IFT6135 Assignment2, Programming part

Xiao Fan (20086722)
Zhibin Lu (20091078)

March 14, 2018

1. Regularization : weight decay, early stopping, dropout, domain prior knowledge

(a) Early stopping and weight decay

In the figure 1 and 2, we show respectively the training and test error and the L2 norm of all parameters for model without regularization and model with L2 regularization.

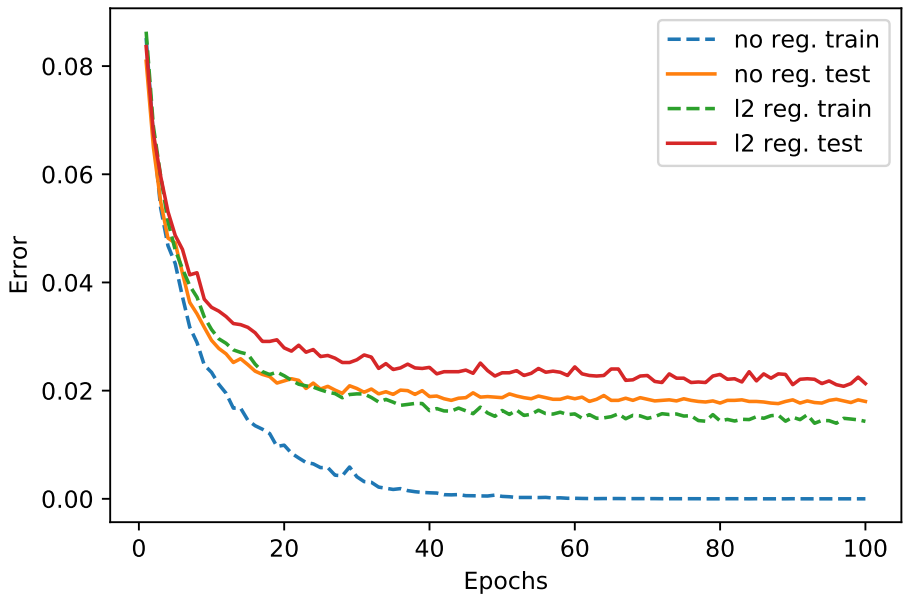


Figure 1: Training and test error of model with no regularization and model with L2 regularization.

As in figure 1, for model without regularization, we observed overfitting. The training error (nearly 0 after 100 epochs) is much smaller than the test error (around

0.02). When applied L2 regularization, overfitting problem is avoided. Both the training and test error is larger but the gap between them is much smaller.

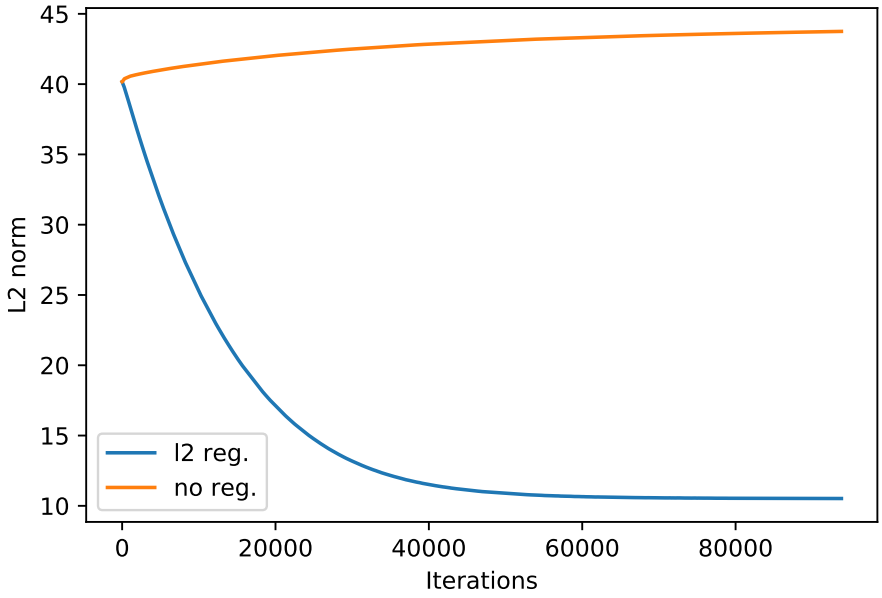


Figure 2: L2 norm of all the parameters as a function of iterations.

