Model Training

Name: Qian Ren

StudentID: 1901203093

At first, the impacts of different optimizer algorithms on neural network should be tested. So, I had trained **resnet18** model with two different optimizers(**SGD and Adam**), and shows their influence in converging during training in section 2. The trend of accuracy of Adam grows fast at beginning then slows down increasing speed during training while the trend of accuracy of SGD is almost linearly increasing.

As a result, my model got 93.022% accuracy for training set, and 71.120% for test set.

1. My model

Here is the structure of my model.

```
[Sequential(
  (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), 
bias=False)
  (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, 
track_running_stats=True)
  (2): ReLU()
), Sequential(
  (0): ResidualBlock(
    (left): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), 
bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, 
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), 
bias=False)
      (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, 
track_running_stats=True)
    )
    (shortcut): Sequential(
      (0): Conv2d(64, 64, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, 
track_running_stats=True)
    )
  )
  (1): ResidualBlock(
    (left): Sequential(
      (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), 
bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, 
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), 
bias=False)
```

```
      (4): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (shortcut): Sequential()
  )
), Sequential(
  (0): ResidualBlock(
    (left): Sequential(
      (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (shortcut): Sequential(
      (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): ResidualBlock(
    (left): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (shortcut): Sequential()
  )
), Sequential(
  (0): ResidualBlock(
    (left): Sequential(
      (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (shortcut): Sequential(
      (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): ResidualBlock(
    (left): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (shortcut): Sequential()
  )
), Sequential(
  (0): ResidualBlock(
    (left): Sequential(
      (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (shortcut): Sequential(
      (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (1): ResidualBlock(
    (left): Sequential(
      (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (2): ReLU(inplace=True)
      (3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (shortcut): Sequential()
  )
), Linear(in_features=512, out_features=10, bias=True)]
```

2. Training process

Consider characteristic of SGD and Adam, Adam was applied in former 6 epochs then SGD took it over for trade-off between speed and accuracy.

