CNN For Image Recognition

Shiyi Liu, 1901213132@pku.edu.cn

Implement Report for Assignment 1 of AI class, 2019

1 Assignment

Build a CNN model to get familiar with what we learned about neural networks in AI class, and the CNN model built on our own is supposed to process the image classification task with CIFAR-10 dataset [2]. To reach a better performance, we use a ResNet18 for ImageNet dataset as a reference. More details about our assignment can be seen in [1].

2 Implement Environment

The assignment is implemented with *Pytorch1.2.0* of GPU version in *Python 3.7.3*. Moreover, the model is trained in GPU with GTX 1060-6GB and CUDA 10.0. The IDE is *Pycharm Community Edition 2019 2.1* for Windows 10 OS.

3 CNN Model

3.1 CNN Structure

Figure 1 shows the CNN structure references ResNet18[3] rebuilt. Compared to the structure described in Table 2 in [1], 3x3 kernel size instead of 7x7, one stride instead of 3 strides, one padding instead of three paddings are used in conv1 layer, considering the structure of Table 2 in [1] is for ImageNet Dataset with input size of 3x224x224, which is far larger than that of CIFAR-10.

The rest four layers(*conv2_x~conv5_x*) consists of two basic blocks. There is a shortcut in basic block to add the residual between input and output into the network to avoid overfitting in a way. Specially, when the dims of the input and the output is different, another convolutional layer is introduced to down-sample the input feature. In Figure 1, *BasicBlock(s2-s1)* stands for two convolutional layers with one stride forward one strides, while *BasicBlock(s1-s1)* stands for two convolutional layers with

one stride forward one stride. The final feature output by basic blocks is then feed into an average pool layer to down-sample the features captured by convolutional layers. And a fully connected layer is introduced to transform the feature tensors into a 10x1 tenor to recognize the input image and the classify task is completed by the structure. Considering the input image width is only 32, there is no max pool layer in the structure used.

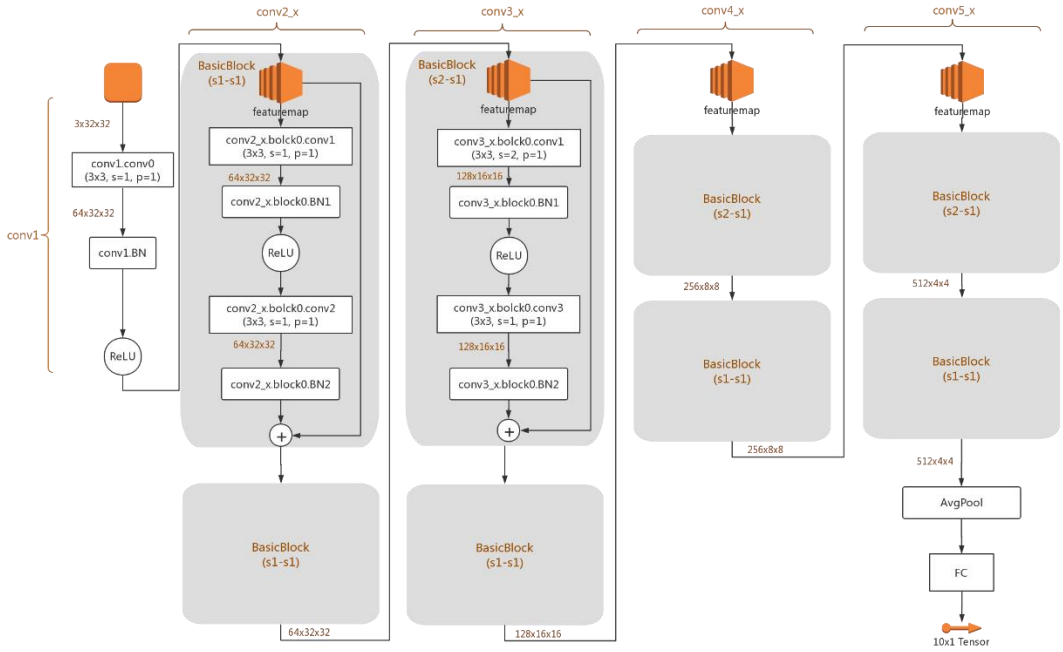


Figure 1 Structure of ResNet18

3.2 CNN Training

3.2.1 Data Preprocessing

Before the model training, the data input to the network needs to be preprocessed first. Then class *Transform* in Pytorch1.2.0 is used to do this. In this part, **random crops, random flip and random rotation** is applied. The input images used for training are then **normalized** to reach a better accuracy, as the analysis in Section 4.2. Moreover, the data read from dataset files are supposed to be transform to *tensors*.

