Multi-path Convolutional Neural Networks for Image Classification

by

Mingming Wang

Submitted in partial fulfilment of the requirements
for the degree of Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
August 2015

Table of Contents

List of Tables

List of Figures

List of Abbreviations and Symbols Used

CMYK	Cyan Magenta Yellow Black
CNN	Convolutional Neural Networks
GPU	Graphic Processing Unit
ReLU	Rectified Linear Unit
RGB	Red Green Blue
ILSVRC	ImageNet Large-Scale Visual Recognition Challenges
SGD	Stochastic Gradient Descent

Abstract

Convolutional Neural Networks have demonstrated high performance in the ImageNet Large-Scale Visual Recognition Challenges contest. Nevertheless, the published results only show the overall performance for all image classes. These models do not include further analysis why certain image are misclassified and how they could be improved. In this thesis, we provide deep performance analysis based on different types of images and point out weaknesses of CNN (Convolutional Neural Networks) through experiment. We designed a novel multiple paths convolutional neural network, which feeds different versions of images into separated paths to learn more comprehensive features. This model has better presentation for images than the traditional single path model. We design different experiments to show that our model gets better classification results in top 1 and top 5 scores on the 100 categories dataset by comparing with the model, which was the best one for the image classification task in ILSVRC 2013.

Chapter 1 Introduction

1.1 Research objective

Image classification is a hot research topic for computer vision and machine learning. Among all popular models, Convolutional Neural Networks [9,12,13,14,28] show significant advantages over other ones. Especially, in recent ImageNet Large-Scale Visual Recognition Challenges [20], series models have made great performance improvement, such as [11] achieved a top 5 error rate at 16.4% in 2012 and [29] reached a top 5 error rate at 14.7% in 2013 for classification task.

Albeit great progress has been made, there is still very limited performance analysis with internal operations and how image data impact performance. More specifically, what images get good results and what images are misclassified by on CNN? Why do some images get worse results and how can we improve it? In this thesis, our research answers the above questions through the following steps.

1. We firstly design an experiment to compare the performance of a single path CNN model on 2 groups of datasets that have different image complexities. Both groups contain training and validation images with equal quantity and class labels. The first group contains the simplest images and the second group contains the most complex images. We run the model on 2 groups of datasets independently and get 2 set of learning curves.

2. Secondly, we use the above learning curves and the internal operations of the CNN to do further analysis. We clearly point out the weaknesses of convolution and pooling operations, which are sensitive to high frequency components of images. This mechanism causes the objects with a simple shape and less texture to gradually disappear

in the clustered background after several iterations of convolution and pooling. Therefore, the final feature vector cannot present foreground objects well and then gets a bad result for prediction.

3. Finally, We define a novel multiple paths networks to overcome the weakness of the single path model. These paths learn different aspects of images and retain more valid features than the single path model. We design 5 different experiments to prove that:

1) Our method get more accurate result on the complex image datasets than the model [29], which was the best one for the image classification task in ILSVRC 2013.

2) Our model shows more generalization ability on simple validation images as well.

3) Our model is more accurate on a 100 class dataset than the model [29], which is used in experiment 1.

4) The counterexample of a single path model that trains and tests bilateral filtered images. It shows worse performance than our multi-path model on a 100 class dataset.

5) The counterexample of training our model with source images and Gaussian filtered images in different paths. It shows worse performance than training source images and bilateral filtered images.

1.2 Outline

In chapter 2, we first introduce the ImageNet dataset. This is the most challenging dataset for computer vision and image classification contests in the world. We then talk about the image processing tools, including 2D wavelet transform and bilateral filter. We use 2D wavelet transform to get wavelet coefficients to measure the complexities of the image. We use a bilateral filter to process images. It is a powerful tool that keeps edges but reduces the noise or high frequency components.

In Chapter 3, we discuss the CNN model and the internal steps of each layer, such as convolution, pooling and forward and backward propagations. For the regression layer, we use softmax regression. Regularization is commonly used in CNN to prevent overfitting. . Dropout is another method to reduce overfitting. It is applied in our model and all experiments in chapter 5. Removing dropout make learning curves show significant overfitting. We review some popular models, which use parallel ways to train the networks. These models have different designs from ours.

Chapters 4 and 5 discuss the experiments. In chapter 4, we present an experiment to compare the performance of a model on two datasets with different complexities. Overall, we have better result on the simple dataset than the complex dataset. We analyze the reason for performance drop on the complex dataset. It was caused by high frequency components dominating the convolution and the pooling. In chapter 5, we design and implement multi-path CNN. It overcomes the weakness of the traditional one path model and learns a broader range of features from the two paths that feed source and bilateral filtered images. We designed 5 different experiments: 1) Training and testing our model and model [29] on complex image dataset. 2) Using above trained models to run model generalization test on simple validation image dataset. 3) Training and testing a 100 class full dataset with source and bilateral filtered images in different paths. 4) Running a 100 class full dataset with bilateral filtered images on a single path. 5) Training and testing a 100 class full dataset with source and Gaussian filtered images in different paths. The results of 5 experiments show that multi-path CNN using source and bilateral filtered images have more advantages than all the others.

In chapter 6, we talk about conclusions and future work. Overall, our model is the first one to analyze the performance of CNN based on the frequency domain of images by using wavelets coefficients. This approach has not been adopted by other models. As we get good results on the 2 paths model, we plan to extend it to more paths and on each path to use different settings and depths. The simple objects need fewer depths and complex objects need more depths, since the objects with different complexities should have different levels of abstraction. To speed up training, we could leverage multiple GPUs to let different path run on different GPU.

Chapter 2 Dataset and Image Analysis

2.1 ImageNet dataset

The ImageNet Large Scale Visual Recognition Challenge [20] is a benchmark for image classification and object detection with millions of images. The challenge has been run annually since 2010 and has become the standard benchmark for large-scale object recognition. More than fifty research groups have taken part in this contest.

WordNetID	Synsets	Description
‘n02119789’	‘kit fox, Vulpes macrotis’	‘small grey fox of southwestern United States; may be a subspecies of Vulpes velox’
‘n02100735’	‘English setter’	‘an English breed having a plumed tail and a soft silky coat that is chiefly white’
‘n02110185’	‘Siberian husky’	‘breed of sled dog developed in northeastern Siberia; they resemble the larger Alaskan malamutes’
‘n02096294’	‘Australian terrier’	‘small greyish wire-haired breed of terrier from Australia similar to the cairn’
‘n02102040’	‘English springer, English springer spaniel’	‘a breed having typically a black-and-white coat’
‘n02066245’	‘grey whale, gray whale, devilfish, Eschrichtius gibbosus, Eschrichtius robustus’	‘medium-sized greyish-black whale of the northern Pacific’
‘n02509815’	‘lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens’	‘reddish-brown Old World raccoon-like carnivore; in some classifications considered unrelated to the giant pandas’
‘n02124075’	‘Egyptian cat’	‘a domestic cat of Egypt’
‘n02417914’	‘ibex, Capra ibex’	‘wild goat of mountain areas of Eurasia and northern Africa having large recurved horns’
‘n02123394’	‘Persian cat’	‘a long-haired breed of cat’

Table 2.1 ImageNet WordNetID, synsets and descriptions [20]

