BS 6207 ASSIGNMENT 3 - *Tian Siqi, G2101015G*

Github link: https://github.com/SiqiT/Assigmnment

Question 1

1.torch.nn.MaxPool2d(kernel_size=2, stride=1, padding=0, dilation=1, return_indices=False, ceil_mode=False),
2.torch.nn.AvgPool2d(kernel_size=2, stride=1, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None)
Pooling layers are used to reduce the dimensions of the feature maps.

MaxPool2d：

```
diff = torch.sum(torch_out-my_out)
diff
```

```
tensor(-9.3453e-05, dtype=torch.float64)
```

AvgPool2d：

```
diff = torch.sum(torch_out-my_out)
diff
```

```
tensor(-0.0001, dtype=torch.float64)
```

Thus, torch_out and my_out are equal up to small numerical errors.
3.torch.nn.Conv2d(in_channels=3, out_channels=6, kernel_size=3, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')
4.torch.nn.Conv2d(in_channels=3, out_channels=6, kernel_size=5, stride=2, padding=0, dilation=2, groups=1, bias=True, padding_mode='zeros')
This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.

Stride=1:

```
diff = torch.sum(torch_out-torch.from_numpy(my_out))
diff
```

```
tensor(0.0003, dtype=torch.float64, grad_fn=<SumBackward0>)
```

Stide=2:

```
my_out = my_Cov2d_dilation(5,2,2,6,weight,l4.bias,x)
diff = torch.sum(torch_out-torch.from_numpy(my_out))
diff
```

```
tensor(3.2881e-05, dtype=torch.float64, grad_fn=<SumBackward0>)
```

Thus, torch_out and my_out are equal up to small numerical errors.
5.torch.nn.ConvTranspose2d(in_channels=3, out_channels=4, kernel_size=3, stride=1, padding=0, output_padding=0, groups=1, bias=True, dilation=1, padding_mode='zeros')
This module can be seen as the gradient of Conv2d with respect to its input. It is also known as a fractionally-strided convolution or a deconvolution.

```
my_out = my_ConvTranspose2d(3,1,4,l5.weight,l5.bias,x)
diff = torch.sum(torch_out-torch.from_numpy(my_out))
diff
```

```
tensor(-0.0002, dtype=torch.float64, grad_fn=<SumBackward0>)
```

Thus, torch_out and my_out are equal up to small numerical errors.
6.torch.flatten(input, start_dim=0, end_dim=-1)
Flattens input by reshaping it into a one-dimensional tensor.

```
my_out=my_flatten(x)
diff = np.sum(torch_out.numpy()-my_out)
diff
```

```
0.00011458222
```

Thus, torch_out and my_out are equal up to small numerical errors.

7.torch.sigmoid(input, *, out=None)

This is one of the activation function.

```
diff = torch.sum(torch_out-my_out)
diff
```

```
tensor(4.3333e-05)
```

Thus, torch_out and my_out are equal up to small numerical errors.

8.torchvision.ops.roi_pool(input: torch.Tensor, boxes: torch.Tensor, output_size: None, spatial_scale: float = 1.0)

Performs Region of Interest (RoI) Pool operator described in Fast R-CNN.

```
diff = torch.sum(l8-torch.from_numpy(my_out))
diff
```

```
tensor(7.9337e-06, dtype=torch.float64)
```

Thus, torch_out and my_out are equal up to small numerical errors.

9.torch.nn.functional.batch_norm(input, running_mean, running_var, weight=None, bias=None, training=False, momentum=0.1, eps=1e-05)

Applies Batch Normalization for each channel across a batch of data.

```
my_out = my_batch_norm(x,running_mean,running_var)
diff = torch.sum(torch_out-my_out)
diff
```

```
tensor(-0.0005)
```

Thus, torch_out and my_out are equal up to small numerical errors.

10.torch.nn.functional.cross_entropy(input, target, weight=None, size_average=None, ignore_index=-100, reduce=None, reduction='mean')

This criterion computes the cross entropy loss between input and target.

```
l10
```

```
tensor(1.3762)
```

```
my_out
```

```
tensor(1.3762)
```

Thus, torch_out and my_out are equal up to small numerical errors.

11.torch.nn.functional.mse_loss(input, target, size_average=None, reduce=None, reduction='mean')

Measures the element-wise mean squared error.

```
tensor(1.9804)
```

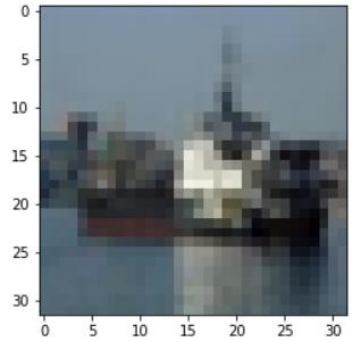
```
my_out = torch.mean((x-target)**2)
my_out
```

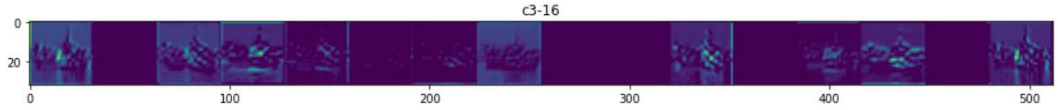
```
tensor(1.9804)
```

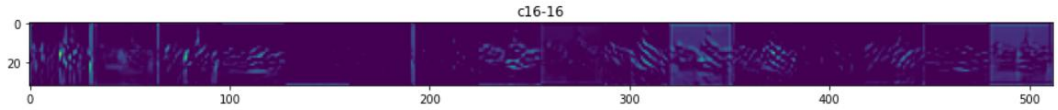
Thus, torch_out and my_out are equal up to small numerical errors.

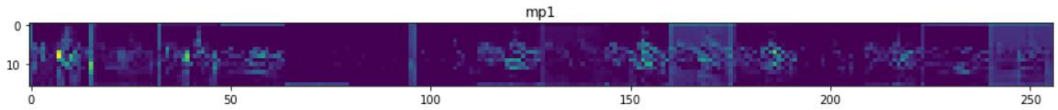
Question 2

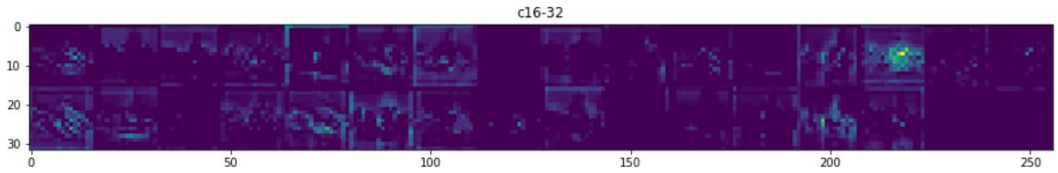
A. No batch normalization
For a single image data