3.2.2 Loss Function and Optimizer

As for the classify task, **cross-entropy loss** is usually used to obtain a better accuracy. And **Adam** optimizer is applied with default parameters.

3.2.3 Scheduler of learning rate

To reach a better training, learning rate plays an import part. The implement applies two tricks about scheduling the learning rate. The first trick is **warmup in the first**

epoch and the second trick is **adjust learning rate with a parameter *gama***, which is used to decay the learning rate, **and a milestone list**, which is used to decide the time when decay the learning rate.

3.2.3 Model Saving Strategy

The model begins to be checked and saved after several of epochs training, and the standard for this is whether the accuracy in test set exceeds that obtained by model training before.

4 Result and Analysis

4.1 Loss and Accuracy

After Series of hyperparameters exploration, the training **batch size is 100**, then the epoch number during training is 20. ***gamma* and *milestone list* described in Section 3.2.3 is set as *0.5* and *[10, 15, 18]***. Then the model trained after 20 epochs obtain **a high accuracy of 92.12%**, and **a low loss of 0.283** in the test set, as Figure 2 and Figure 3 shows.

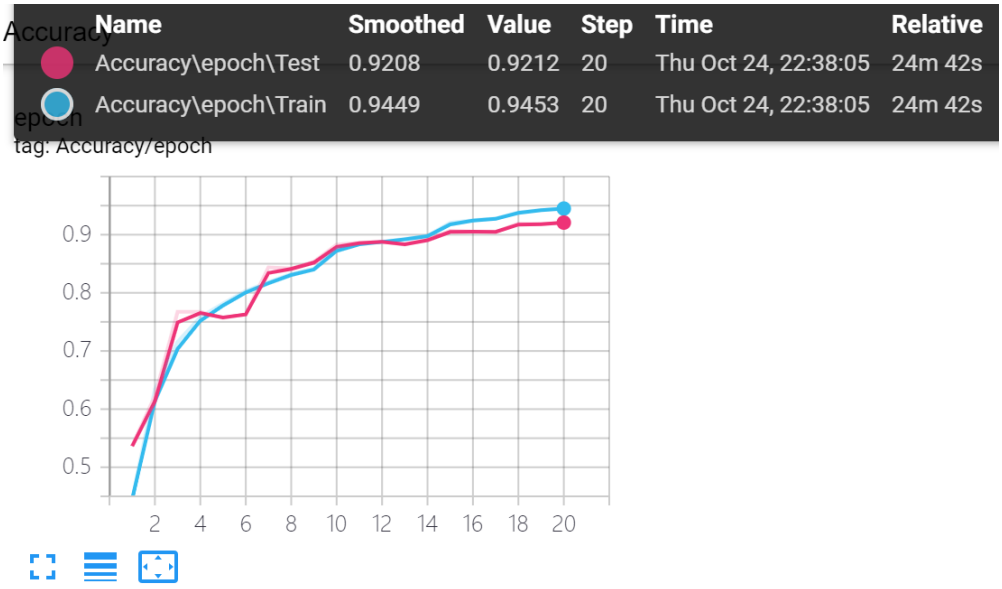


Figure 2 Accuracy of training and test

As the curves in Figure 2 and Figure 3, the model is trained well and does not overfitting at all, and is expected to reach a better performance with more steps.

The reason why test accuracy is sometimes higher than train accuracy may be that the transform operations applied in train dataset but not applied in test dataset.

Figure 4 shows the train loss Curve as a function of number of iterations. The tend

of the curve is stable and shows the training convergence.