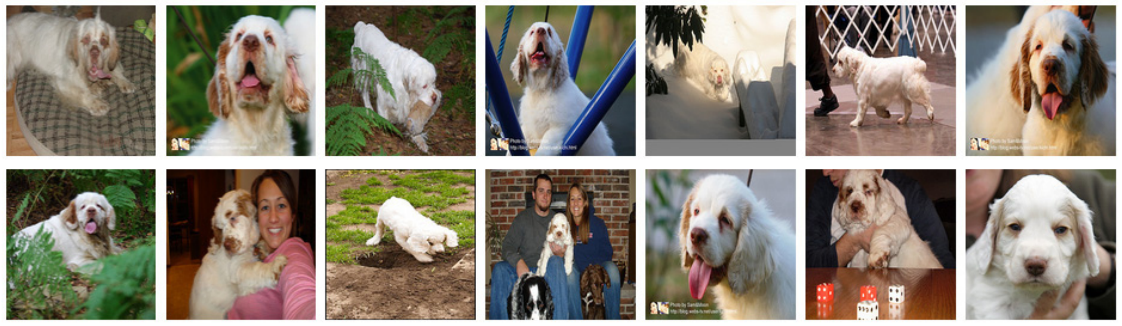
The ImageNet Dataset for the classification task has 1.27 million training images and 50,000 validation images. They were collected from the internet and manually labeled through Amazon Mechanical Turk crowd sourcing tool by using synsets in WordNet as queries. These images are spread over 1,000 classes, for example natural scenes, animals and human made objects. Table 2.1 shows 10 object categories.

The ImageNet dataset is by far the most difficult challenge for image classification algorithms. The reasons are not only the large number of images and varieties within each class, such as different backgrounds, resolutions, objects combinations, exposures and angles, but also the similarities between the different classes. For example, there are more than 100 species of dogs. They look alike but have different class labels, such as “Brittany spaniel” and “Clumber spaniel” shown in Figure 2.1a and 2.1b. Other animals, for instance “kit fox”, are included in this dataset as well. It has a similar shape and fur pattern as dogs. Some image classes belong to human made objects like “beer glass”, “beer bottle”, “wine bottle” and “cocktail shaker” in Figure 2.1c-f. They usually have same the background and are mixed together. “Beer glass” appears with “beer bottles” in Figure 2.1d; “wine bottle” appears with “beer glass” in Figure 2.1e; “cocktail shaker” combines with other bottles and glasses in Figure 2.1c. Human made objects have similar sizes and shapes. Many of them have a simple shape and less texture, which makes these objects not easy to be distinguished from each other, especially the foreground objects which only occupy a small area of the image or are far away from camera. Another kind of image class is human made scenes, for instance “toyshop” in Figure 2.1g. The images in this class do not share common objects. They could be any toys or an outdoor picture of a toyshop only.

All the above conditions dramatically increase the difficulty for training and testing. The ILSVRC classification contest uses the top-5 error rates for model evaluation. If the true validation image label is among the top-5 model prediction, then the result is marked as right, or else the prediction is wrong. Concretely, each image i has a unique class label C_i. The model is allowed to return 5 prediction labels c_{i1} ... c_{i5}, and is marked as correct if $c_{ij} = C_i$ for some j. Let the error prediction $d_{ij} = d(c_{ij}\ ; C_i)$ be 1 if $c_{ij} \neq C_i$ and 0 otherwise. The error rate of model is the fraction of test images on which the algorithm makes error prediction:

$$error\ rate = \frac{1}{N}\sum_{i=1}^{N} \min_j d_{ij}$$

(a) Brittany spaniel

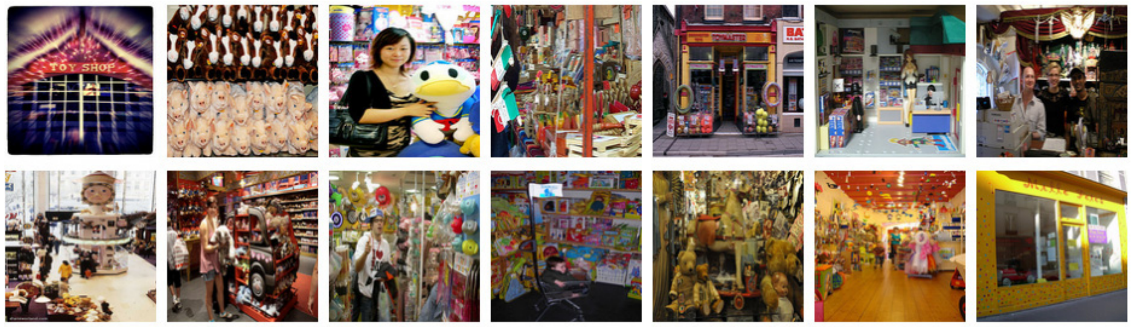

(b) Clumber spaniel

(c) Cocktail shaker

(d) Beer glass

(e)Wine bottle

(f) Beer bottle



(g) Toyshop

Figure 2.1 Images from [20] dataset

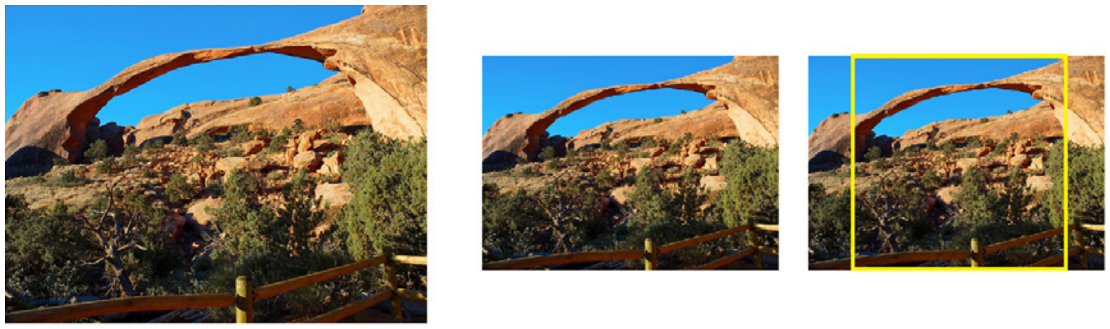
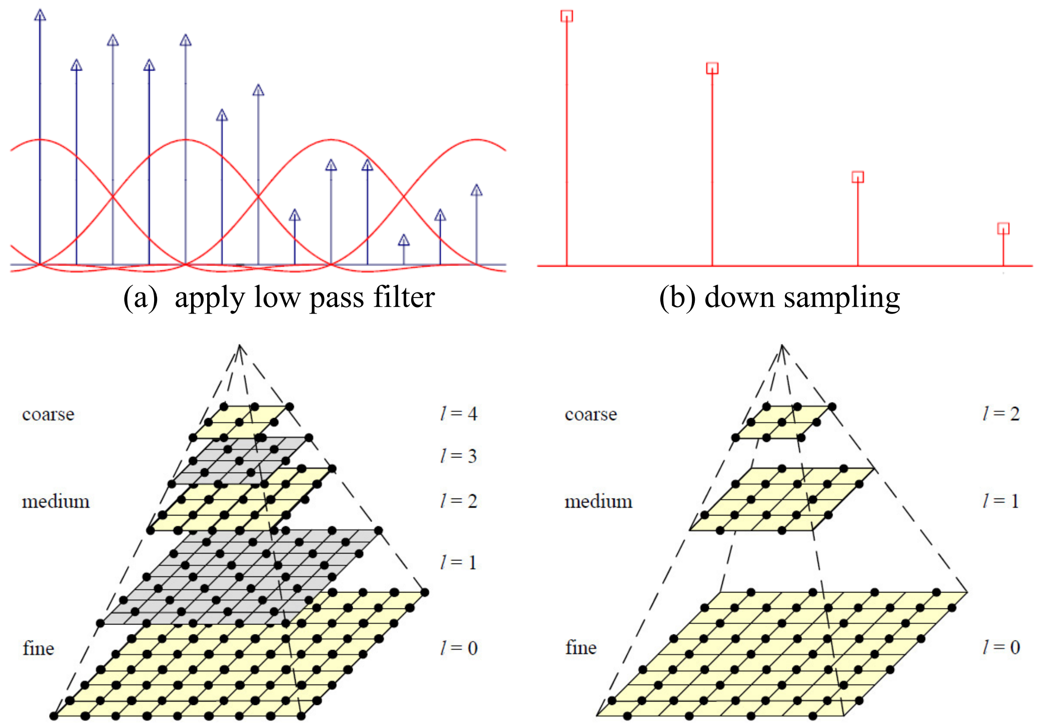