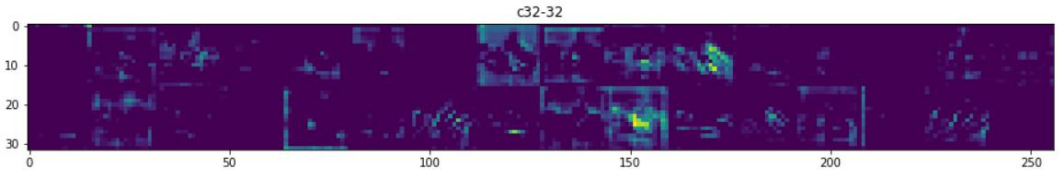


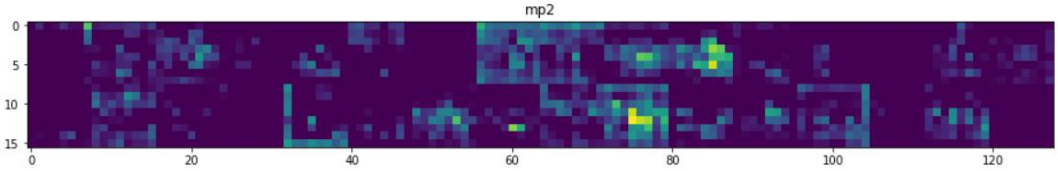


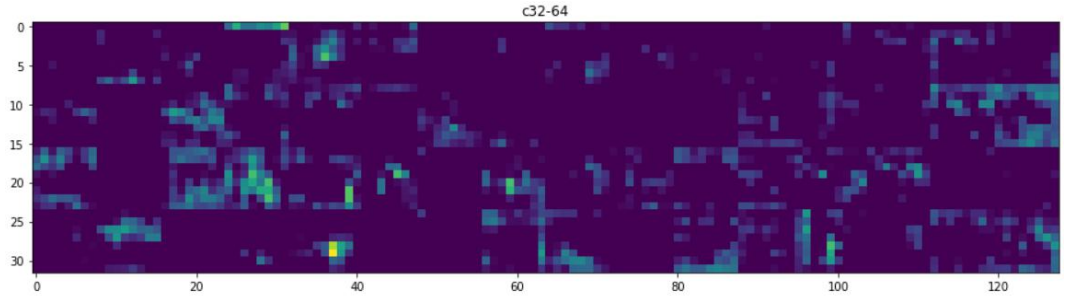


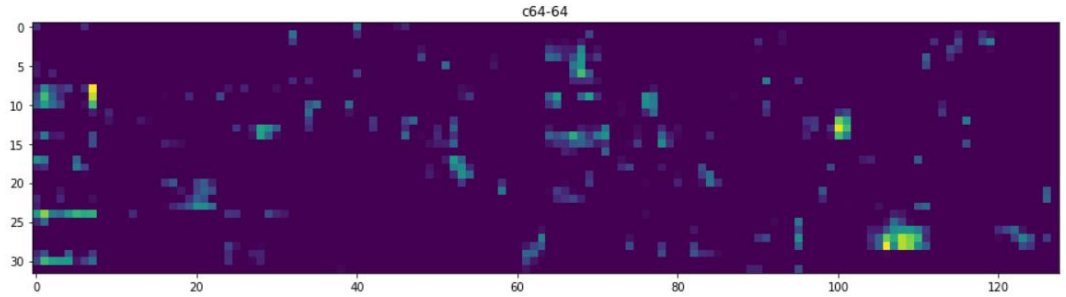


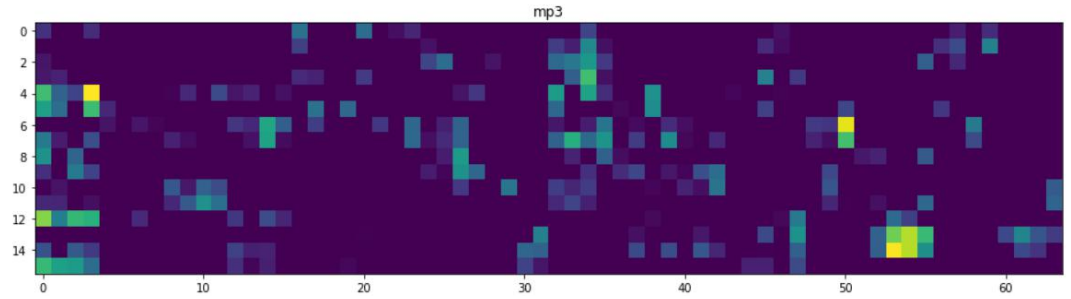


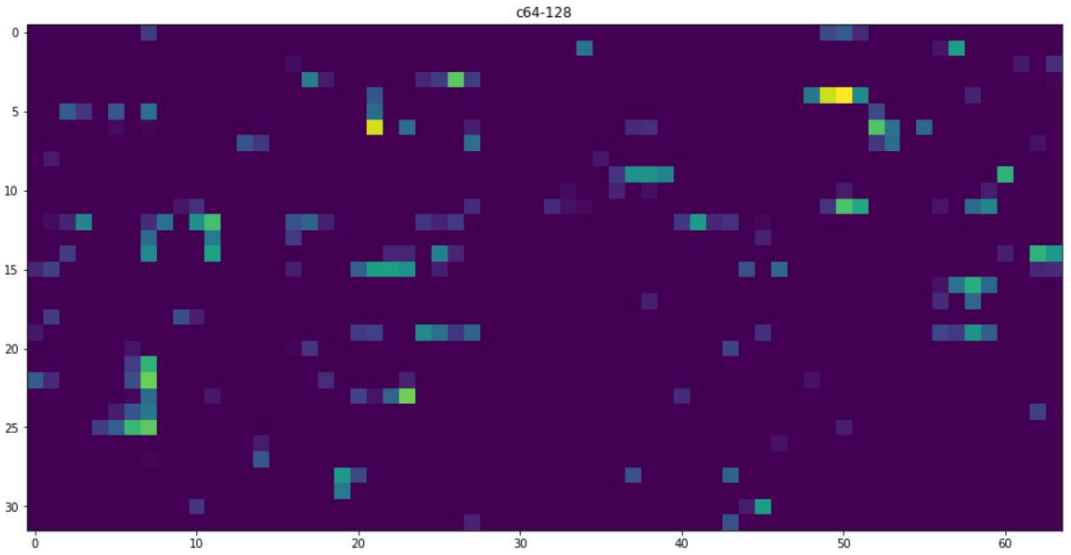


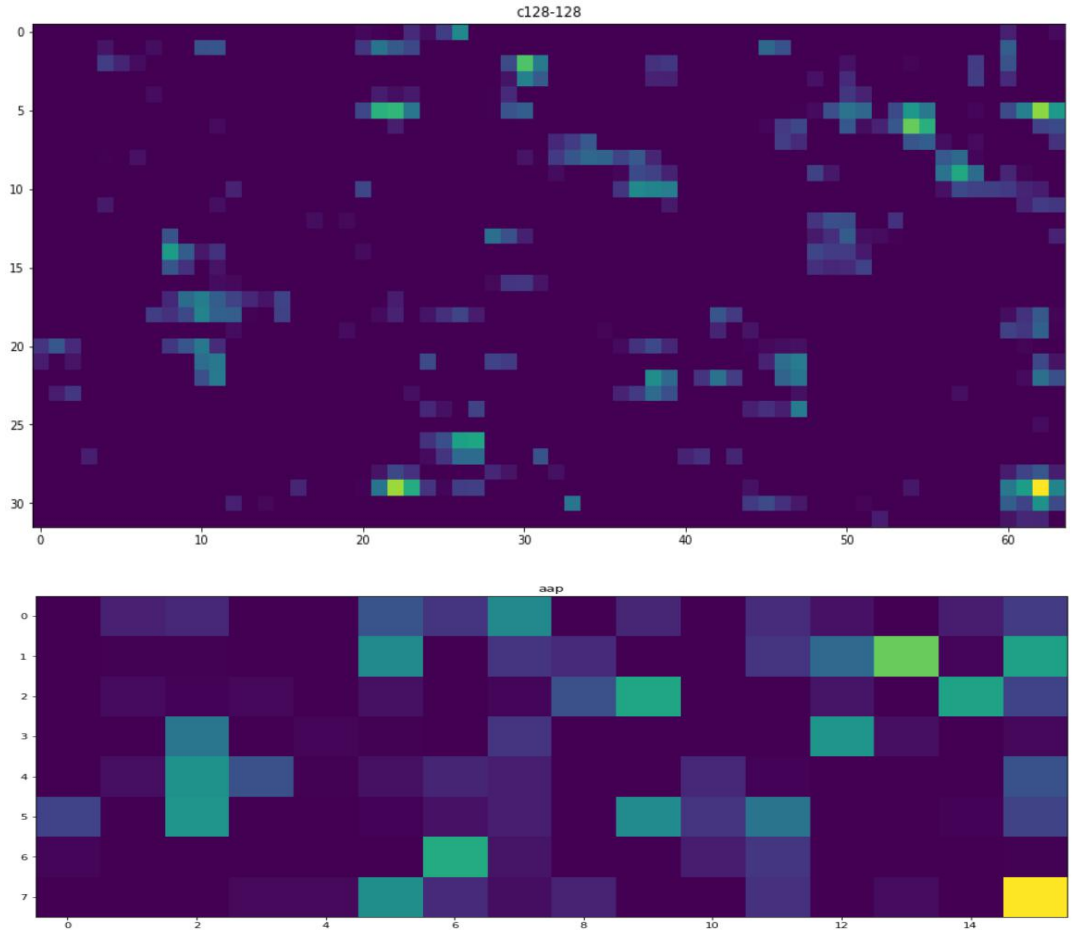



c32-64
0
5
10
15
20
25
30
0
20
40
60
80
100
120

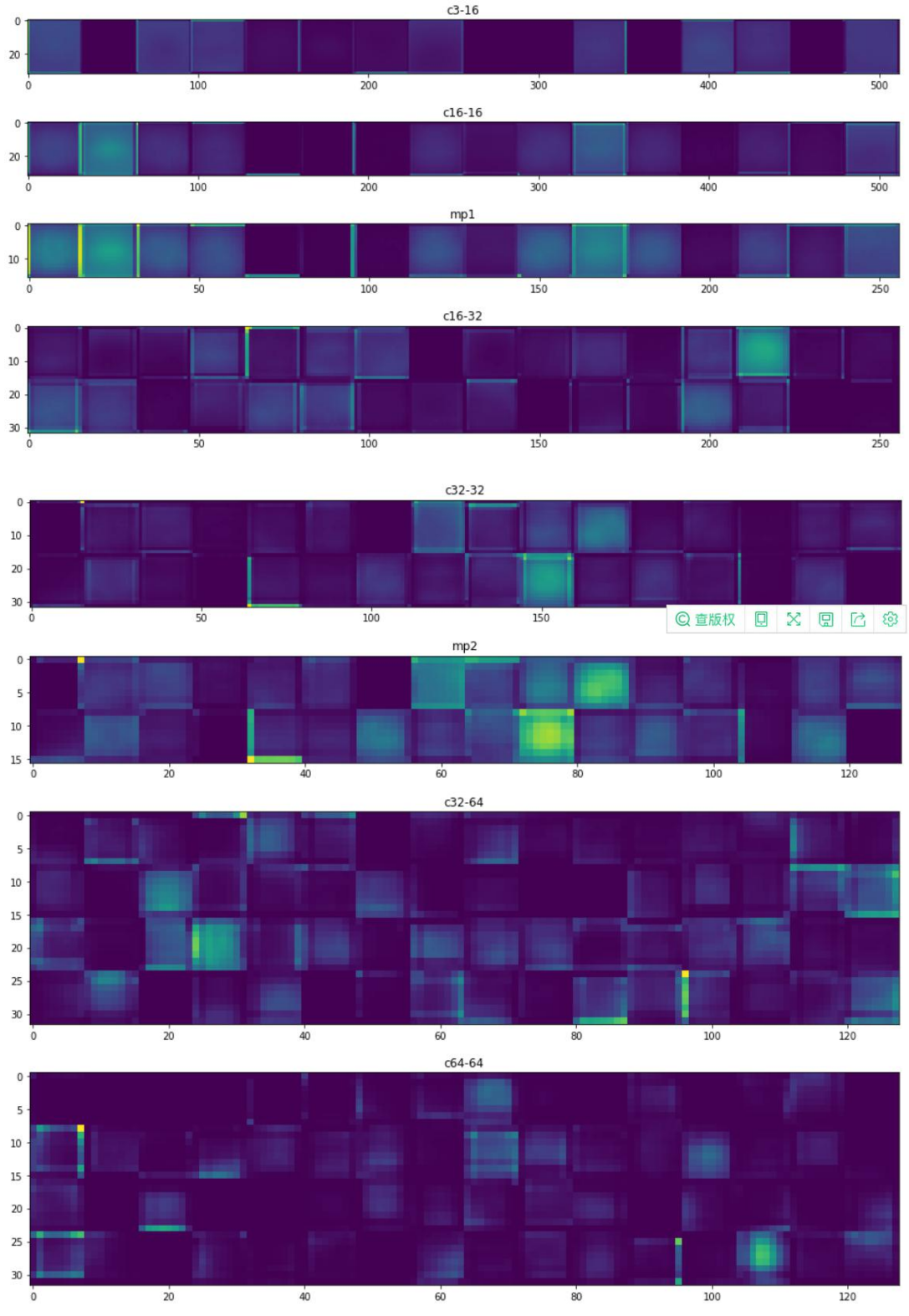

c64-64
0
5
10
15
20
25
30
0
20
40
60
80
100
120