Figure 1. Training with Adam

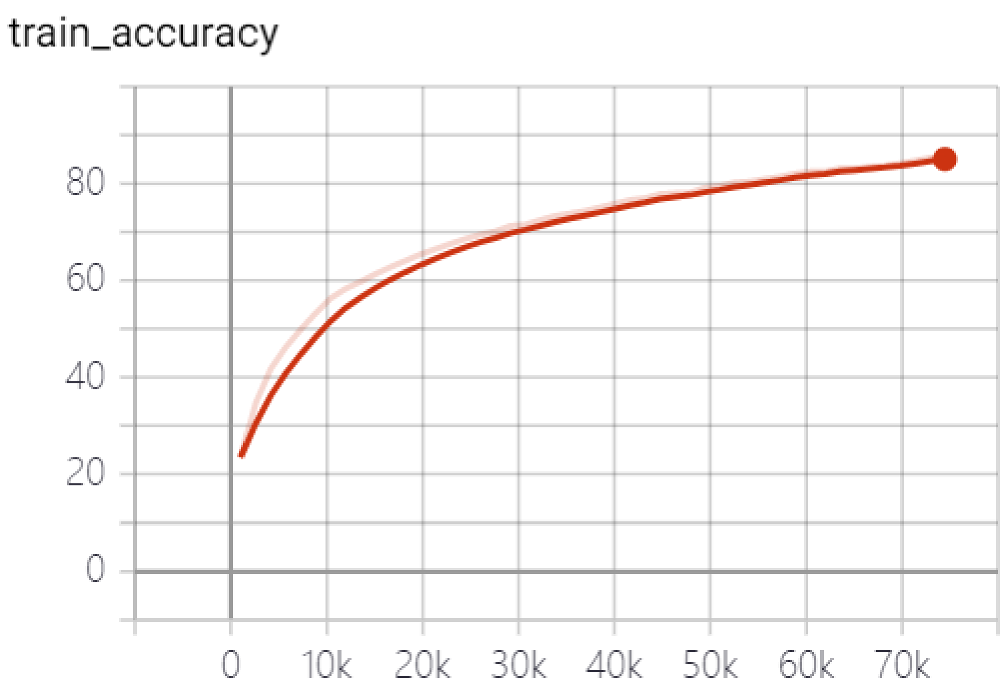


Figure 1.2 Training with Adam