In figure 2, we can see how the penalty term affect the parameters of model. With L2 regularization, the norm of parameters is decreasing during the training which is known as weight decay.

(b) Dropout

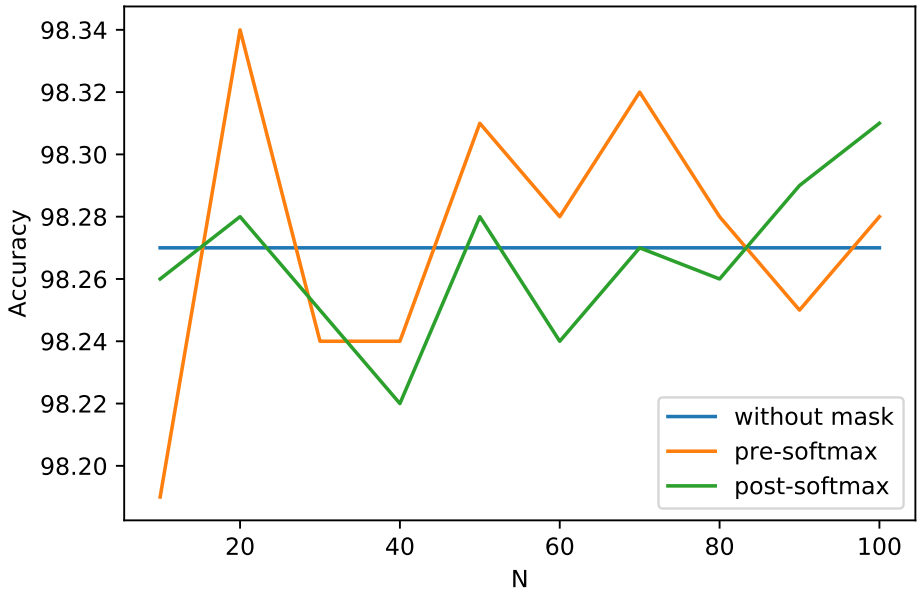


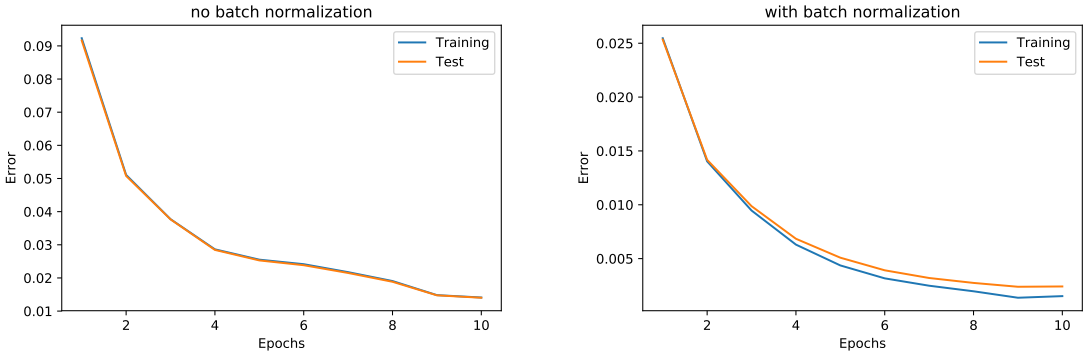
Figure 3: Prediction accuracy of test set as a function of N.

Figure 3 shows the accuracy on test set as a function of N for the three different prediction schemes.

The first schema uses the scaled weight to approximate the geometric mean. The second uses the bagging with arithmetic mean averaging and the third uses the geometric mean averaging. We observed in the figure that there is no advantage to using the arithmetic mean over possible subnetworks rather than the geometric mean and the scaled wight approximation performs as well as geometric mean.

(c) Convolutional Networks

Figure 4 shows the training and test error with or without batch-normalization.



(a) without batch-normalization (b) with batch-normalization

Figure 4: Training and test error against training time.

With batch-normalization, both the training and test error is greatly decreased compared to the model without batch-normalization. The model with batch-normalization begins with a smaller error and converges more rapidly. This technique has greatly accelerated the training process.