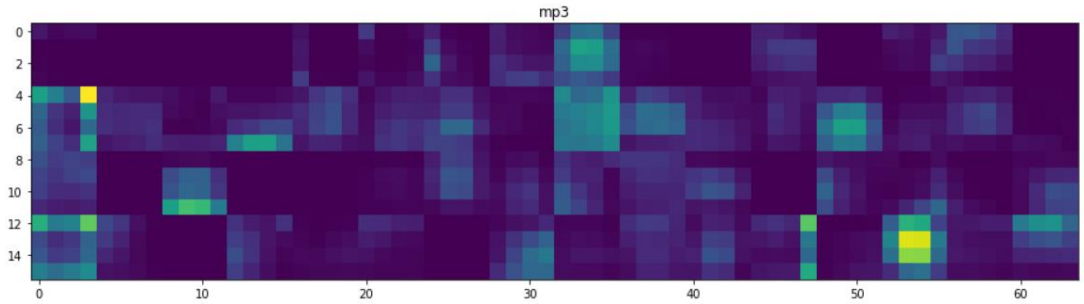

mp3
0
2
4
6
8
10
12
14
0
10
20
30
40
50
60

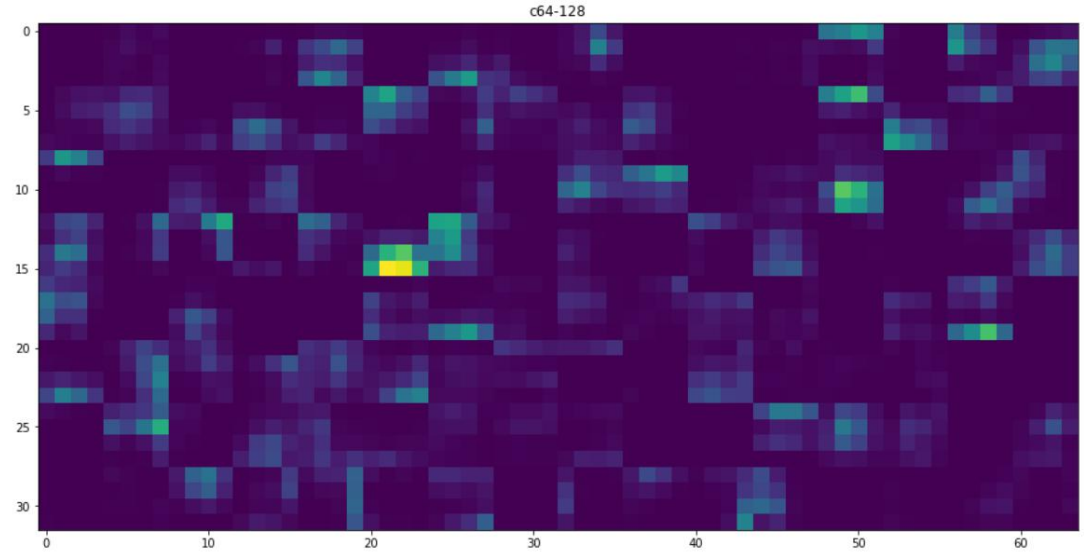

c64-128
0
5
10
15
20
25
30
0
10
20
30
40
50
60

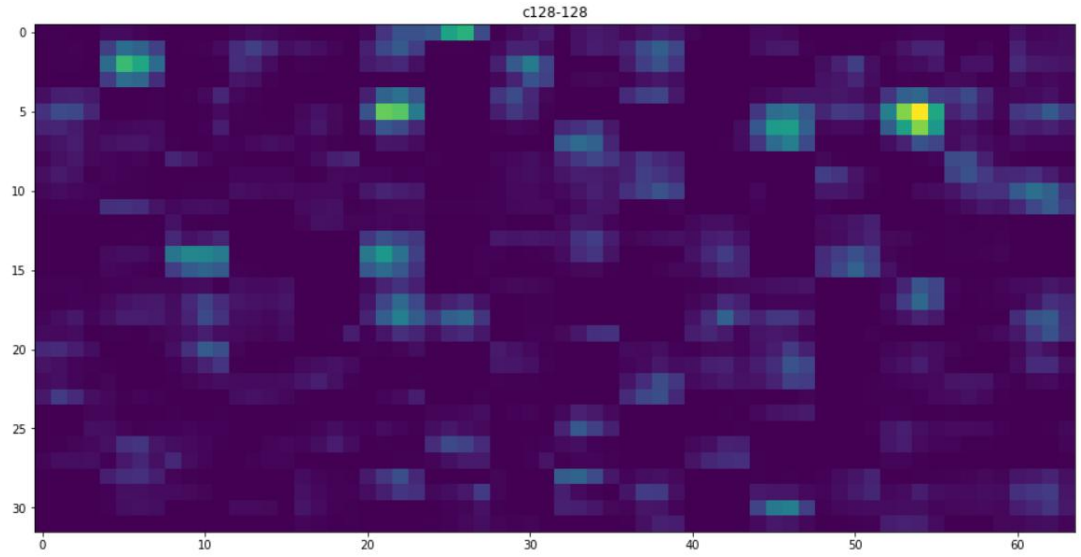


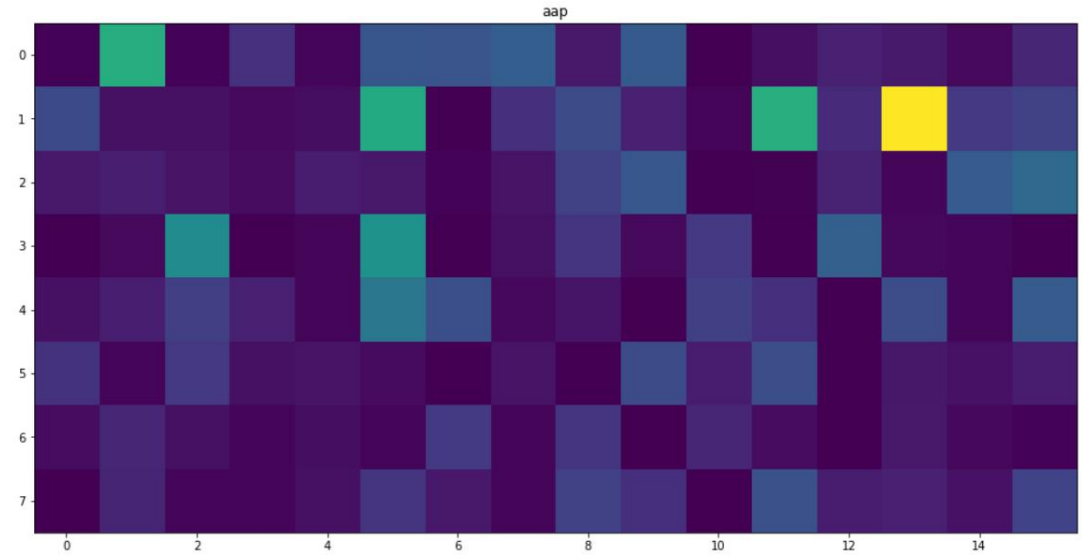
Observation: Each channel encodes relatively independent features, some focus on the edge, some focus on a single border, etc, we can see this from the first layer's activation.
In the first layer, it retains almost the full shape of the image, it contains the full image of the original image.
As it goes deeper, the activation is harder for us to understand, they begin to encode higher-level concepts, this increasingly less information about the visual contents of the image, and increasingly more information related to the class of the image.
(ii)For 10000 training image data(can't use all, the kernel would crush)