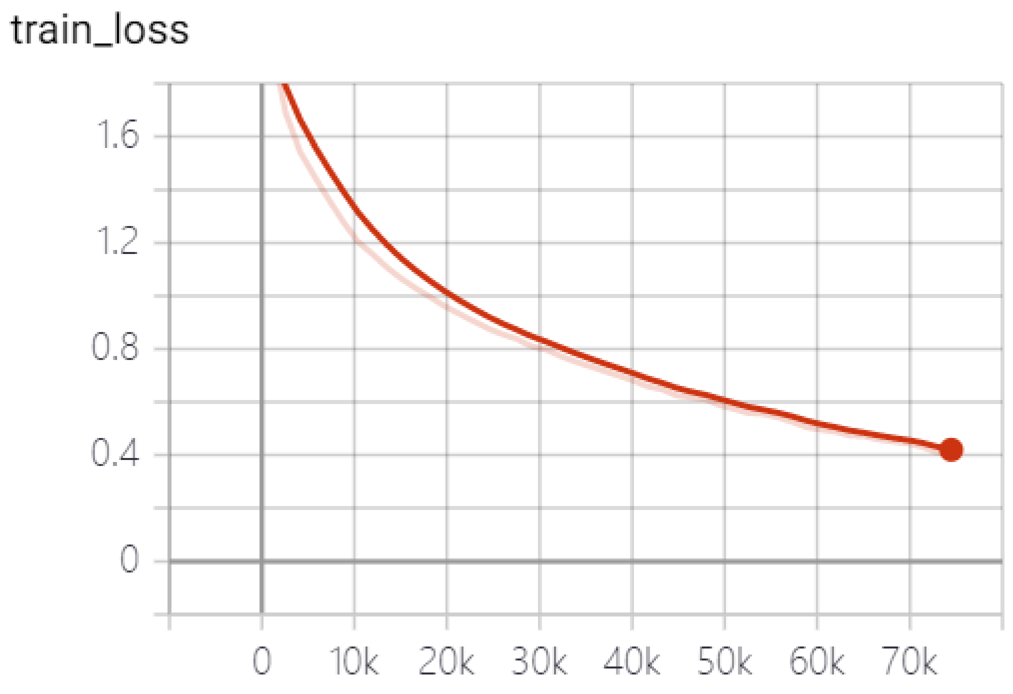
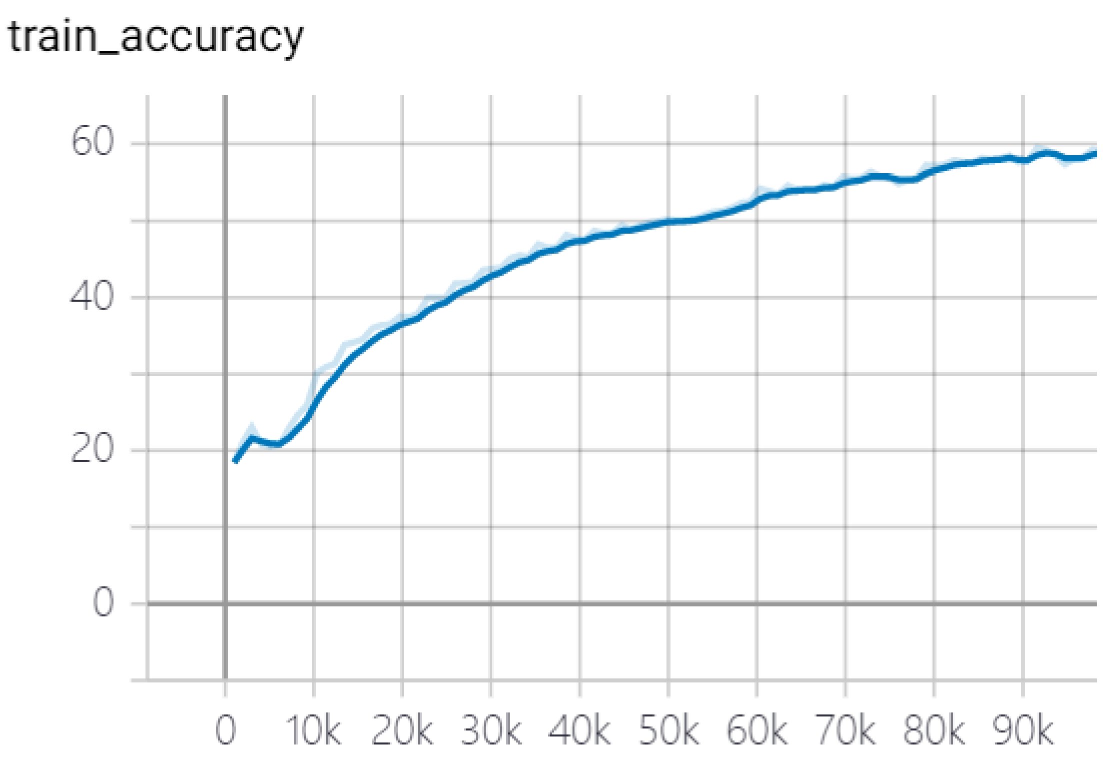
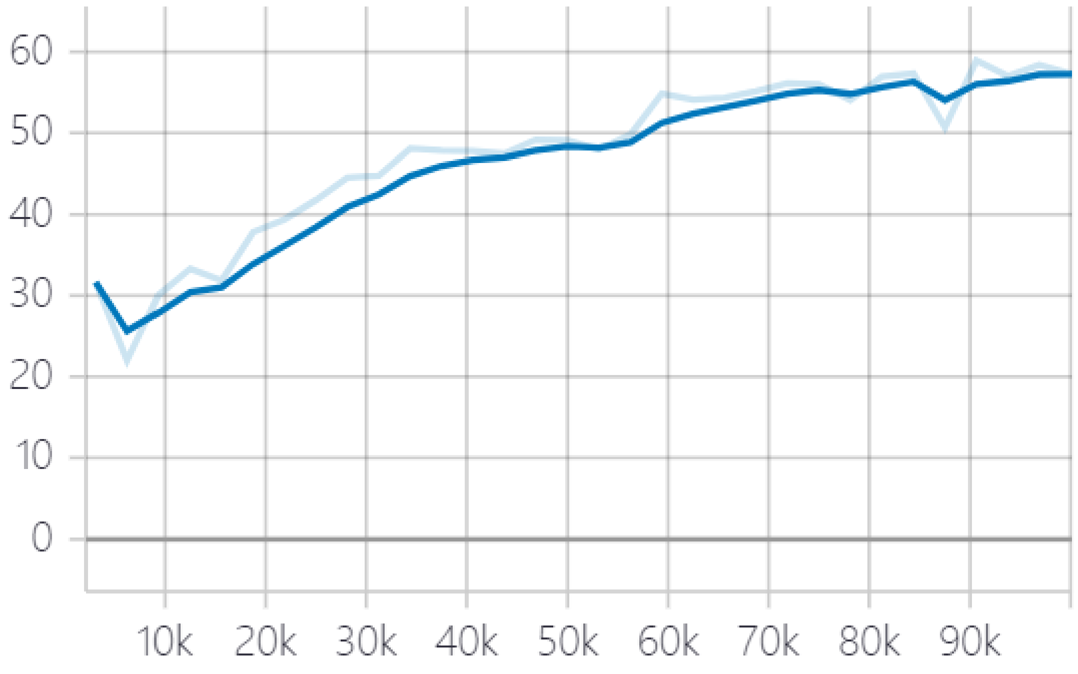