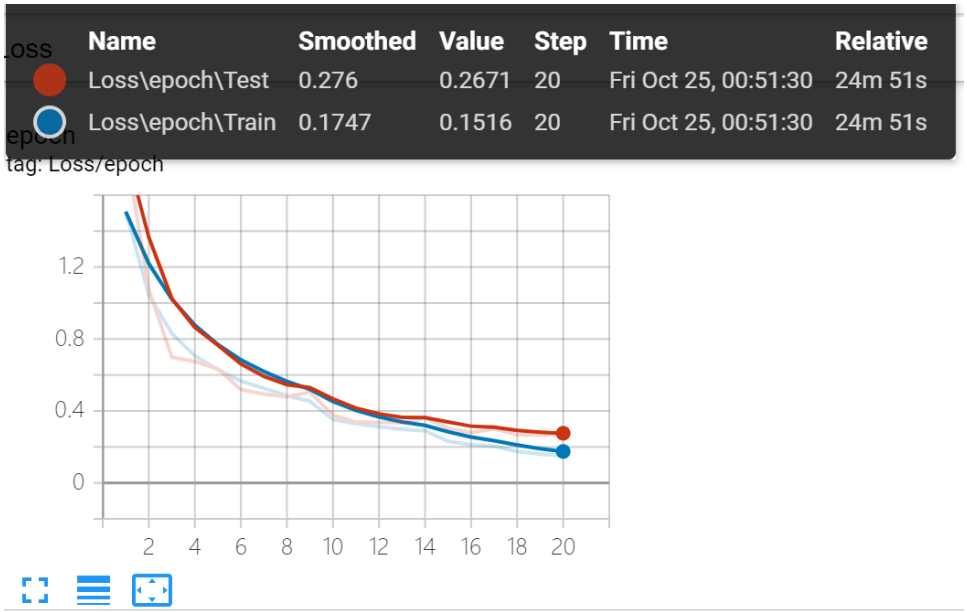


Figure 3 Loss of training and test as the function of epoch

Considering the randomness caused by the shuffle operation of dataset used for training, we trained the architecture with the same hyperparameters many times and got an average train accuracy of 94.02%, average train loss of 0.128, and **average test accuracy of 91.87%, average test loss of 0.213**. Figure 5 shows some other train and test accuracy curves.

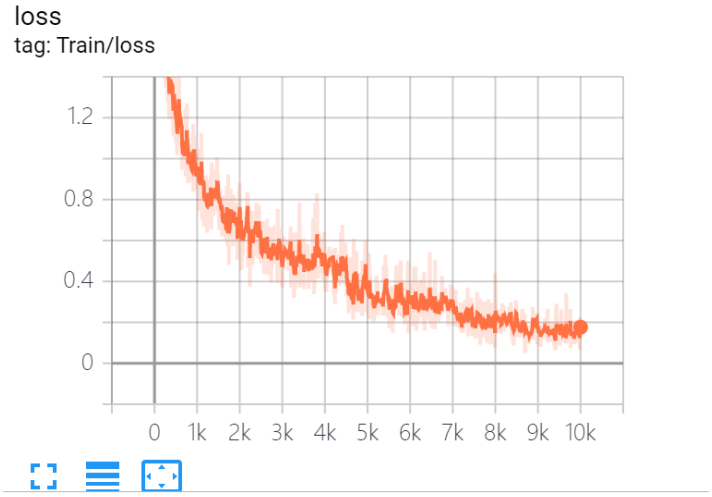


Figure 4 train loss as a function of steps

Specially, among the 7 complete training and testing of the network with the same hyperparameters described above, there are three test accuracy rates above 92%, and

the 7 test accuracy rates all above 91%. The stability of the architecture is proved.