c3-16
c16-16
mp1
c16-32
c32-32
mp2
c32-64
c64-64


mp3
0
2
4
6
8
10
12
14
0
10
20
30
40
50
60


c64-128
0
5
10
15
20
25
30
0
10
20
30
40
50
60

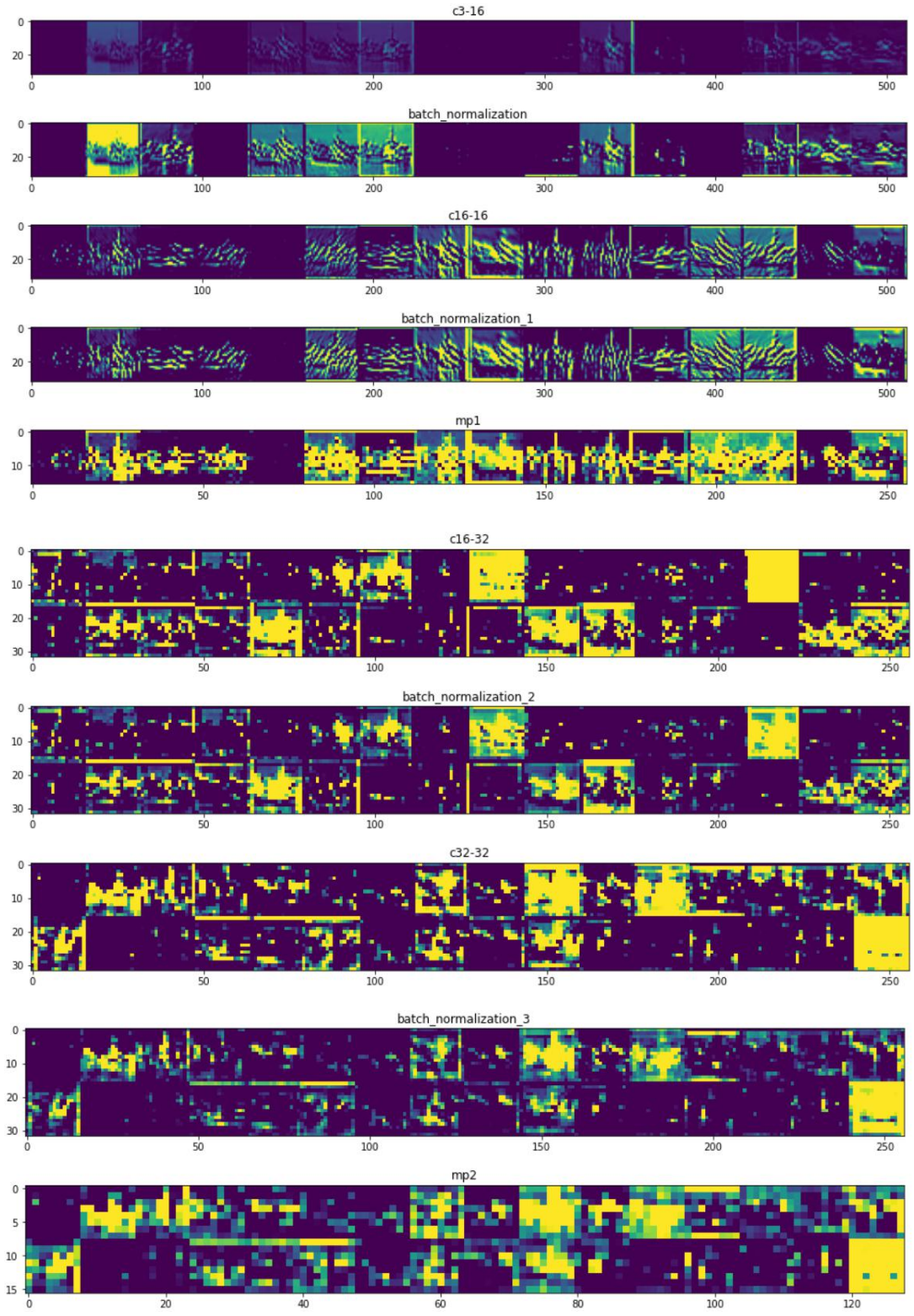


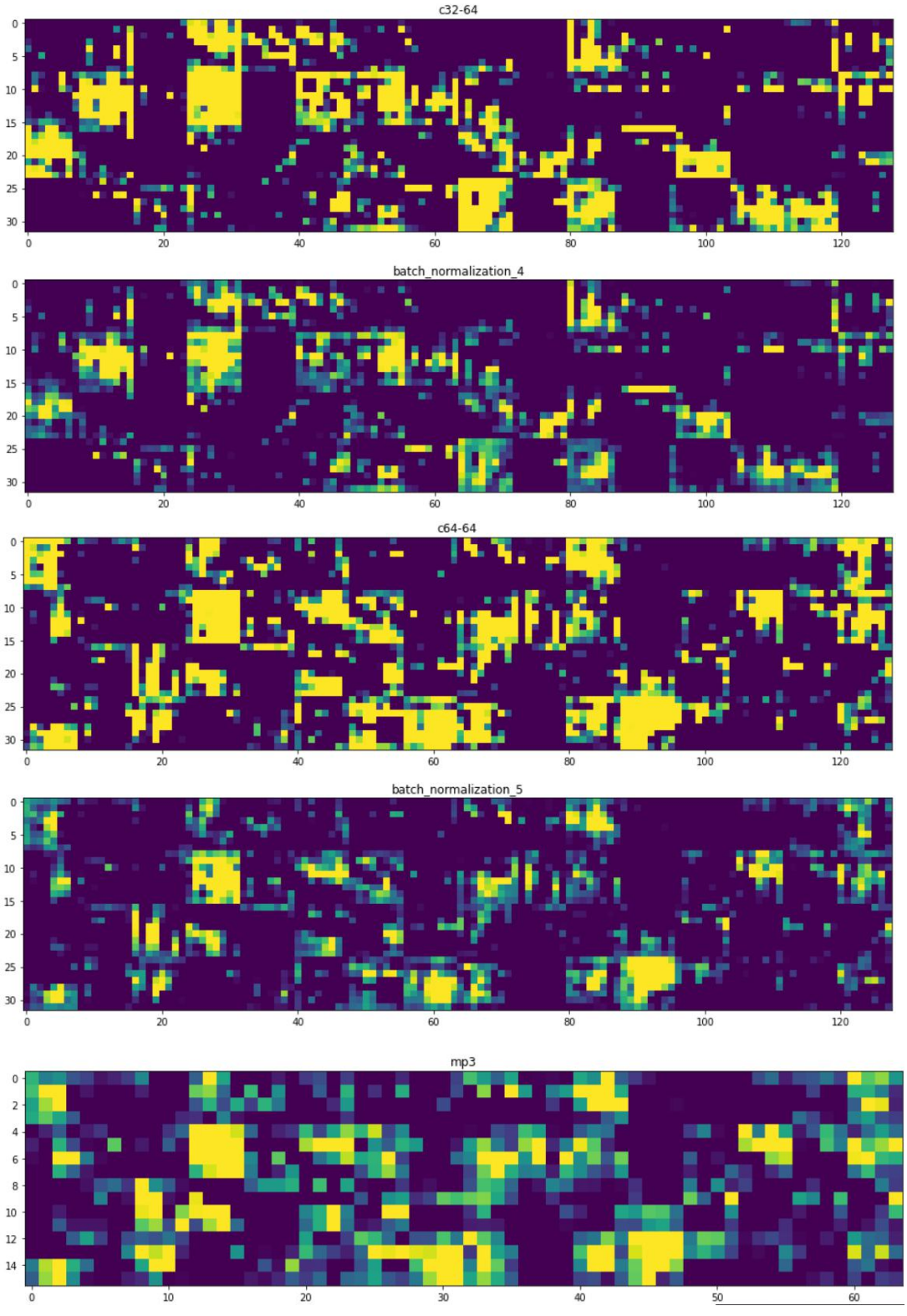


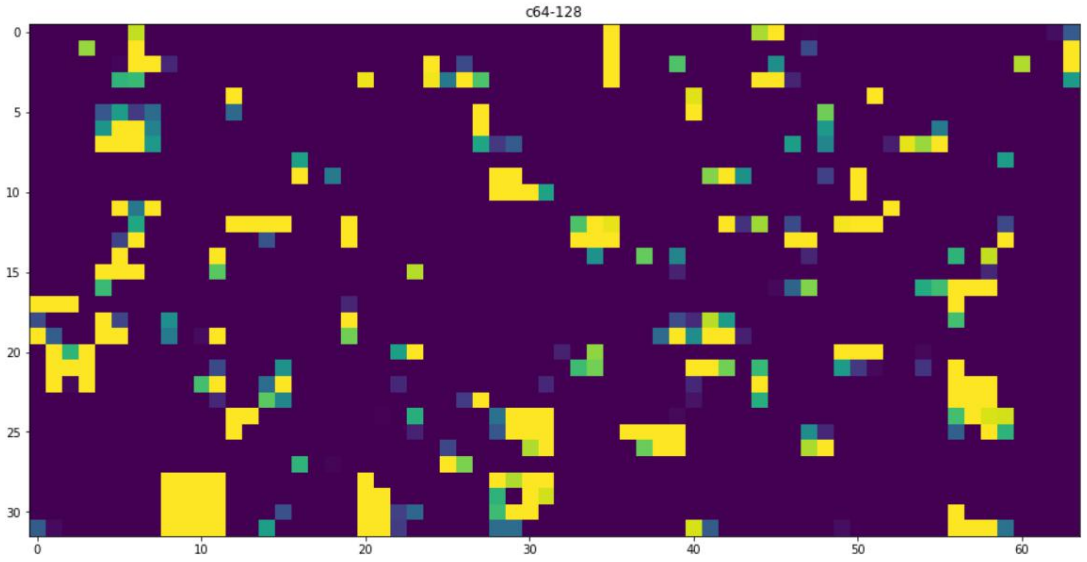
When we average many training data into the picture of activation for layers, we can’ t tell images information, but we can see different channels still focus on a different character for the image data, and when the layer goes deeper, the features are more abstract.