Figure 2.1 training with SDG

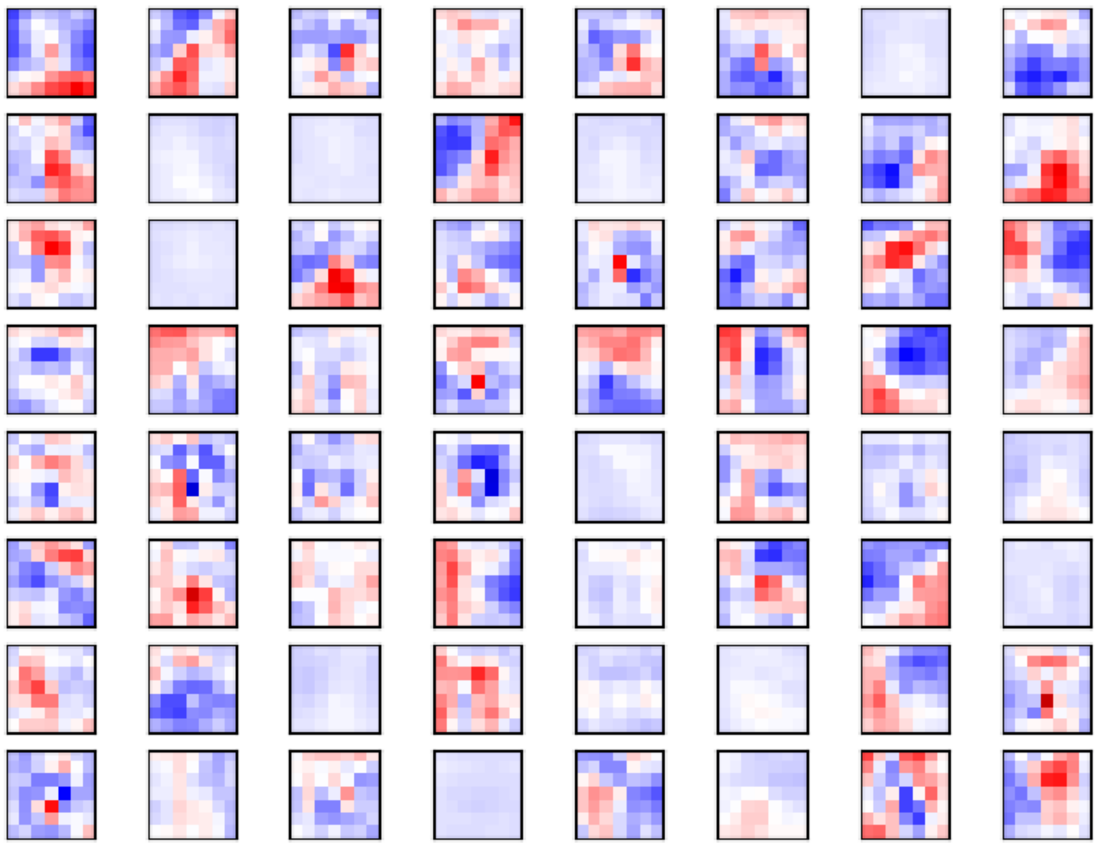


Fifure 2.2 training with SDG



3. visualization

2.1 Filter visualization for conv_1

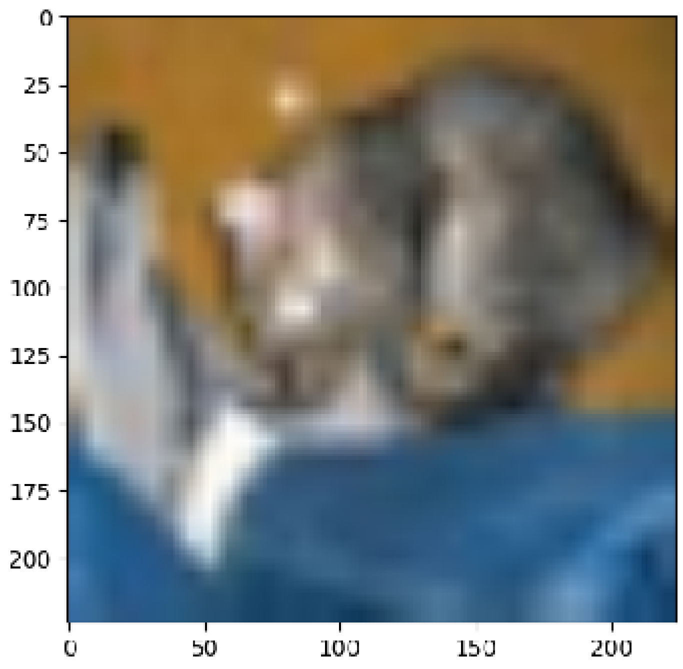


3.2 Feature mapping visualization