(a) source image (b) image resize (c) image crop

Figure 2.2 Transform an image of [20] dataset.

In this thesis, we chose 100 of 1,000 categories at step 10 based on the ImageNet categories list. This subset includes many animals and 14 of 100 categories are different species of dogs, for example "silky terrier", "white terrier", "cairn terrier", "cocker spaniel" and "Scottish deerhound". Human made object classes contain "beer glass", "wine bottle", "cocktail shaker" and "pill bottle". Human made scenes include "toyshop". Most of the classes have 1,300 training images. If some classes have less than 1,300 images, then we created new training images by randomly choosing from existing ones and flipping them right and left to make these classes have exactly 1,300 images. As all image data were collected from the internet, they have different sizes, color channels in RGB, CMYK and gray. We transformed all images in both training and validation sets into 3 channel RGB format. To make sure the image layer has the same input size, we resize and crop images into 256×256 squares as shown in Figure 2.2. Each image is subtracted into zero mean instance before they are fed into the network.

2.2 2D wavelets transform

Wavelets were first shown as a powerful tool for signal processing and analysis called multi-resolution theory in [4,15]. For the image domain, a simple structure to represent multi-resolution is image pyramids (Figure 2.3c). It is acquired by applying a low



(a) apply low pass filter (b) down sampling

(c) image pyramids (d) wavelets pyramids

Figure 2.3 Two types of image pyramids [25]

pass filter to avoid aliasing and down sampling (Figure 2.3.a-b). Image pyramids are used widely in computer vision applications. Many researchers use wavelets pyramids as an alternative (Figure 2.3d). It provides another way to decompose a signal into frequency components that are closely related to image pyramids.

Both image pyramids and wavelets pyramids decompose an image into multi-resolution descriptions. The major differences between these two methods are:

1. Traditional pyramids are over complete. They use more pixels than the original image to represent the decomposition. Whereas wavelets provide a tight frame. They keep the same size of the source image [25].

2. Wavelets are more orientation selective than regular band-pass pyramids [25], such as steerable multi-scale transform in 4 different orientations in Figure 2.4.

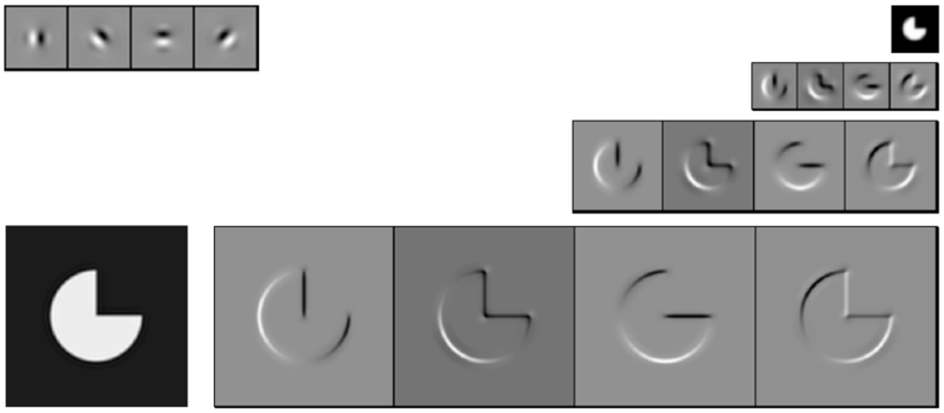


Figure 2.4 Steerable filter for different orientations [22]

2D wavelets functions are defined as below:

$$\psi^{H}(x,y) = \psi(x)\varphi(y)$$

$$\psi^{V}(x,y) = \varphi(x)\psi(y)$$

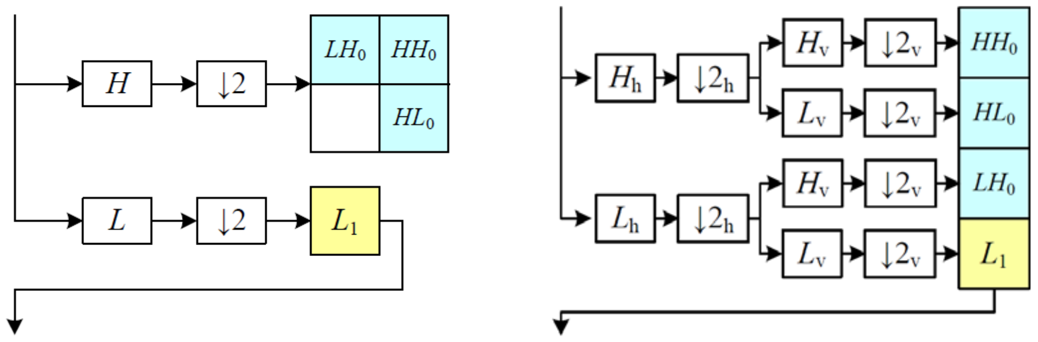
$$\psi^{D}(x,y) = \psi(x)\psi(y)$$

$\psi^{H}(x,y), \psi^{V}(x,y)\ and\ \psi^{D}(x,y)$ are horizontal, vertical and diagonal 2D wavelet kernel functions, which are separable by 1D wavelet kernel functions $\psi(x)$ and $\varphi(x)$:

$$\psi_{j,k}(x) = 2^{\frac{j}{2}}\psi(2^{j}x - k)$$

$$\varphi_{j,k}(x) = 2^{\frac{j}{2}}\varphi(2^{j}x - k)$$

k defines the translation position of 1D function along the *x*-axis. *j* defines the scale of wavelets and controls the width along *x*-axis. $2^{\frac{j}{2}}$ controls wavelet amplitude.



(a) High level chart (b) detail chart

Figure 2.5 2D wavelets decomposition flow chart [25]

Figure 2.5a shows a high-level diagram of one stage of the recursive analysis pipeline of 2D wavelets decomposition. In this diagram, the high-pass filter followed by decimation keeps 3/4 of the original pixels and 1/4 of the low frequency coefficients are passed to the next stage for further analysis. In practice, the filtering is usually broken down into two separable sub stages, as shown in Figure 2.5b. This produces three detail wavelets images, which are called the high-high (HH), high-low (HL), and low-high (LH) images. The high-low and low-high images for the horizontal and vertical edges and gradients, while the high–high image is for diagonal edges and gradients. In detail, at the first stage, the input image is applied by high and low pass filters and down sample in column by one-half. In the second stage, the output of the previous stage applies high pass and low pass filters in row and down sample by one-half again. The final output has four images. The L_1 image is the result of the input image is applied by low pass filters twice. It can be further decomposed for the next recursion. The total size of the four detail images is same as the input image.