2. Dogs vs. Cats Classification

We firstly used network VGG16 and trained the model from scratch. Image size is 64. The architecture of VGG16 with input image size of 64 is in figure 5. Total number of parameters is 14,727,234.

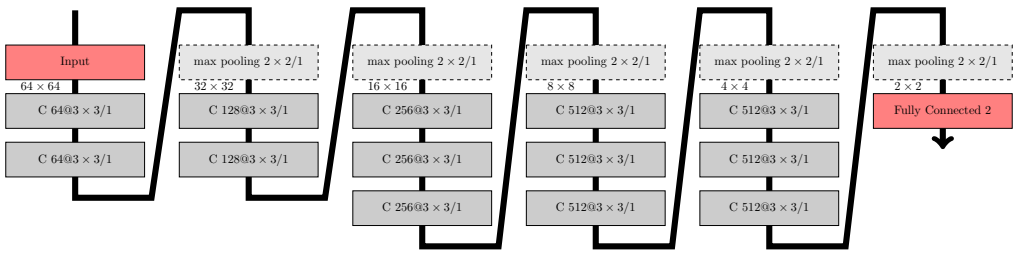


Figure 5: Architecture of VGG16 with input image size of 64.

20000 examples are used as training and validation set and the split is 80/20. The remaining 4990 examples are used as test set. Batch-normalization layer is added after each convolution layer to accelerate the training. The best validation accuracy after 100 epochs is 93.95%. We used SGD with learning rate of 0.1, momentum of 0.9, weight decay of 5×10^{-4} and minibatch size of 25. Accuracy on test set is 93.808%. Figure 6 shows the training and validation error. Experiment results are in table 1.

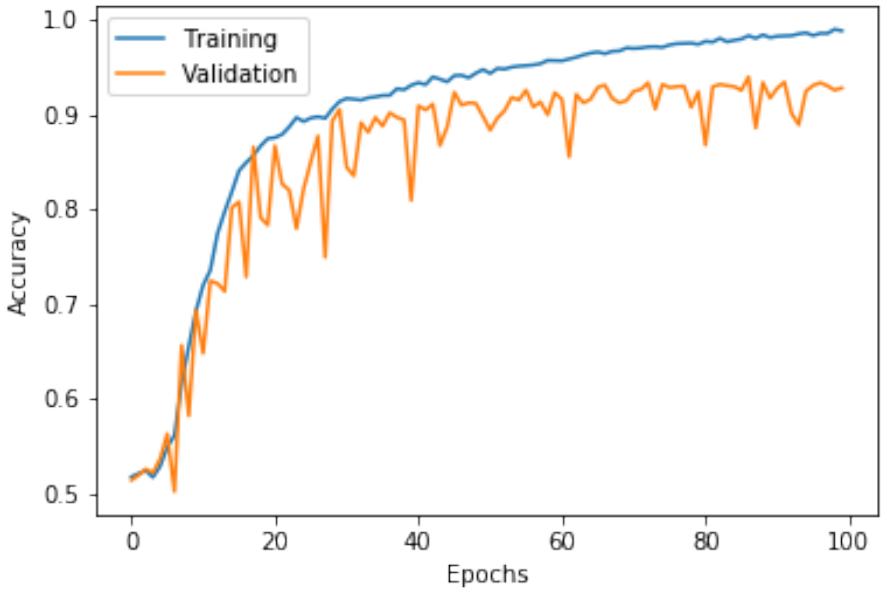
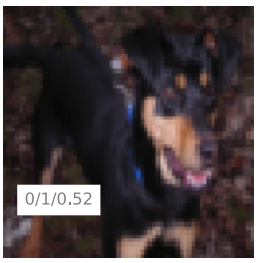
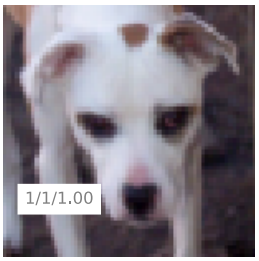
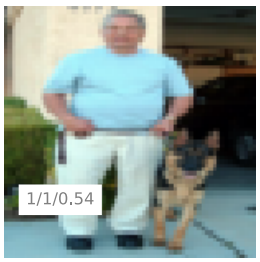


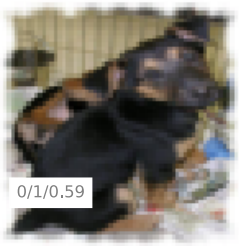
Figure 6: Accuracy on training and validation set with VGG16 network.