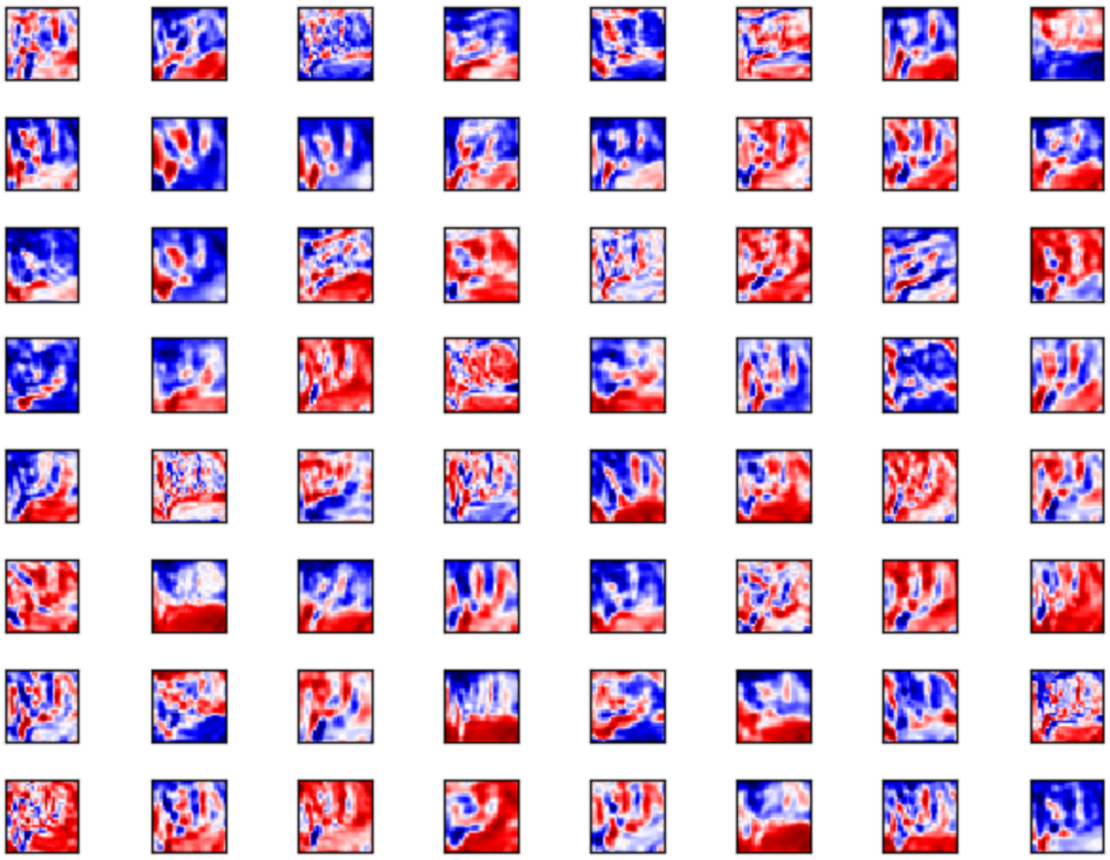
3.2.1 Feature mapping

Given a image as input, features can be extracted from 64 filters in Conv_1 layer. Above pictures shows the output of Conv_1 and Conv_5 with a cat`s image as an input.

