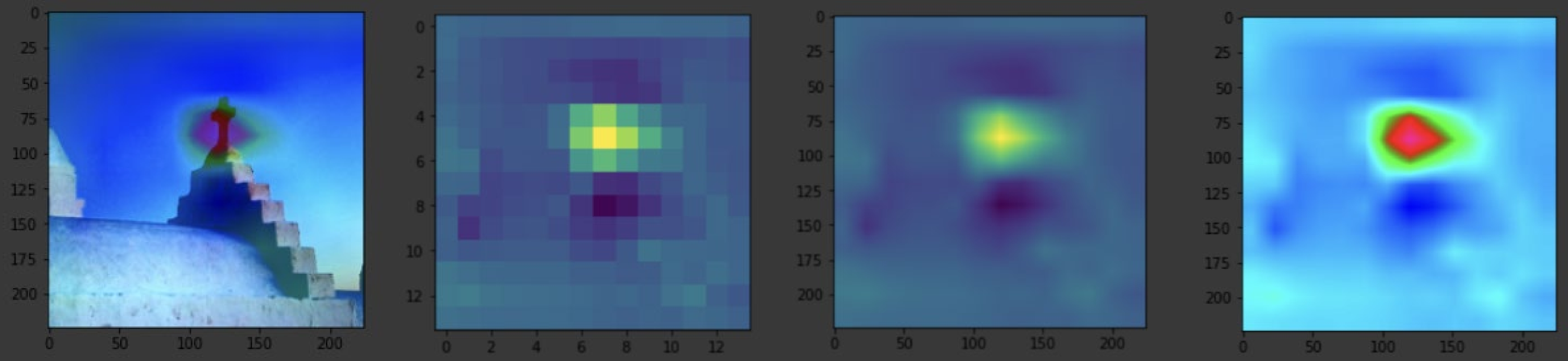
```
cam = np.uint8(255*cam) # necessary step for upsampling
upsampled_cam = cv2.resize(cam, size_upsample)

return upsampled_cam, cam
```



Implementation: Results



Implementation: Results



4. Additionally…

GAP vs GMP

GAP

Encourages the network to identify the **full extent** of the image

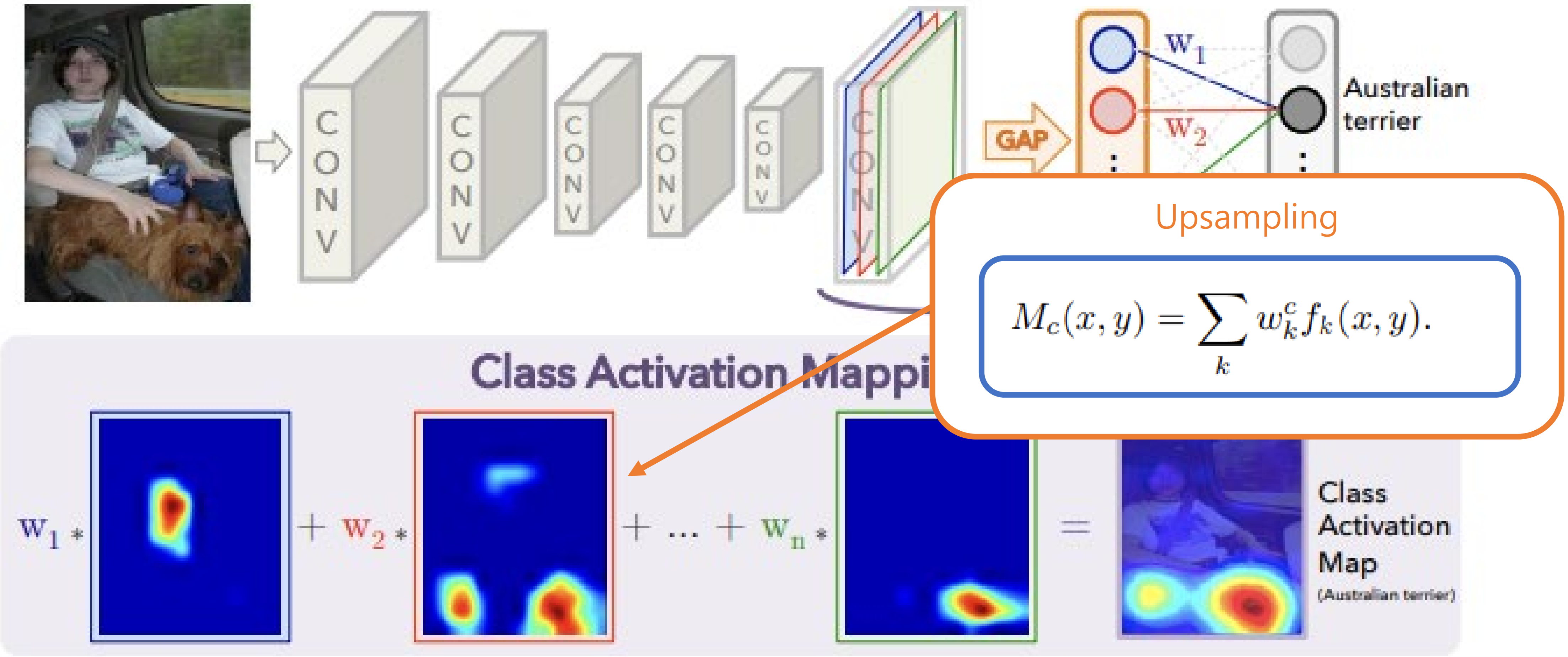
The average of a map is maximized by finding all discriminative part of the object