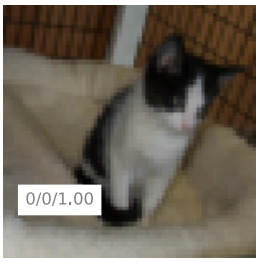
Figure 7 shows some examples of images where (a) the classifier predicts around 50% on both classes (b) the prediction is correct with high probability (c) the prediction is wrong with high probability.

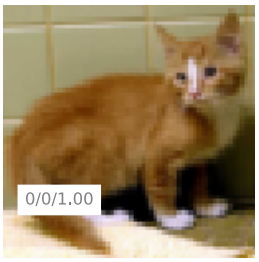




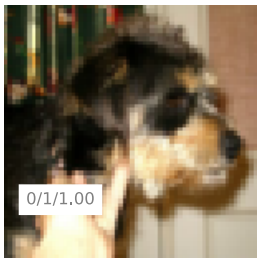


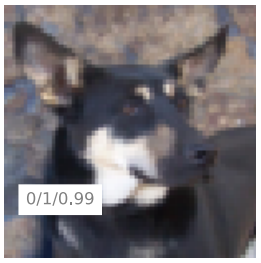


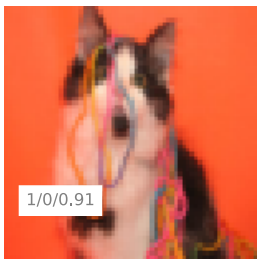


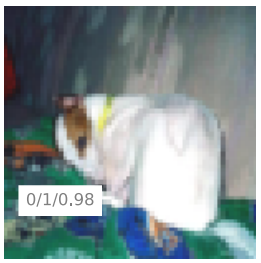


(a) Probability around 50% on both classes (b) Correctly predicted with high probability









(c) Wrongly predicted with high probability

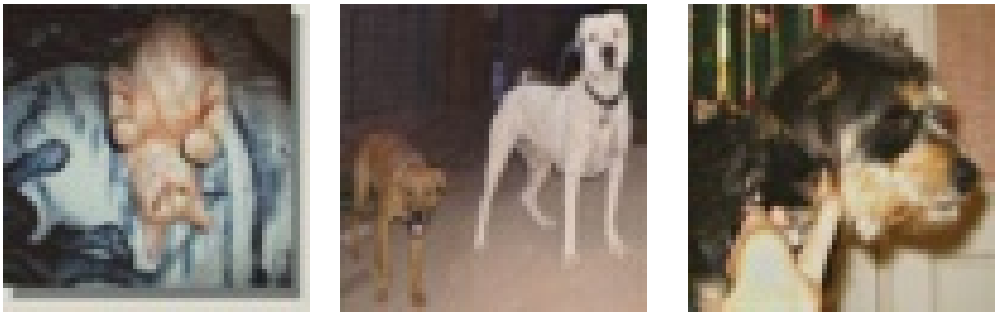
Figure 7: Examples of image (text in the image means {predicted class}{label class}{prediction probability}, 0 is cat, 1 is dog)

Compared (a) and (b), we observed that the background of the object has a great effect on the prediction. When background of image is more complex, for example, black dog in the darkness, a person beside a dog, the classifier tends to give a lower probability on the prediction. In the contrast, when there is no distraction in the background, the model outputs a high probability as in (b). For images in (c) which output high probability but with wrong prediction, we found that they are indeed deceptive. For example, the cat in orange background in the lower left corner really looks like a dog. It is difficult to distinguish even for human beings.