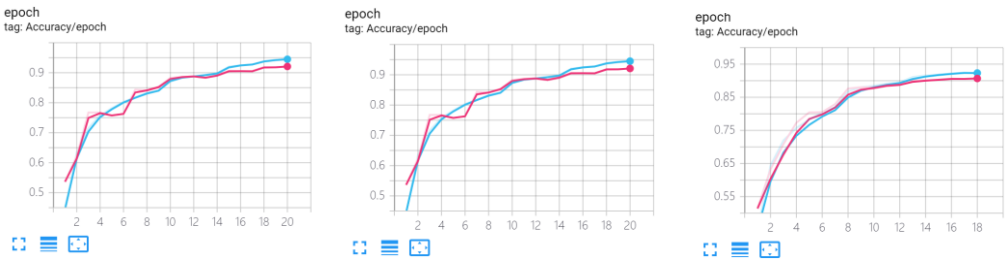


Figure 5 accuracy results with the totally same hyperparameters

4.2 Hyperparameters Exploration Results

The three hyperparameters explored in this section is batch size, milestone list, and gamma. Table 1 shows the results of different combinations. The leftmost column represents the combination of different hyperparameters in the format *(batch size, milestone list, gama)*.

Table 1 results of different hyperparameters combinations

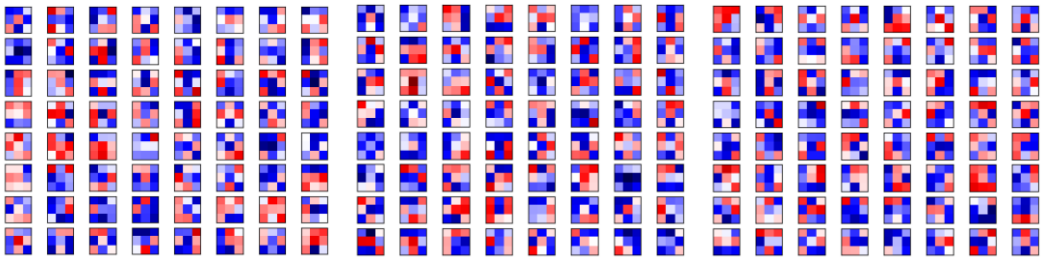
Combinations	Train Loss	Test Loss	Train Acc	Test Acc	Overfitting?
(16, /, /)	0.3524	0.3654	84.23%	85.25%	Underfitting
(64, [10, 15], 0.2)	0.3547	0.3648	89.72%	87.78%	No
(100, [10, 15], 0.2)	0.2897	0.3354	90.69%	89.45%	No
(100, [10], 0.2)	0.2798	0.2574	93.25%	89.21%	No
(100, [10, 15, 18], 0.5)	**0.2127**	**0.2671**	**94.49%**	**92.12%**	**No**
(100, [8,13,16,18,19], 0.5)	0.2578	0.3021	92.21%	88.25%	No

5 Network Visualization

5.1 Filter Visualization

For inter layer analyzing, the filters of the model trained are captured as images, as Figure 6 shows.

The convolutional layer in conv1 in Figure 1 has 63x3x3x3 kernels, the filters in three channels are visualized as Figure 6(a), (b), (c). The color red and blue respectively represent the values of different intervals. From Figure 5, it can be seen that the interval of weight distribution in layer conv1.cov0 is relatively uniform.