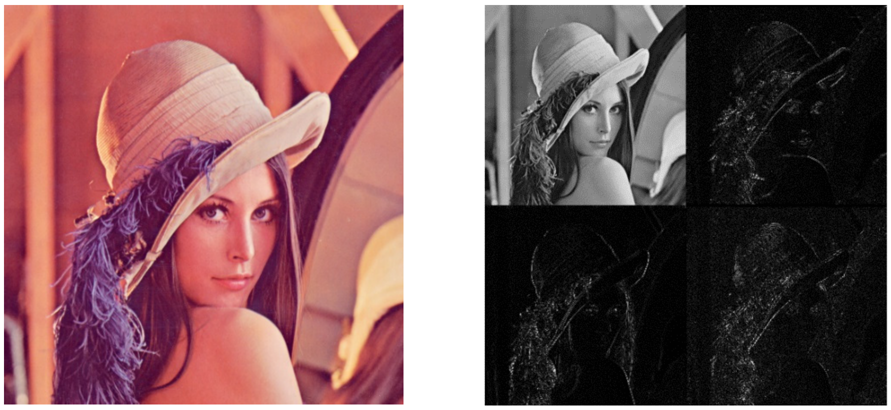

(a) Image of Lena (b) 2D wavelets decomposition

Figure 2.6 2D wavelets decomposition

Figure 2.6 shows an example of 2D wavelets decomposition. In three detailed horizontal, vertical and diagonal images, only high frequency components have large wavelet coefficients. Most of the plain area has very small or zero coefficients.

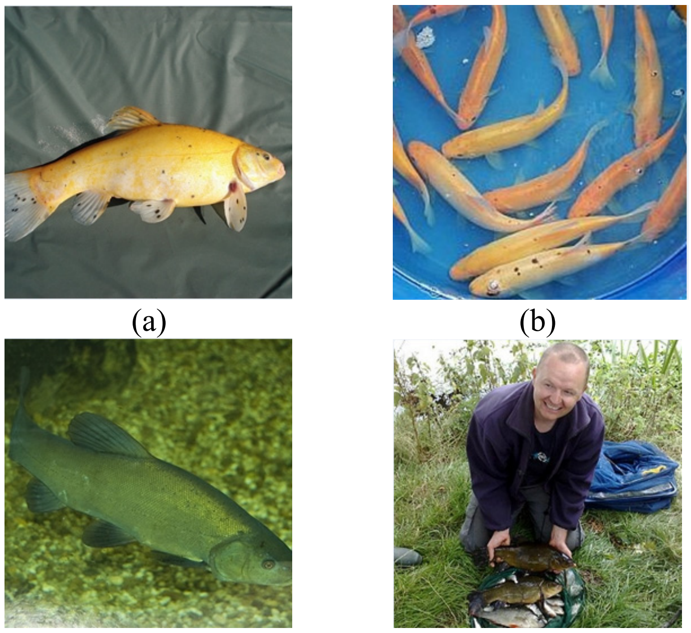

(a) (b) (c) (d)

Figure 2.7 Four images in [20] dataset “ Tench, Tinca tinca”

2.3 Image complexity analysis

Most of the popular CNN models train the source images minus a global mean value directly. Image processing is not involved much. Our model uses the wavelets transform and bilateral filter to process the data. This step brings more findings. Now we have a subset of the ImageNet dataset with 100 classes and each class has 1,300 training images. After reviewing images, we find a large variety of each class. Figure 2.7 shows four images of the same class “Tench, Tinca tinca”. It is a freshwater dace-like game fish of Europe and western Asia noted for ability to survive outside water. Figure 2.7a only has one fish and the background is flat. Figure 2.7b has fifteen fish and the background is still plain. Figure 2.7c has one fish but with a textured background with small cobblestones. This image is visually more complex than previous 2 images. Figure 2.7d is the most complex one. It shows many different objects with a clustered grassland background. The fish only occupies a very small region of the whole image in the bottom. However, this image still has the same label as all the other images. We suppose figure 2.7a and 2.7b are ideal training cases; Figure 2.7c and Figure 2.7d are not ideal training cases. CNN can extract more features from the fish in the first two images. In comparison, the last two images, especially figure 2.7d is too noisy. It contains more many objects, but not limited to the fish. Therefore, the final feature vector includes the abstraction of multiple objects.

Overall, we presume the simple images of each class are ideal training examples and the complex images are bad examples. The complex images are more likely to contain clustered backgrounds and irrelevant objects. There are many ways to measure image complexity, such as entropy filter [24], which computes the entropy of local region of the image and wavelet coefficients. Entropy filter is very sensitive to noise (figure 2.8)

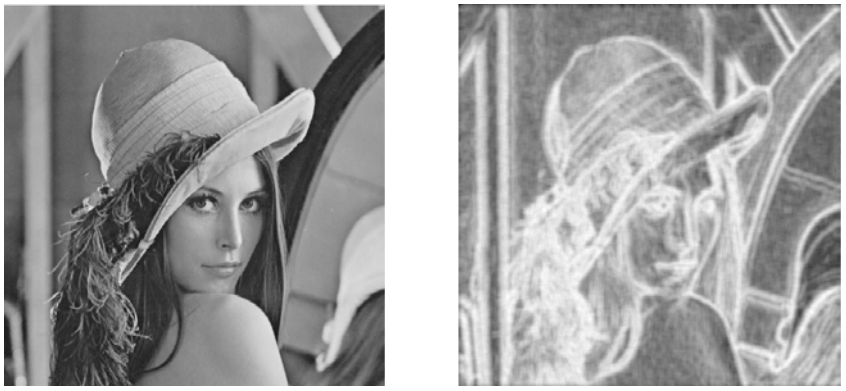

(a) gray scale image (b) apply entropy filter

Figure 2.8 Apply entropy filter at size 7×7 to a gray image

We use 2D wavelets transform in section 2.2. It computes fast as the kernel are separable and is not sensitive to noise as image entropy. The large gradients in local image region have large wavelet coefficients. So image complexity has positive correlation with the quantity of the large wavelet coefficients. We transformed all 256×256 training and validation RGB images into gray scale, performed first order wavelet transform and normalized wavelet coefficients between 0 and 1. We defined an index *C* below that counts the number of normalized coefficient $d_{x,y}^{k}$, which is larger than 0.5 threshold from three detail images in horizontal, vertical, and diagonal directions, where *k* is the index of the detail images and *x, y* are pixel coordination. This *C* index measures the complexity of each image.

$$C = \sum_{k} \sum_{x,y} c_{x,y}^{k} \begin{cases} 1, d_{x,y}^{k} > 0.5 \\ 0, d_{x,y}^{k} \leq 0.5 \end{cases}$$

Figure 2.9 shows four images of "black swan" class. The number in the title bar shows the index *C* of each image. The index of the first image is 6,871. This value is relative small, since the background is plain. These 2 black swans are far from the camera.

In the last image, 2 black swans are near the camera. The texture area takes a large proportion, then the index *C* has large value 32,266.