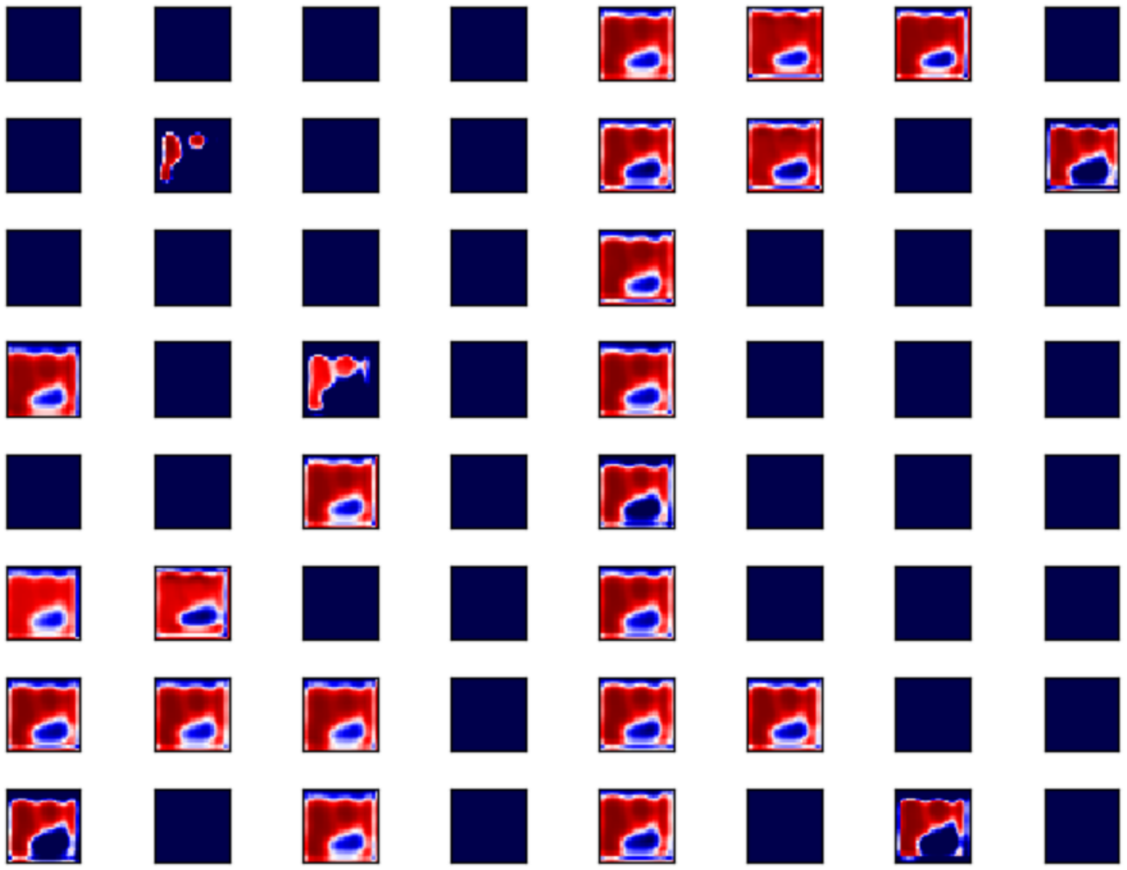
Original Image



feature mapping (Conv_1)



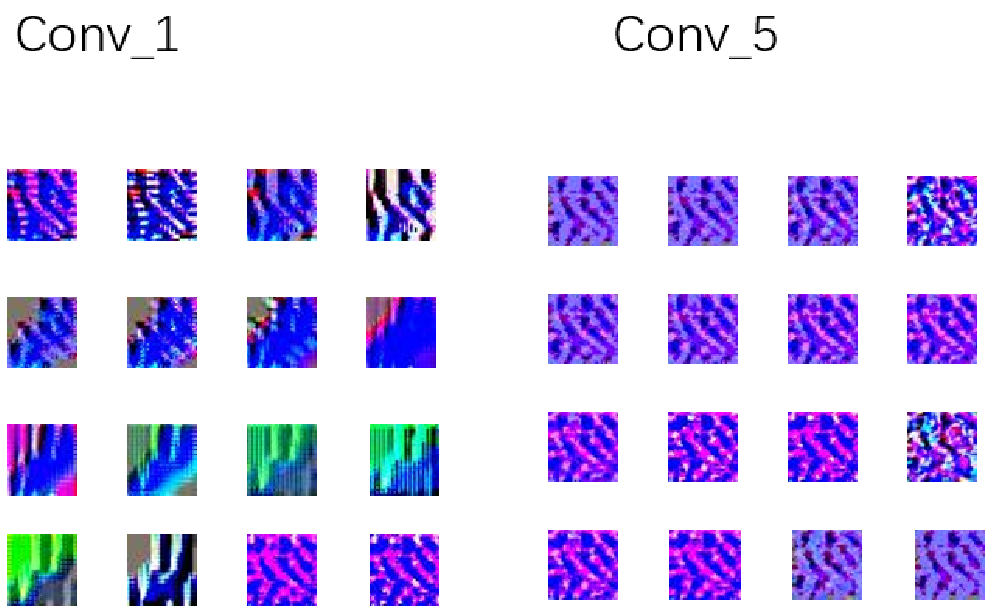
feature mapping(Conv_5)