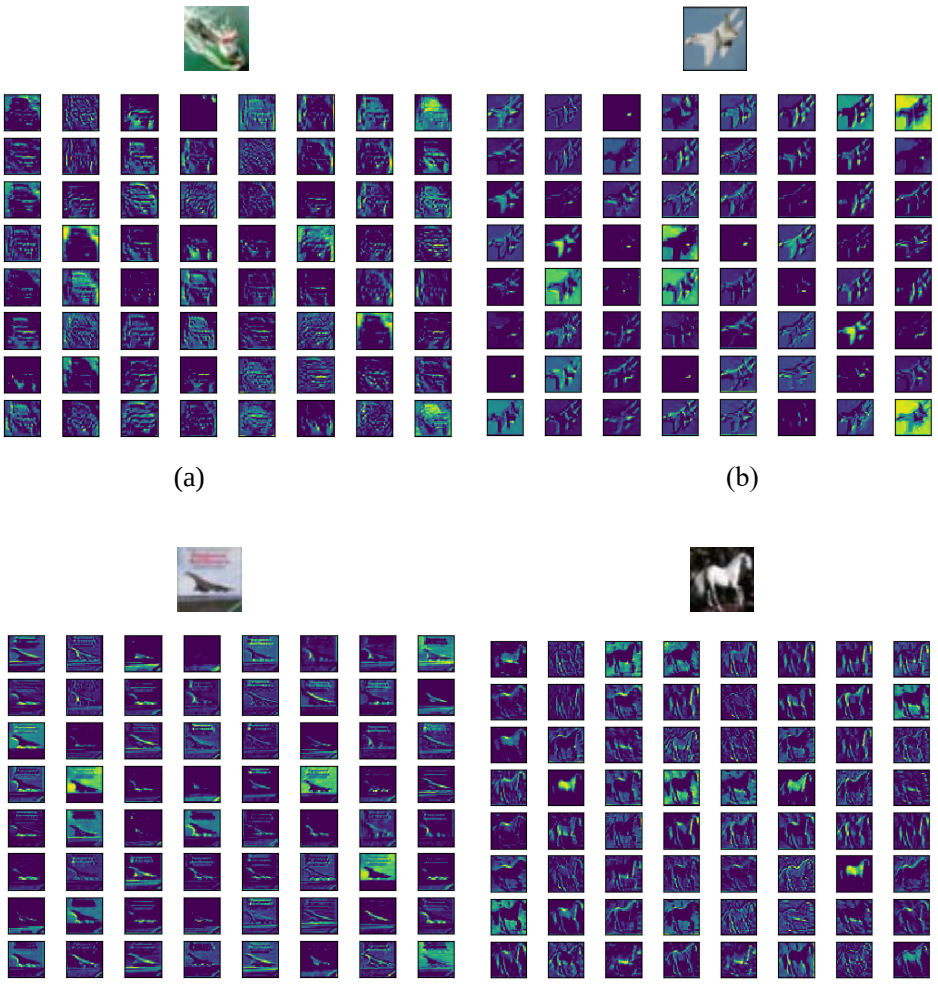


(a) (b) (c)

Figure 6 kernels in conv1.conv0

5.2 Feature Map Visualization