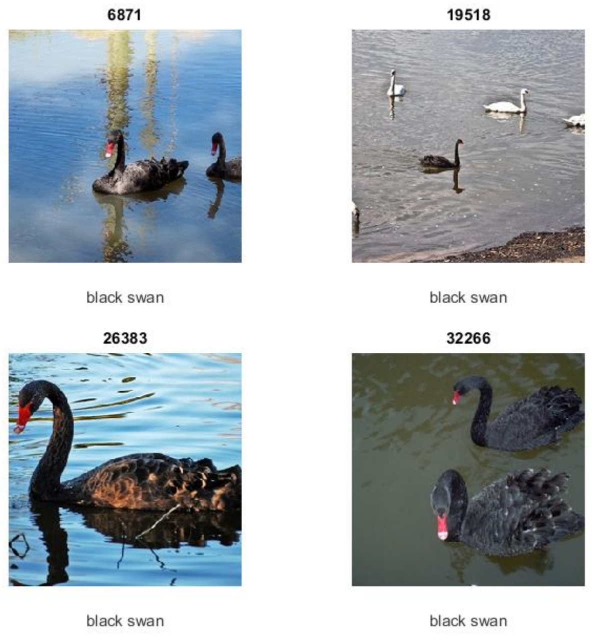


Figure 2.9 Images of "black wan" in [20] dataset with *C* index

2.4 Bilateral filtering

Bilateral filter is a powerful tool to smooth images while preserving edges. It started from the work of [1] on nonlinear Gaussian filters in 1995 and was revised by [23]. Bilateral filter has been used in many image applications, such as denoising, texture editing and relighting, tone management, de-mosaicking, and optical-flow estimation.

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2}\exp(-\frac{x^2+y^2}{2\sigma^2})$$

Bilateral filter is similar to Gaussian filter, which is defined in the above function in a continuous form in Figure 2.10. Gaussian filter is the simplest tool to smooth images. The coefficients of Gaussian filter can be written in a matrix form in Figure 2.11. The filtering process is the convolution between Gaussian filter and the input image as the formula below. The output pixel $g(i,j)$ is determined as a weighted sum of the input pixel values. $h(k,l)$ is a Gaussian filter, which defines the weights for each pixel.

$$g(i,j) = \sum_{k,l} f(i+k, j+l) h(k,l)$$

The convolution operator could be in a more compact form:

$$g = f \bigotimes h$$

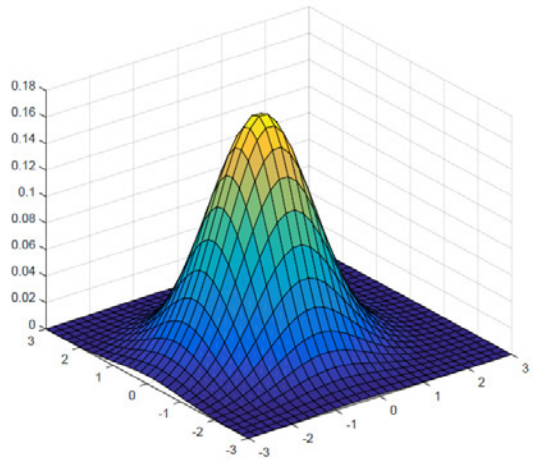


Figure 2.10 2D Gaussian filter.

$\frac{1}{7.6}$

0.2	0.25	0.3	0.25	0.2
0.25	0.35	0.4	0.35	0.25
0.3	0.4	0.6	0.4	0.3
0.25	0.35	0.4	0.35	0.25
0.2	0.25	0.3	0.25	0.2

Figure 2.11 2D discrete form of Gaussian filter.

Figure 2.12 shows an example of an image with blurry effects after applying Gaussian filter. The detail information, such as the edges and textures, in the image is reduced a lot. The flowers and leafs are very blurry. The edges of vase and bowls are not clear, since the high frequency components are suppressed. But the plain area in the background has less impact.

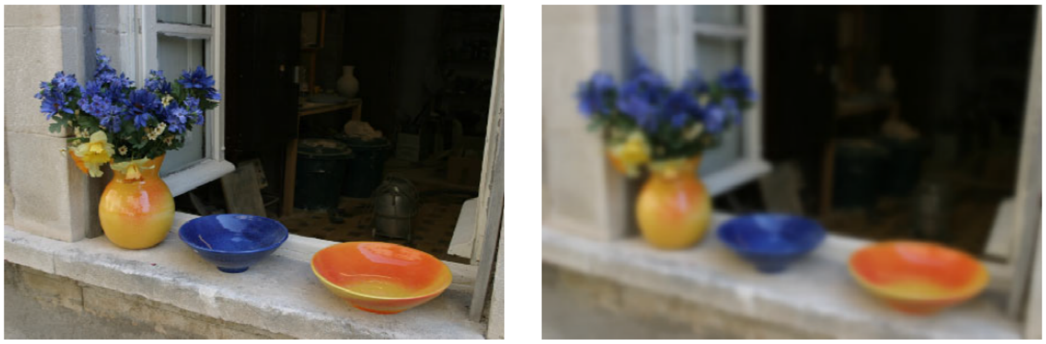


Figure 2.12 Source image (left) applies Gaussian filter to show blurry effects [25]

The bilateral filter is defined as a weighted sum of the input pixels as well. The bilateral filter takes into account both spatial domain and range domain. Range domain is related to the radiometric, such as color intensity in the image. The output pixel value $g(i,j)$ depends on a weighted combination of the nearby pixel values.

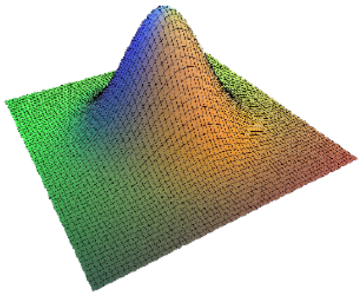
$$g(i,j)=\frac{\sum_{k,l} f(k,l)\omega(i,j,k,l)}{\sum_{k,l} \omega(i,j,k,l)}$$

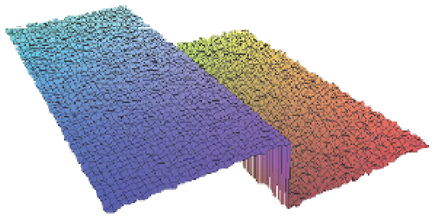
The weight coefficients $\omega(i,j,k,l)$ are the product of the domain filter $d(i,j,k,l)$and the range filter $r(i,j,k,l)$ below. The domain filter is the major filter uses a 2D Gaussian function. The range filter measures the color intensity similarities to the center pixel. σ_d and σ_r are the standard deviations of the domain and range filter. Figure 2.13 shows three filters and a step edge output image. We can see the noise is removed but the edge is still preserved.