GMP

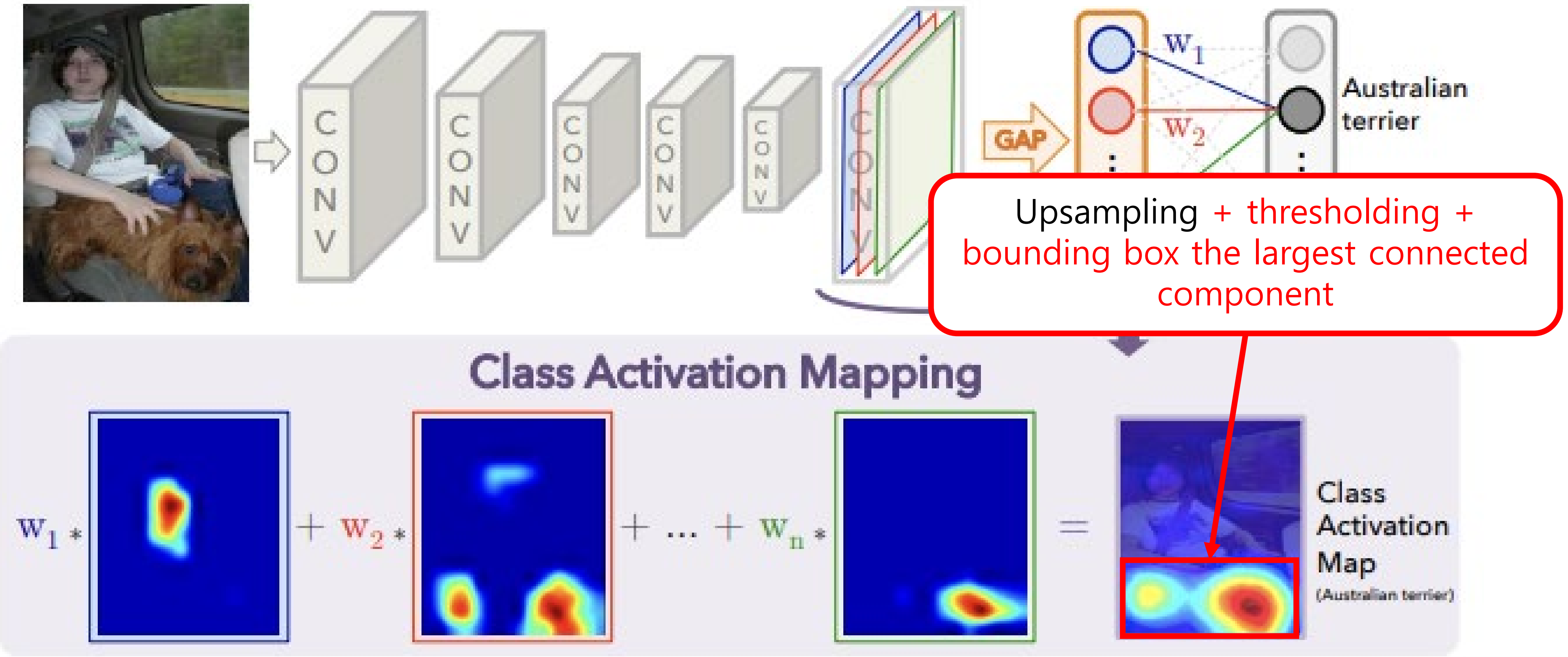
Encourages the network to identify **just one** discriminative part of image