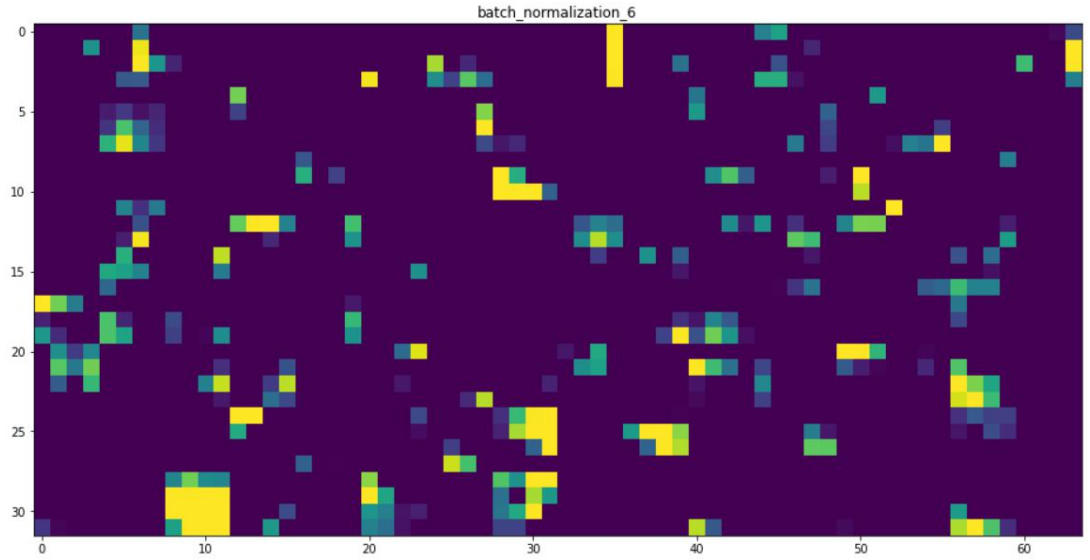
B. Batch normalization

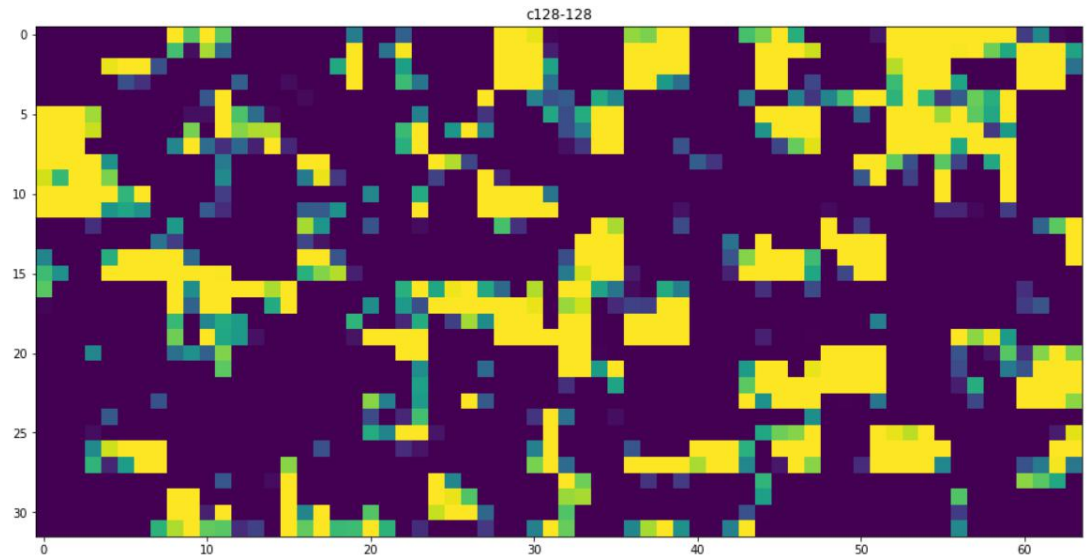
For a single image data

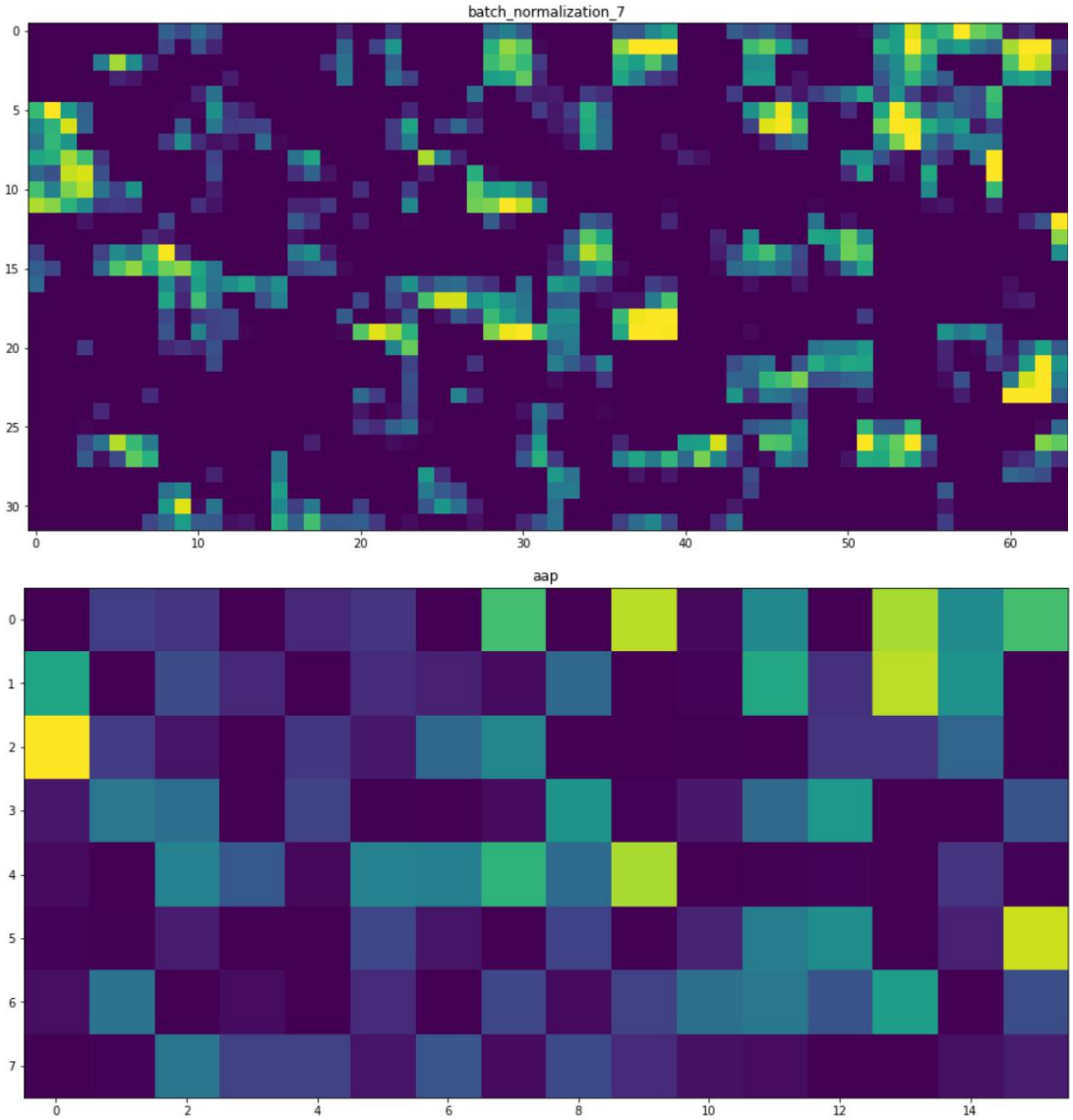



c32-64
batch_normalization_4
c64-64
batch_normalization_5
mp3

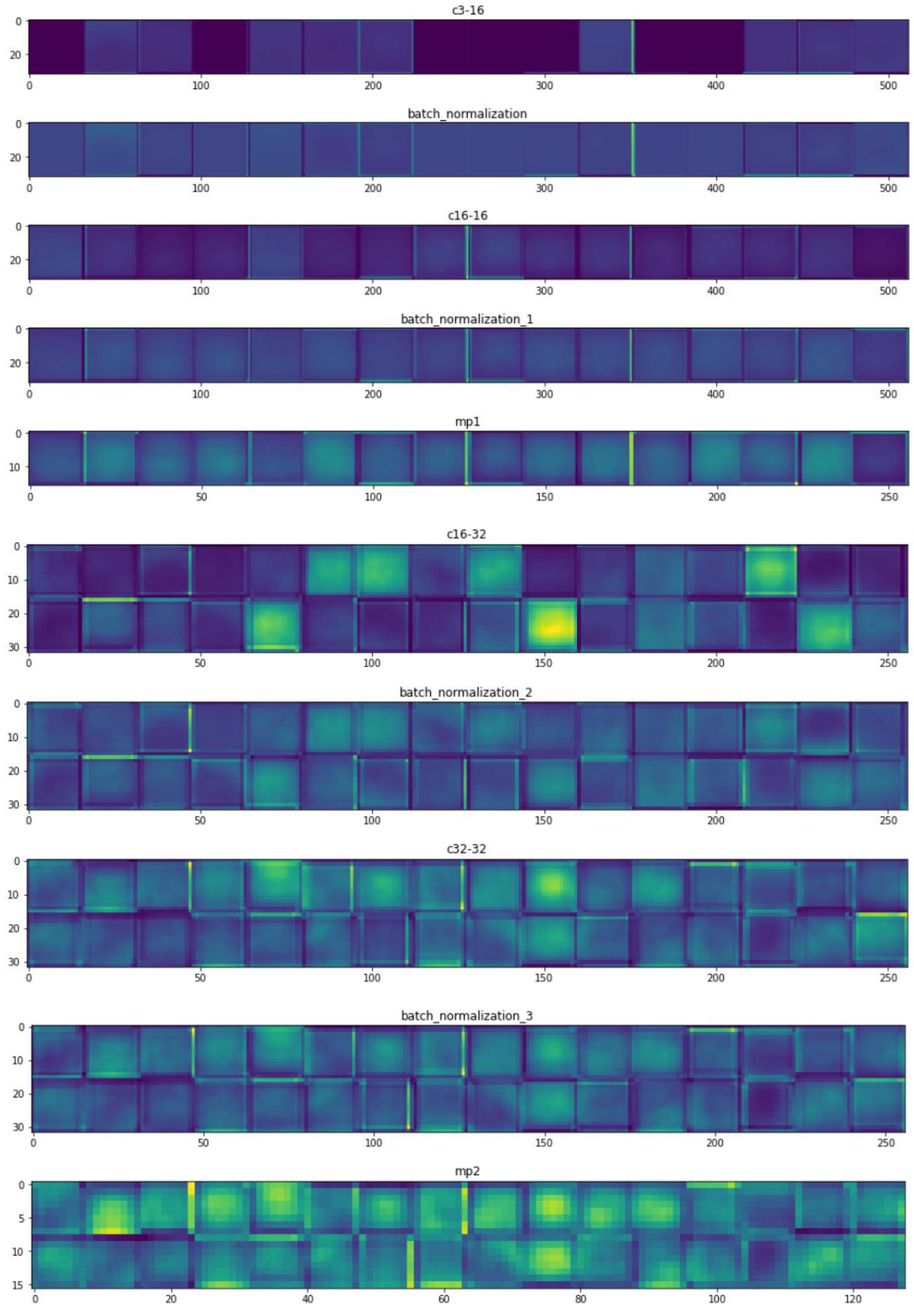

c64-128
0
5
10
15
20
25
30
0
10
20
30
40
50
60

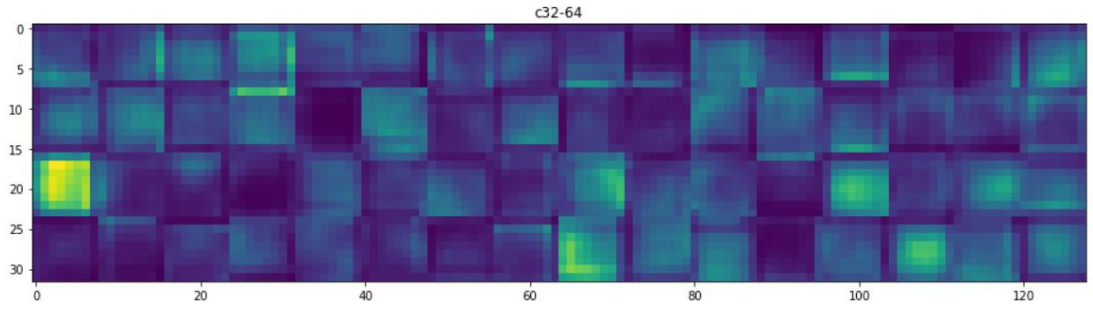

batch_normalization_6
0
5
10
15
20
25
30
0
10
20
30