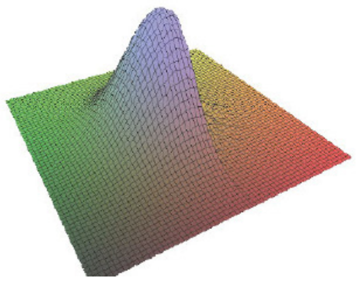
$$d(i,j,k,l) = \exp(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2})$$

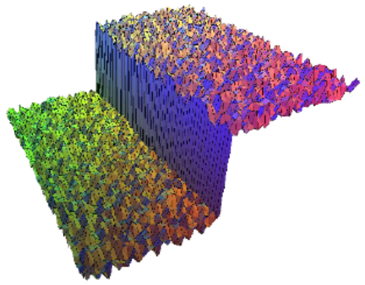
$$r(i,j,k,l) = \exp(-\frac{\|f(i,j) - f(k,l)\|^2}{2\sigma_r^2})$$

$$\omega(i,j,k,l) = \exp(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|f(i,j) - f(k,l)\|^2}{2\sigma_r^2})$$


(a) Domain filter (Gaussian)


(b) range filter


(c) Bilateral filter


(d) Noisy step edge input

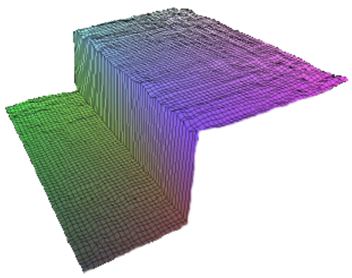

(e) Bilateral filtered step edge output

Figure 2.13 Illustration of bilateral filter [18]

Chapter 3 Convolutional Neural Networks

3.1 Introduction

CNN were inspired of the cat's visual cortex from the research work of [8]. The cells in the visual cortex have very complex patterns. These cells can capture a small sub-regions of the visual field, which is called a receptive field. All these sub-regions are combined together to cover the whole visual field. There are two basic cell types: 1) the simple cells respond to edge like patterns within receptive field. 2) The complex cells have larger receptive field and are locally invariant to the position.

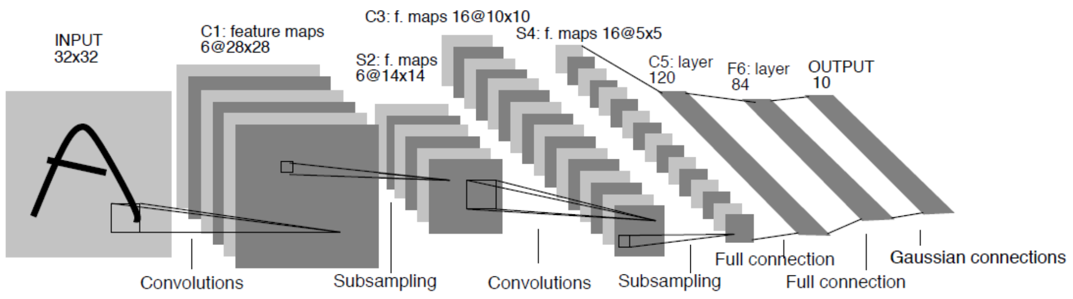
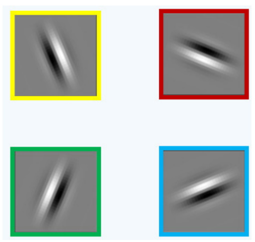


Figure 3.1 Convolutional neural networks of LeNet5 [12]

CNN are comprised of several convolutional and pooling layers, which are followed by one or more fully connected layers in a standard multilayer neural networks. CNN are easier to train and have less parameters than the completely fully connected networks with the same number of hidden units. CNN was applied in handwritten digits recognition in [12] at beginning. The networks structure is in figure 3.1. This is a typical multilayer networks. The lower layers are composed of 2 convolutional layers and 2 pooling layers. The upper layers are the fully connected layers. The input of the first fully connected layer is from the features maps of the previous layer. The final output of the whole networks has 10 units, which have the number of class labels from 0 to 9.

(a) input image


(b) edge filters

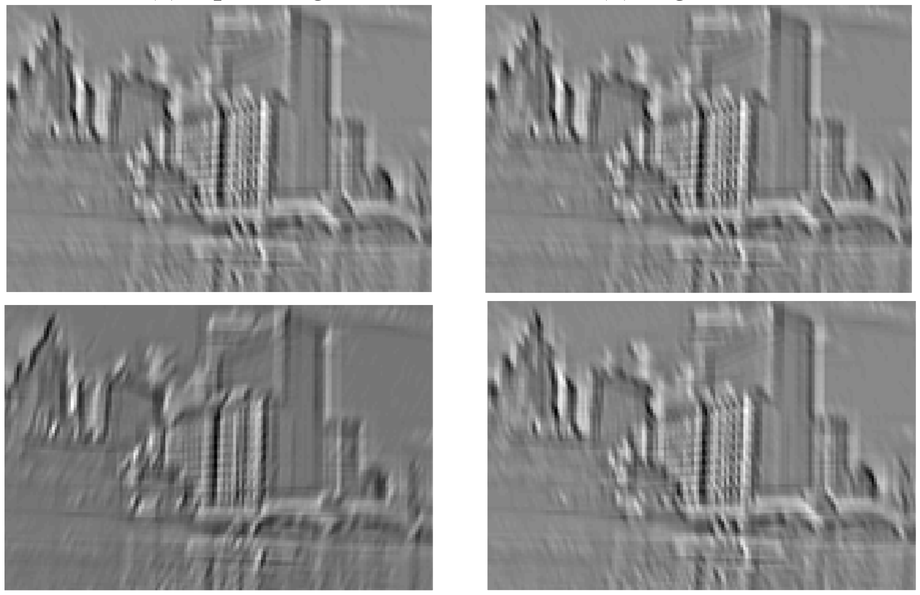
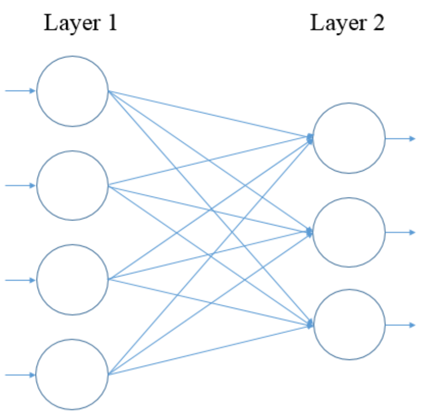

(c) 4 feature maps of convolution output

Figure 3.2 Image convolution with edge filters [25]

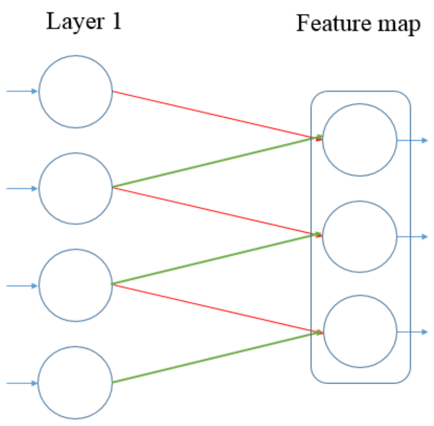
3.2 Convolutional layer

The natural images have “stationary” property, which means the statistics of one local region of the image could be same or similar to the other local regions. This suggests the features learned at one region can be applied to the other regions as well. As shown in Figure 3.2, edge filters apply to each location in the image and then generate 4 feature maps. Features reuse means we need less parameters. Suppose we have a learned feature at a 16×16 patch. We use this 16×16 feature to convolve the whole image and obtain the response value at each location of the image. For CNN precisely, the output of convolutional layer is obtained by running convolution for the input images with the filters, adding a bias term and then applying a non-linear activation function. The output value of convolutional layer composes feature map. Each filter creates one feature map. We denote x as an input image and the kth feature map as a^k. The filter is determined by the weights matrix W^k and biases b_k. The feature map a^k could be obtained by the formula below:

$$a^k = f(W^k \bigotimes x + b_k)$$



(a) full connections



(b) convolution layer

Figure 3.3 Weights sharing

The benefit of convolutional layer is weights sharing. This mechanism dramatically reduces the number of parameters. Figure 3.3a shows the full connections between 2 adjacent layers. The total number of weights for a full connection layer is $m \times n$, where m is number of neurons in layer 1 and n is the number of neurons in layer 2. Suppose the input layer is an image at size of 256×256 with single channel and we use a full connection layer after the image layer, then each neuron in the full connection layer has 65,536 input connections, which respond to 65,536 parameters. By contrast, the convolutional layer only needs $n \times n$ weights plus one bias, where n is the width of the filter. Some CNN models use random sparse convolutional layer. Suppose a convolutional layer produces 128 feature maps and the following convolutional layer uses 96 filters. These 96 filters do not convolve all 128 feature maps, but only randomly chose 64 feature maps each time. So random sparse convolutional layer reduces the number of parameters.