Figure 8 is the inverted image representations [1]. Idea here is to reconstruct the original images after nth layer using the features extracted. We used some existing code and adapted it to our project [2]. To restore the image, it tries to minimize the loss between the nth layer output of the original image and that of the restored im-

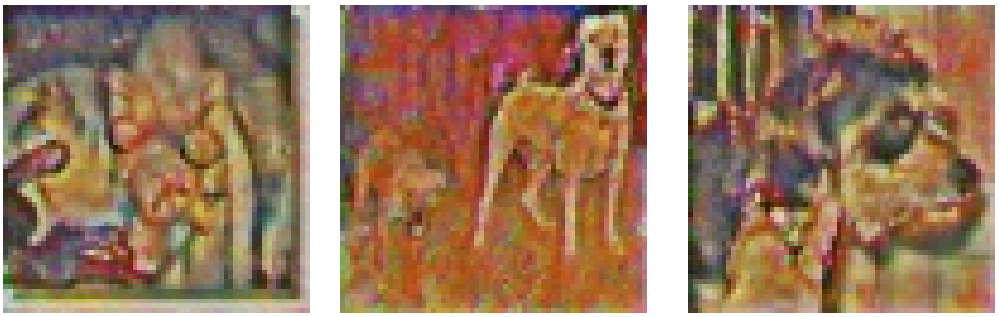
age and uses gradient descent for optimization. As it goes deeper in the network, the restored image has more distortion and is more abstract because the mapping between input and output is more complex and more information is lost when iterating through the network.

In addition, we observed something interesting. The image in the middle containing a white dog is correctly predicted and the output probability is 1. The tail of the original image is not prominent in the darkness, but in the deeper layer, the tail is highlighted with clear contour (layer 7 is the most obvious). This shows that the network is able to extract features that are meaningful for classification but may not be important in the original image. What's more, compared the three images where the first has a prediction around 0.5 and the second has a prediction about 1, we found that in the restored image at 7th layer, the second image is still recognizable but the first is not. We have an intuition why the second image has a higher probability.



(a) Layer 0 : **Conv2d**

(b) Layer 3; **Conv2d**



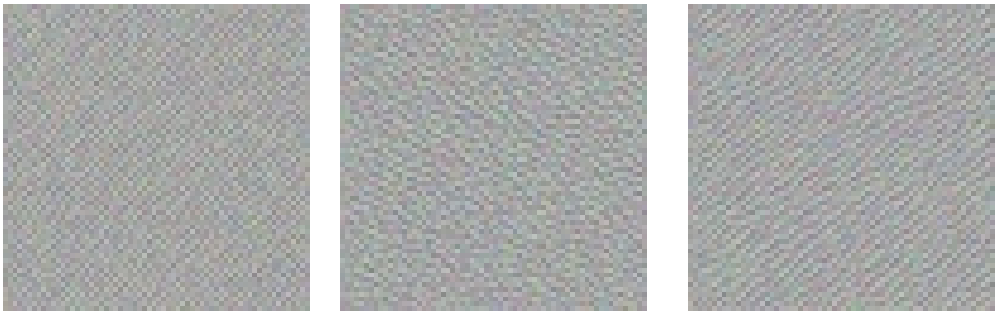
(c) Layer 6 : **MaxPool2d**

(d) Layer 7 : **Conv2d**

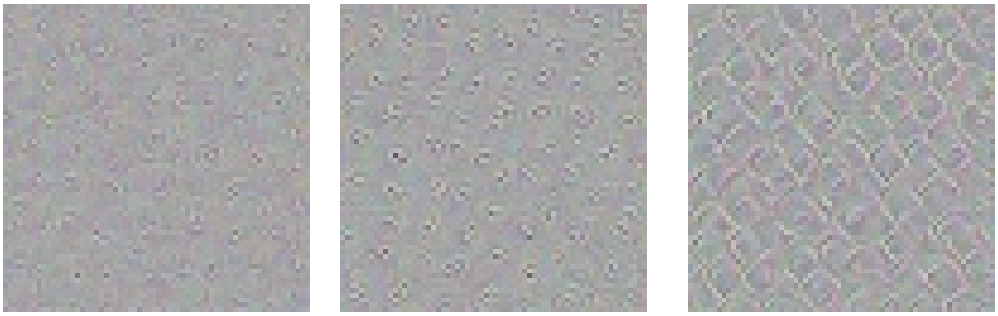
Figure 8: Inverted Image Representations