40
50
60

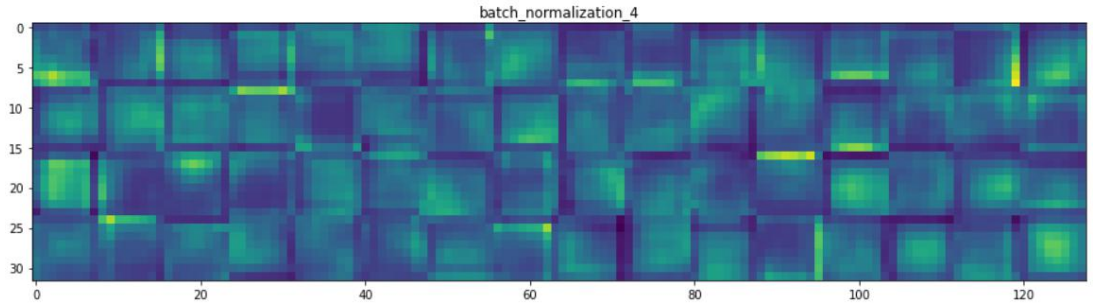

c128-128
0
5
10
15
20
25
30
0
10
20
30
40
50
60

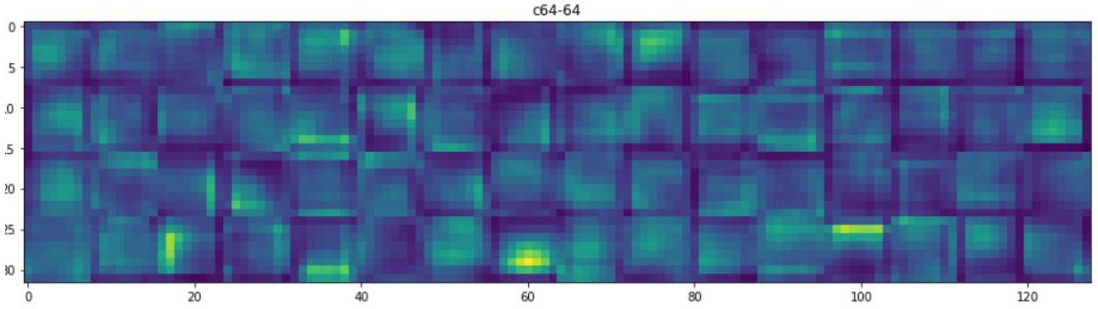


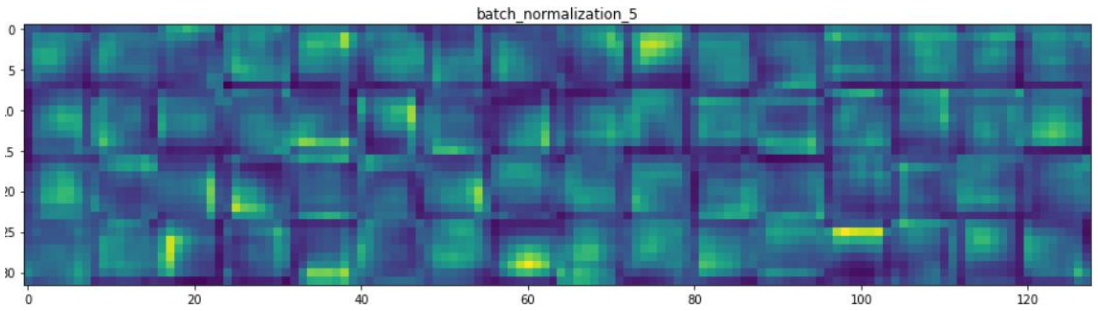
(ii)For 10000 training image data(can’t use all, the kernel would crush)

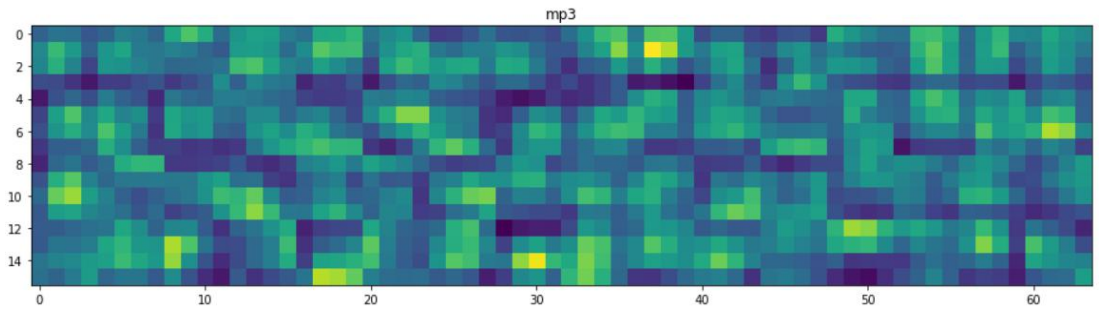

c3-16
batch_normalization
c16-16
batch_normalization_1
mp1
c16-32
batch_normalization_2
c32-32
batch_normalization_3
mp2