3.3 Pooling layer

Pooling is a subsampling procedure. It summarizes the output of the adjacent neurons in the same feature map. This aggregation operation computes the max or average value of a small region in the feature map. If the pooling is calculated continuously in the image, then the pooling produces translation invariant features. This means the same features are still active even if the image has small translation. The translation invariant features are often desirable, since in many image and audio classification tasks the labels of the data are the same even if the data have phase changes. For example, if we have a hand write digit and translate it from left to right in figure 3.4, we expect the classifier still accurately classify it as the same digit.

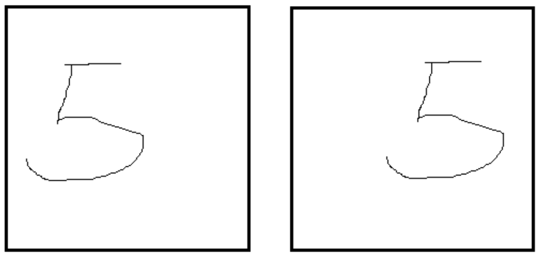


Figure 3.4 Example of image translation from left to right

Pooling can be overlapping or none overlapping in Figure 3.5. Overlapping pooling means the consequential pool regions in the image has intersection; none overlapping pooling does not. In general, overlapping pooling has less overfitting issue.

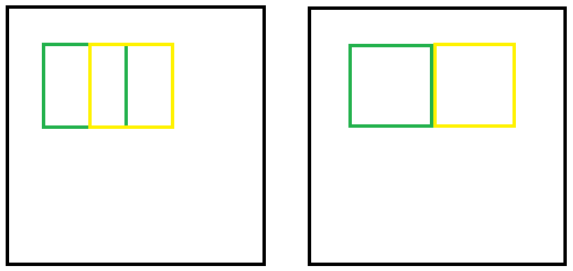

(a) overlapping (b) none overlapping

Figure 3.5 Overlapping and none overlapping

3.4 Forward and backward propagations

CNN uses forward and backward propagation methods to learn the weight and bias parameters. As described in section 3.2, the forward propagation in convolution layer is a filtering process plus a bias term and pass through an activation function. The initial values of the weights and biases can be randomly sampled through a zero mean Gaussian function with a small standard deviation. This random initialization serves the purpose of symmetry breaking. All parameters are updated during the backward propagation.

Suppose we have *m* training examples $\{(x_1,y_1), \ldots (x_m, y_m)\}$, where x_i is a data vector and y_i is a class label. We can train CNN through gradient descent. In detail, for a single training example (x_i,y_i), we define a square error cost function as below, where $f_{W,b}(x)$ calculate the output value of forward propagation. *W* and *b* are weights and biases

$$J(W,b;x_i,y_i) = \frac{1}{2}\|f_{W,b}(x_i) - y_i\|^2$$

For all *m* training examples, we can define an overall cost function as below.

$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} J(W,b;x_i,y_i)$$

$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} \left(\frac{1}{2}\|f_{W,b}(x_i) - y_i\|^2\right)$$

To prevent overfitting, the cost function usually add a regularization term or weight decay term below, which decreases the magnitude of the weights. Weight decay is not applied to the bias terms. W_{ij}^{l} are the weights, where l is the layer index and ij are the neuron indexes of the adjacent hidden layers. The weight decay parameter λ controls the relative importance between the sum of square error term and the regularization term. Larger λ makes W_{ij}^{l} becomes smaller Small W_{ij}^{l} value means model has less variance and shows more generalization ability. We will extend more about the regularization in section 3.5.

$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\|f_{W,b}(x_i) - y_i\|^2\right) + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(W_{ij}^{l})^2$$

This cost function can be used for both classification and for regression tasks. For the classification task, we set y = 0 or 1 to represent two class labels through a sigmoid function, which outputs values between 0 and 1. For the other activation function like tanh, we can use -1 and +1 to denote the labels.

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

The goal of training is to minimize $J(W, b)$ as a function of W and b. This can be done by updating W and b through gradient descent. Each iteration of gradient descent updates the parameters W and b at a small scale. α is learning rate that controls this scale.

$$W_{ij}^{l} = W_{ij}^{l} - \ \alpha\frac{\partial}{\partial W_{ij}^{l}}J(W,b)$$

$$b_{i}^{l} = b_{i}^{l} - \alpha\frac{\partial}{\partial b_{i}^{l}}J(W,b)$$

The derivative of the cost function $J(W, b)$ is defined as below

$$\frac{\partial}{\partial W_{ij}^{l}} J(W,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{l}} J(W,b;x_i,y_i) + \lambda W_{ij}^{l}$$

$$\frac{\partial}{\partial b_{i}^{l}} J(W,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial b_{i}^{l}} J(W,b;x_i,y_i)$$

For each neuron *i* in layer *l*, we can compute an error term δ_i^l that measures the gap between the output value and the true value. For the output of the last layer, we can directly define the error term as $\delta_i^{n_l}$, which is difference between the output value and the true target value that is from the label data *yi* , where layer n_l is the output layer.

$$\delta_i^{n_l} = -(y_i - f(wx_i + b_i)^{n_l}) \cdot f'(wx_i + b_i)^{n_l}$$

For the error of *i*th neuron in hidden layer *l*, δ_i^l is defined as below:

$$\delta_i^l = \sum_{j=1}^{s_{l+1}} W_{jl}^{l} \delta_i^{l+1} \cdot f'(wx_i + b_i)^l$$

Then the partial derivatives are defined as:

$$\frac{\partial}{\partial W_{ij}^{l}} J(W,b;x_i,y_i) = f(wx_i + b_i)^{n_l} \cdot \delta_i^{l+1}$$

$$\frac{\partial}{\partial b_{i}^{l}} J(W,b;x_i,y_i) = \delta_i^{l+1}$$

We usually write the above formulas in the matrix form:

$$\Delta W^l := \Delta W^l + \nabla_{W^l} J(W,b;x,y)$$

$$\Delta b^l := \Delta b^l + \nabla_{b^l} J(W,b;x,y)$$

And *W* and *b* are updated by:

$$W^l = W^l - \alpha\left[\left(\frac{1}{m}\Delta W^l\right) + \lambda W^l\right]$$

$$b^l = b^l - \alpha\left(\frac{1}{m}\Delta b^l\right)$$

The training step is to calculate the gradient descent repeatedly to reduce the value of the cost function, which is equivalent to reduce the training error.