We also visualized the kernels as in figure 9 . The principle is to randomly generate a picture of the same size, do forward propagation through the network, and obtain the corresponding output feature in the nth layer. It then use the mean of this feature as loss and minimize it using gradient descent to update the input image. We refer to the code [2]. We observed that low-level CNN layer learn some simple geometric shapes such as lines, edges, and corners. In deeper layer, it learns features such as eyes or noses. As the network goes deeper, it learns the body parts, such

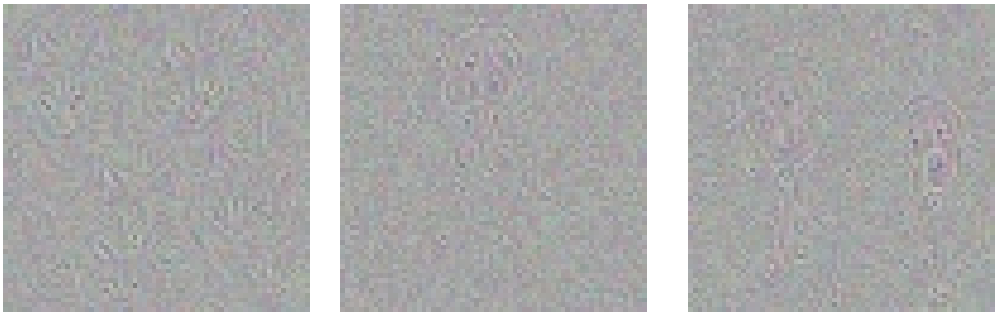
as the head of a dog and the head of a cat. It confirms the fact that deeper layers are able to extract more complex features composed of basic shapes from shallow layers. Sometimes, the features can be too abstract to be explained such as in (d).



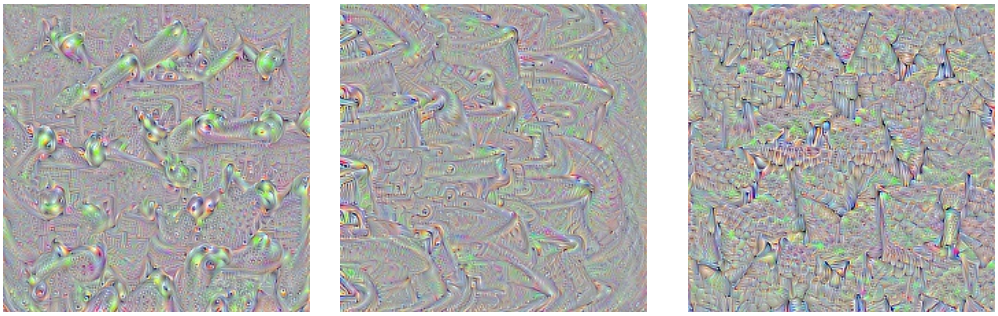
(a) Layer 7 : **Conv2d**



(b) Layer 14; **Conv2d**



(c) Layer 34 : **Conv2d**



(d) Layer 20 : **Conv2d with 224 size and pretrained model**

Figure 9: visualize the kernel

We also tried with pretrained models provided by pytorch. Image size is 224. We fixed parameters in convolutional layers and trained only the classifier layer. Just after a few epochs, we obtained a high accuracy on validation set. Among several

networks, pretrained resnet152 achieves the highest accuracy of 99.025% on validation set and 98.778% on test set. We used SGD with learning rate of 0.001.

Network (Image size)	Epochs	Training	Validation	Test
VGG16 (64)	100	98.294	93.95	93.808
Pretrained Resnet152 (224)	20	98.381	99.025	98.778
Pretrained VGG16 (224)	20	92.381	95.150	95.170

Table 1: Best accuracy obtained with different network architecture.

From the table and the visualization above, we consider that increasing the size of the original input image and increasing the depth of the network helps to improve the accuracy. In addition, using the Resnet can indeed greatly improve the accuracy.

References

[1] A. Mahendran and A. Vedaldi, "Understanding deep image representations by inverting them," *CoRR*, vol. abs/1412.0035, 2014. [Online]. Available: http://arxiv.org/abs/1412.0035

[2] "Convolutional neural network visualizations." [Online]. Available: https://github.com/utkuozbulak/pytorch-cnn-visualizations