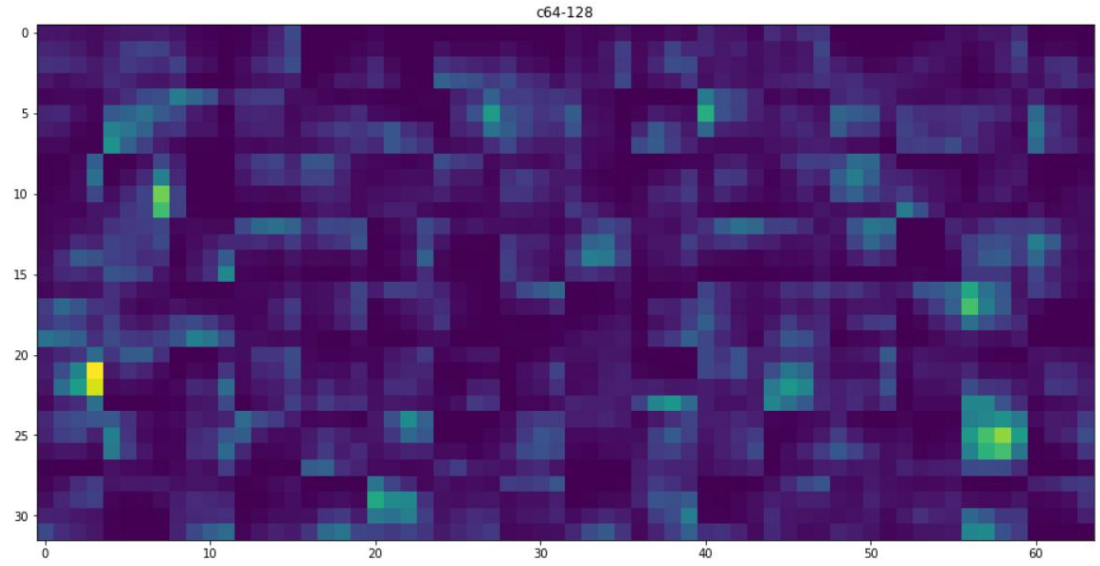

c32-64

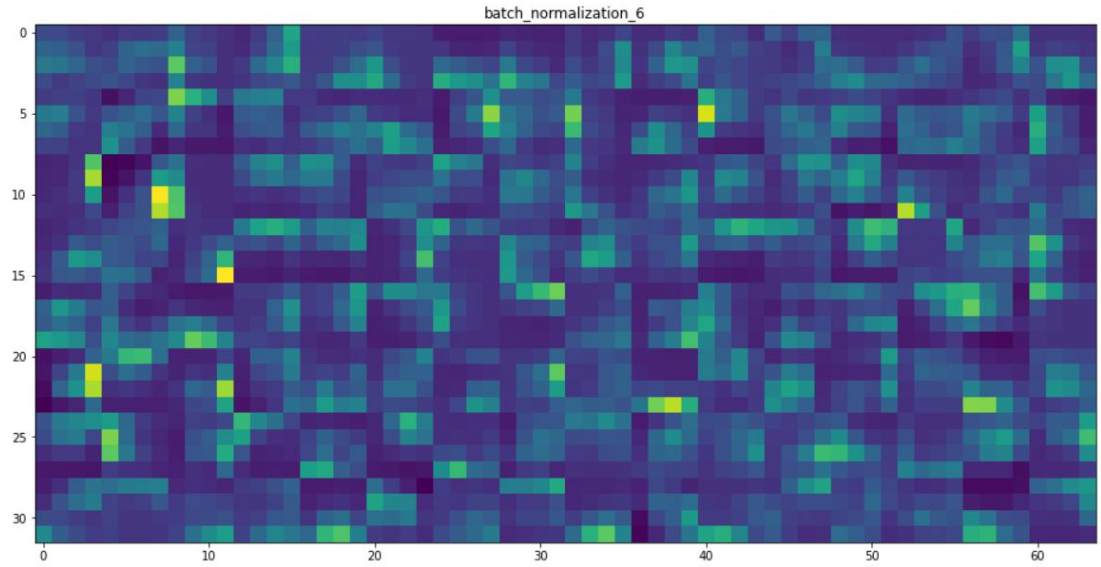

batch_normalization_4

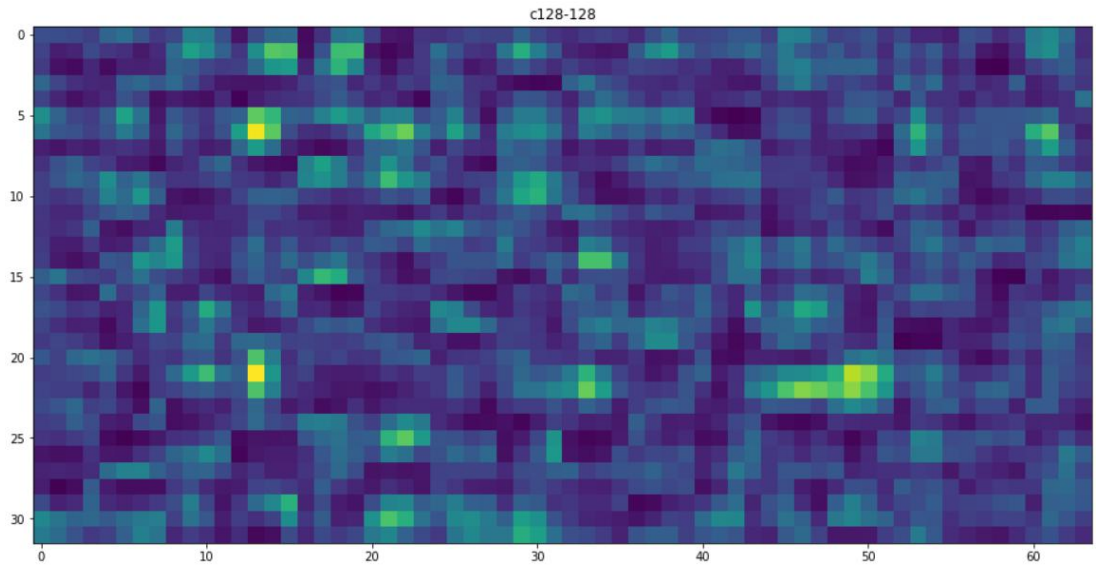

c64-64

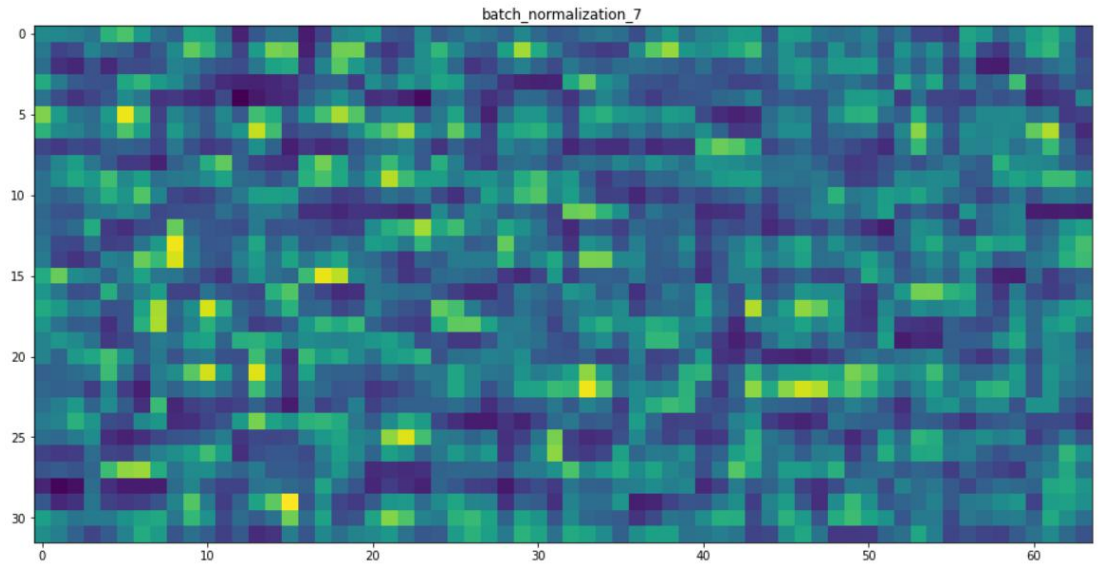

batch_normalization_5

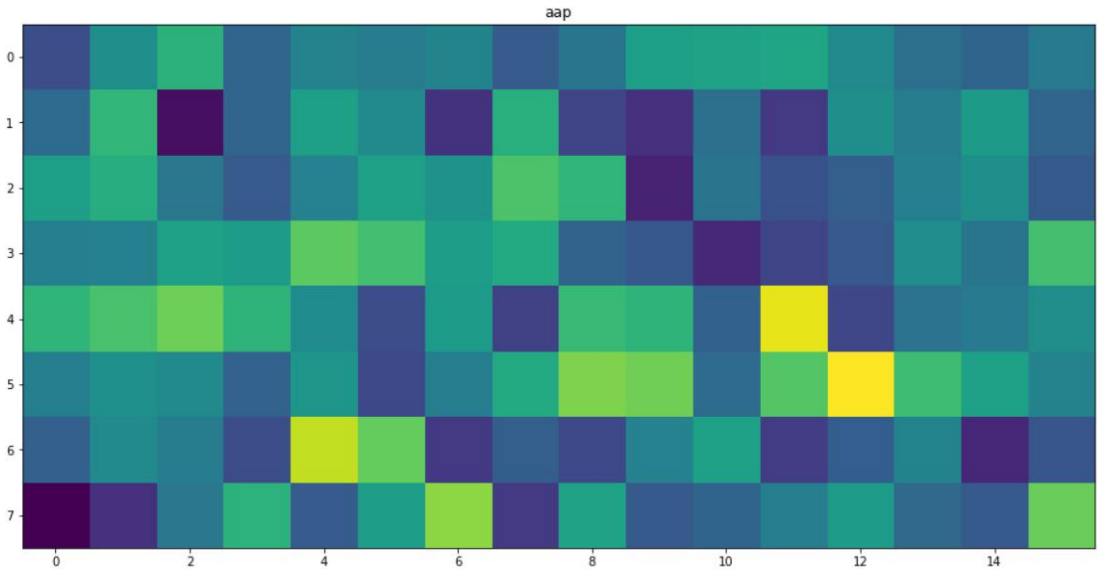

mp3


c64-128
0
5
10
15
20
25
30
0
10
20
30
40
50
60


batch_normalization_6
0
5
10
15
20
25
30
0
10
20
30
40
50
60


c128-128
0
5
10
15
20
25
30
0
10
20
30
40
50
60





Comparing the result from Q2A and Q2B, we can find batch normalization can standardize the inputs to a layer for each mini-batch, It achieves a **stable distribution of activation** values throughout training. This has the effect of stabilizing and speeding-up the training process of deep neural networks.