5.2.1 Feature Map analysis and Image Reconstructed after conv1



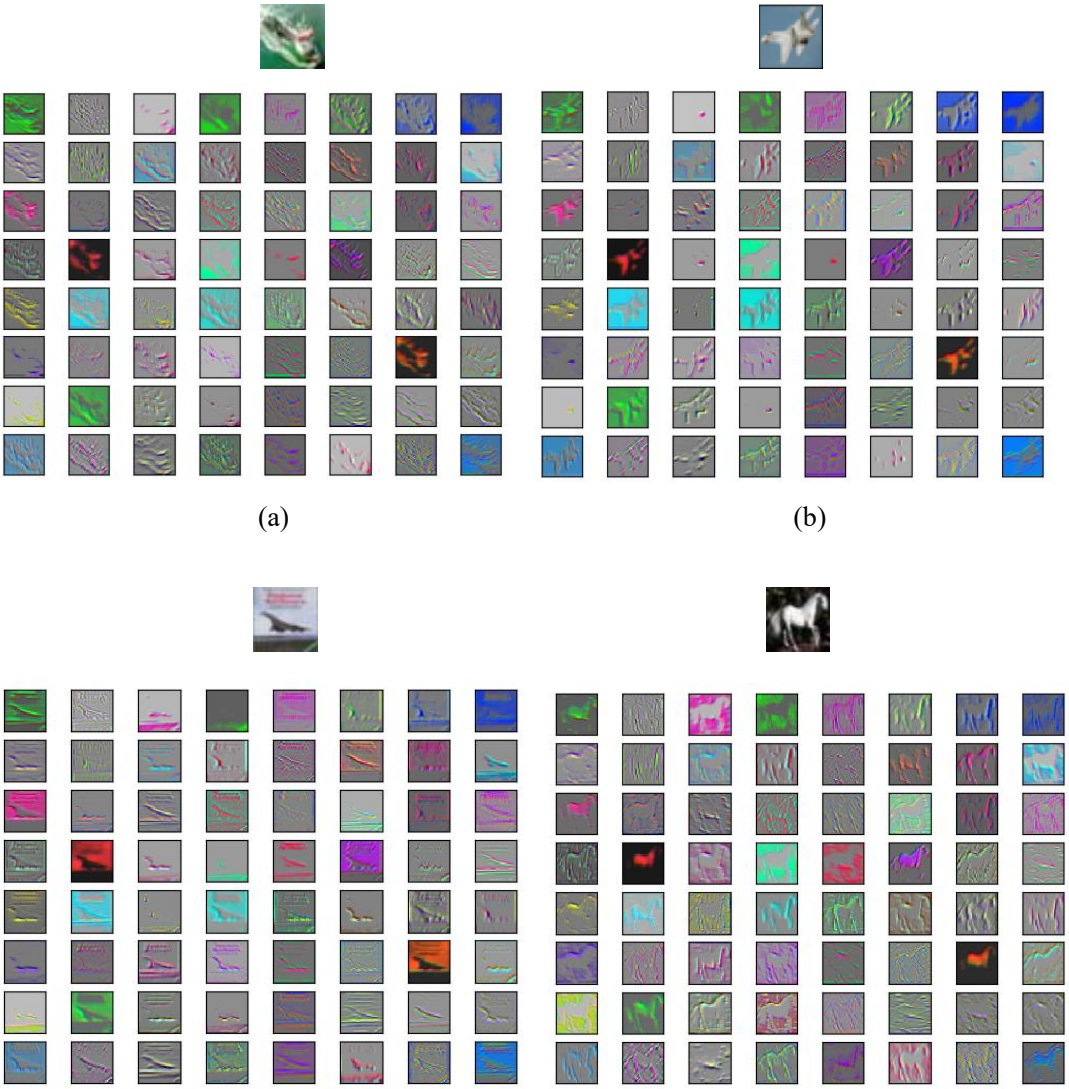
(a) (b)