3.2.2 reconstructed pattern

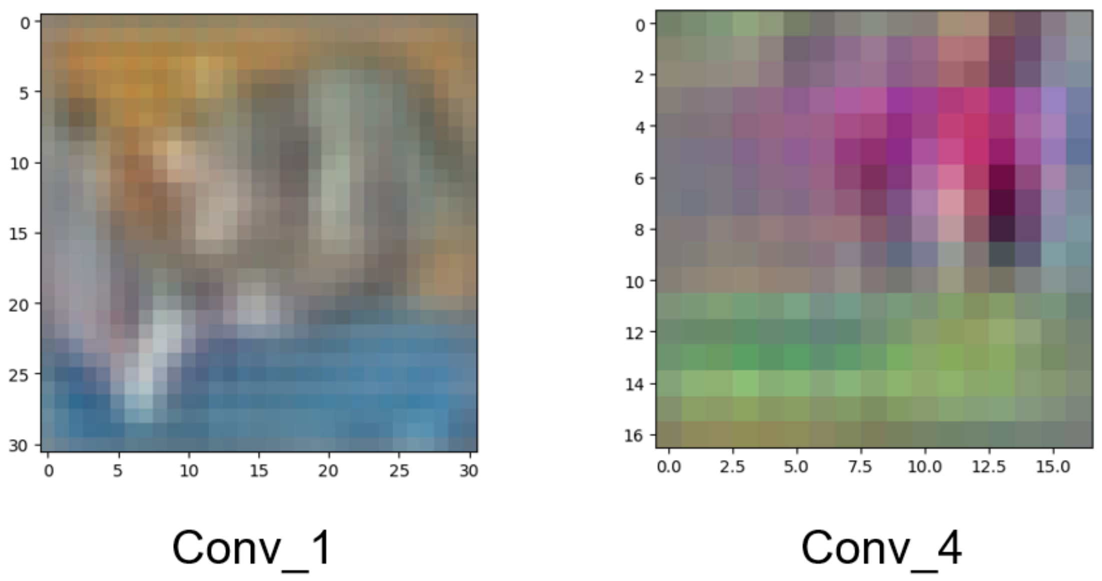
Deconvolution network for reconstructed pattern is annoying and need to copy weight from ResNet to DeResNet one by one.   Hence an other method was used to visualize reconstructed pattern. This method make use of gradient descent to maximum activation of each filter in Conv_1 and Conv_5 respectively. The result only shows 16 filters output for each layer, (show in Figure 3.2.2).

Figure 3.2.2



After have enough time to do this work, I also attempted Deconvolution method to compare the gradient descent method and Deconvolution network. I design a simple deconvolution network to reconstruct pattern. Considering our input image size is 32 X 32 and can't be de-convoluted in layer-5, layer-4 is selected to be visualized in Figure 3.2.3

Figure 3.2.3



4. Feedback

- Spend more than 5 days

- The course is good, but homework is challenging. Although I'm excited about difficulty of this homework, time spending is too much. And I hope that we can have enough time for next Homework.
- We spent too much time on CNN, and I want to learn more about other parts of this course like reinforcement learning.

5. Reference

misc_function,BackProp

```
"Reference: https://github.com/utkuozbulak/pytorch-cnn-visualizations"
```