The max of a map is dependent only on the most discriminative part of the object

CAM: Class Activation Mapping



Localization



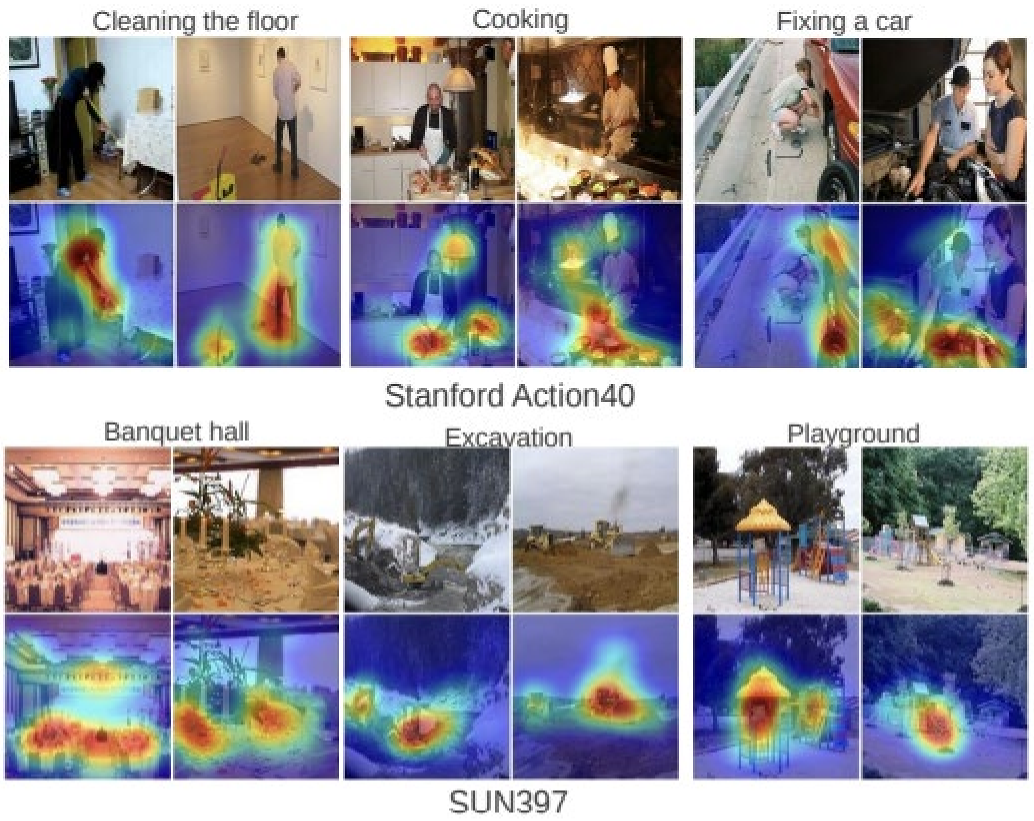
Weakly vs Fully Supervised

Table 3. Localization error on the ILSVRC test set for various weakly- and fully- supervised methods.

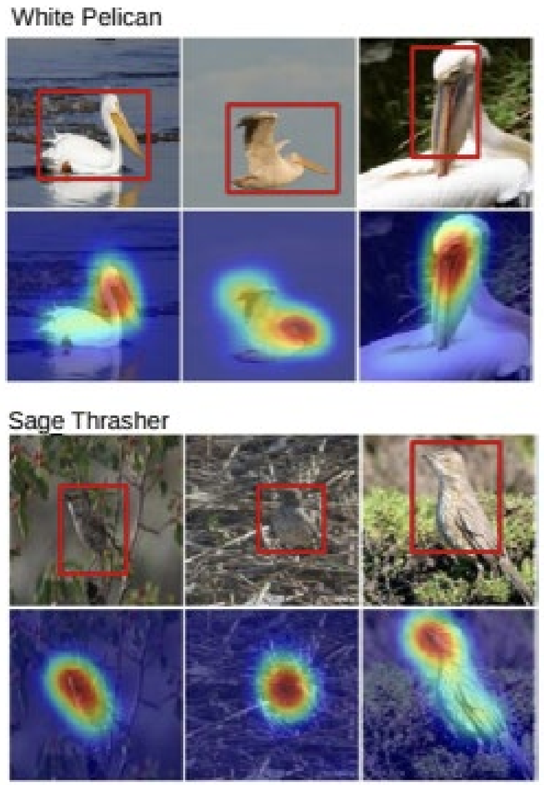
Method	supervision	top-5 test error
GoogLeNet-GAP (heuristics)	weakly	**37.1**
GoogLeNet-GAP	weakly	42.9
Backprop [22]	weakly	46.4
GoogLeNet [24]	full	26.7
OverFeat [21]	full	29.9
AlexNet [24]	full	34.2

And more…

Generating localizable deep features for generic tasks



Fine grained recognition



And more...

Concept localization

Text detector

Visual question answering