(c) (d)

Figure 7 Feature maps of conv1

The 64 feature maps output by conv1 layer of four images can be shown as Figure 7(a), (b), (c), (d). Then the 64 reconstructed images from each feature map as can be seen in Figure8(a), (b), (c), (d). From the analysis of the 64 reconstructed images of the four images, it can be seen that the feature maps with indices 0, 16, 46, 49 have a good effect as an activation to feed a deconvolutional network [4] of the structure in Figure 1.

The selected activations are then feed to the deconvolutional network mentioned above and obtained the reconstructed images. Figure 9(a), (b), (c), (d) shows the 16 images and the corresponding reconstructed images activated by the four feature maps.



(a) (b)

(c) (d)

Figure 8 Reconstructed Images from Feature map of conv1

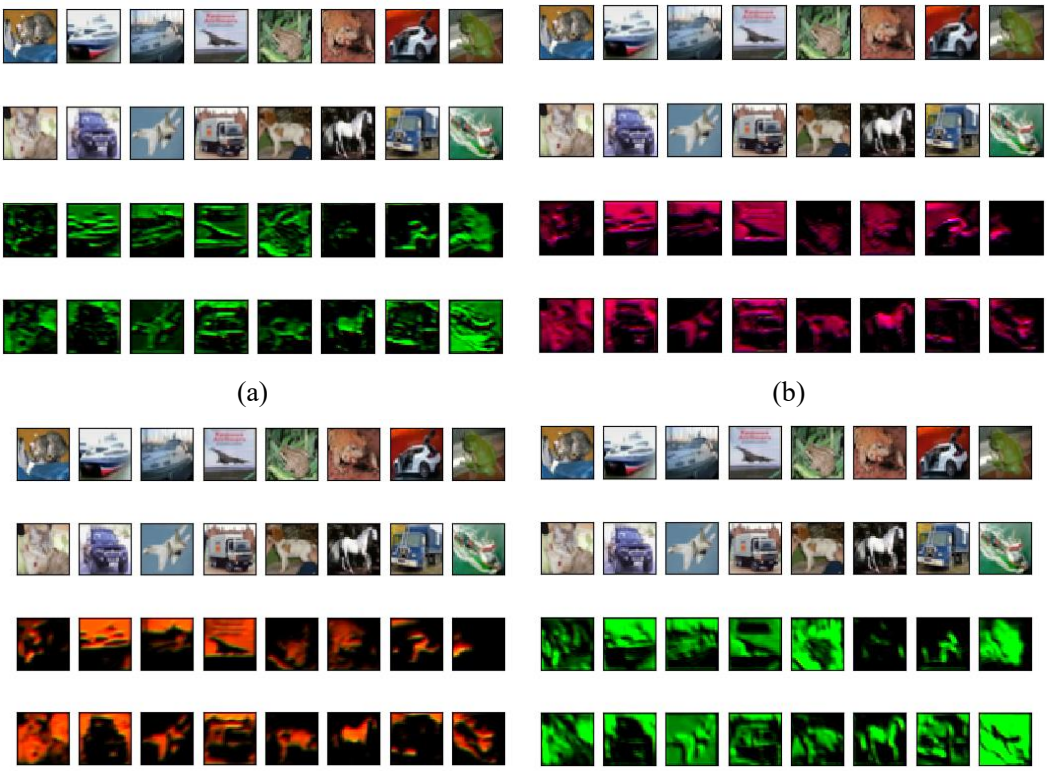


Figure 9 Reconstructed Images from 4 activations

5.2.2 Feature Map analysis and Image Reconstructed after conv5

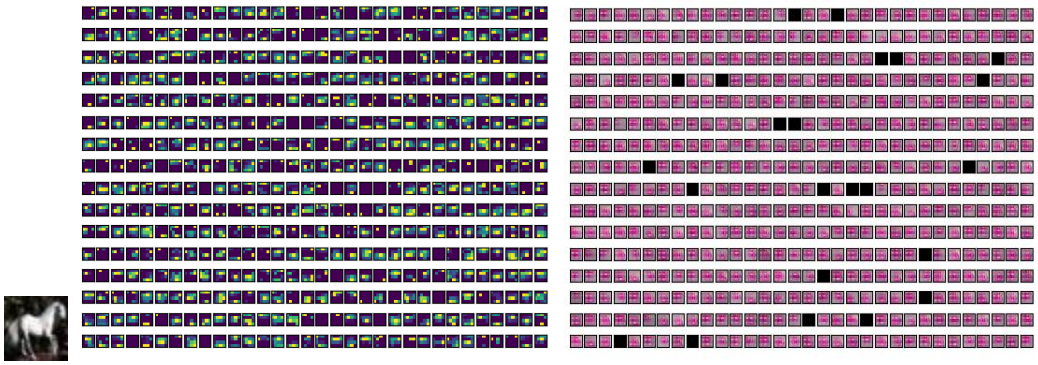


Figure 10 raw image, feature maps and reconstructed images of conv5

Figure 10 shows the raw image, feature maps and reconstructed images of conv5. Compared to Figure7~Figure 9, either the feature map or the reconstructed image is hard to understand and it is difficult to have similarity compared with the texture and content of the original image.

The reason for this may be that the output size of the conv5 layer is too small to

express anything. Furthermore, since continuous convolution is easy to produce a checkerboard effect, images reconstructed from the feature map may have a large rasterization phenomenon.

6 Feedback

- **Time you spend for this assignment, i.e., how many hours?**
  Almost the whole week, 45+ hours spent coding and bugging.

- **Comments for this course?**
  In my opinion, The AI class is set to summarize the AI as a whole, enhance our understanding, and expand our thinking, instead of cultivating our deep learning techniques.

- **Comments for this assignment?**
  As a small assignment, it may be too cost-of-time, some students already have heavy research task to complete on their back and this task who focus on coding and implementation seems to bring too much burden! I spent a whole week, including three nights to 3 o' clock working on it. Time limit needs to be increased!

- **Suggestion for the following lectures?**
  Considering that the level of students is mixed, the foundation and the promotion should reach a better trade-off. The focus should not be coding and implementation in my opinion.

Reference

[1] Assignment1 CNN for image recognition, https://github.com/Ryanrenqian/AI_HW.

[2] CIFAR-10 Dataset https://www.cs.toronto.edu/~kriz/cifar.html.

[3] ResNet18 https://arxiv.org/pdf/1512.03385.pdf.

[4] https://arxiv.org/pdf/1311.2901.pdf

[5] Code Reference:
train_classify.py reference to: https://github.com/weiaicunzai/pytorch-cifar100
conv_ cvisual.py reference to: https://github.com/grishasergei/conviz/blob/master/conviz.py