If apply momentum method [30] for the parameter updates, both current gradients and previous updates are considered. The momentum term is β, which control decay rate of the previous updates. β is between 0 and 1 (e.g. 0.9). Current gradients at time t and previous updates at time t-1 are in the opposite signs in order to damp the oscillations in directions of high curvature surface. Then W and b are updated by below functions:

$$\Delta W(t) = \beta \Delta W(t-1) - \nabla_W(t)$$

$$\Delta b(t) = \beta \Delta b(t-1) - \nabla_b(t)$$

3.4.1 Stochastic Gradient Descent

Using the full training dataset to update W and b in each cycle tend to converge to a local optima. Furthermore, computing the cost and gradient through the entire training set is very slow and is not feasible on a single machine if the training dataset is too big to load into memory. Stochastic Gradient Descent (SGD) [31] updates and computes the gradient of the parameters using mini data batch. Training a mini data batch each time reduces the variances in the parameters and leads to a more stable convergence. Mini data batch allows the computation to take the advantages of optimized matrix operations, which is well vectorised for the cost and gradient. Mini batch usually contains 128 or 256 training examples. The ideal size of a mini data batch can vary based on different tasks.

3.5 Combat overfitting

Overfitting is a major problem for many machine learning algorithms including CNN. When a model has a large number of parameters, then this model can fit almost any complex function. The Large parameters mean high freedom and are able to describe any dataset. But they may not capture the underlying pattern. Even if a model works well on the training dataset, this model may still fail to make prediction on the validation dataset. The true test of a model is its ability to make predictions on the validation dataset.

Overfitting happens when a model starts to memorize the dataset in the training phase, but fail to generalize for the trend. Under very extreme condition, if the number of the parameters is the same as or larger than the number of training examples, the training process can perfectly predict the training dataset simply by memorizing the whole dataset. However, this model will typically fail drastically when making predictions for the validation dataset. Figure 3.6 shows an example of overfitting for a regression task. The training examples are in green and the testing examples are in red. The final model is a high polynomial function, which perfectly fits all training examples with no errors. But for the test examples, none of them are captured by the model. This model shows very bad generalization ability for the test examples.

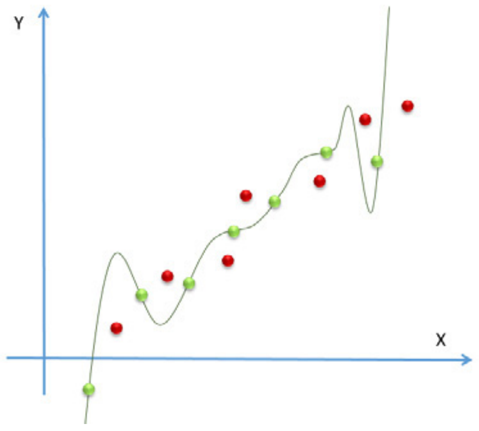


Figure 3.6 Example of overfitting for regression.

3.5.1 Regularization

Many methods can reduce overfitting. Regularization [26] is a common and effective method for neural networks. There are 2 major variants for regularization that are

L_1 and L_2 norms. L_1 norm penalizes the large weights and tends to make the model prefers small weights and zeros. This is essentially a feature selection procedure, since many features have less or no contributions for regression or classification. Only a few features play the key roles for learning algorithms. L_1 norm does not fit for neural networks, since it is not differentiable. Therefore, L_2 norm is a proper choice for CNN. L_2 norm is called weight decay term, since it makes weights become smaller.

During the training phase, CNN try to create a non-smooth function, which fits the training dataset with noise. A non-smooth function needs very large weights. Once the networks have overfitting on the training dataset, the weights increase in a very large magnitude. There is very strong correlation between large weights and overfitting. L_2 regularization just penalizes this process, which reduces the weights. If the weights become small, CNN tend to produce a smoother function which changes slowly. In other words, smaller weights mean the function has lower complexities, which make each feature has small contribution. Thus the whole model is not decided by a few features with large weights. In general, a simpler model provides a more powerful explanation for the data and should be preferred. Figure 3.7 shows the examples of regularization apply to the regression task. The blue curve is the true model and the red curve is fit model.

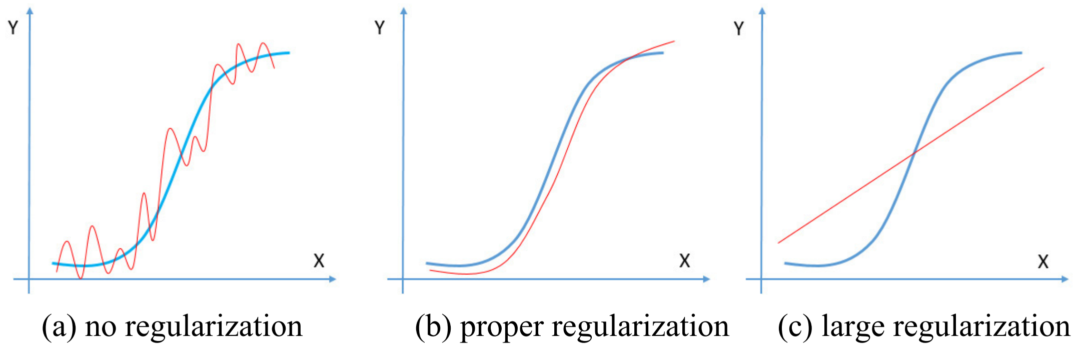


Figure 3.7 Examples of regularization for regression task

3.5.2 Dropout

Dropout [7] is another way to reduce overfitting. Unlike L_1 and L_2 norms, dropout does not add an extra term to the cost function, but changes the networks structure directly. Dropout randomly removes part of neurons in the networks during the training phase. The networks use the forward propagation to feed a mini data batch through the dropout networks and then back propagate through the same dropout networks. The training parameters W and b are updated during the back propagation. The next iteration repeats the same process but chose a new random subset of hidden neurons to remove from the networks. These neurons update the training parameters through a different mini batch. By repeating this process, the networks will learn the whole set of training parameters. During the test phase, all neurons are active for the prediction computation. Figure 3.8 shows the dropout networks, which remove part of neurons for training in the dash lines

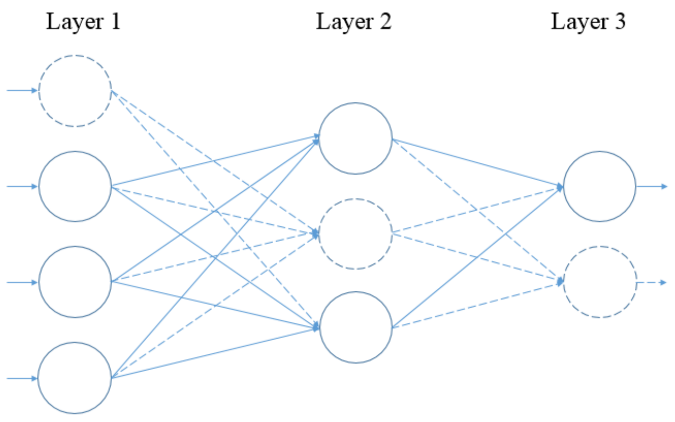


Figure 3.8 Example of dropout.

Dropout removes different neurons during the training phase is equivalent to train the different CNN on different mini batches. Thus, the dropout has the averaging effects for a large number of different CNN. The individual network may overfitting in the different way, then the whole networks during the test phase can reduce overfitting.

3.6 Softmax regression

Softmax regression (multinomial logistic regression) is derived from logistic regression [2]. It could be used for the class labels larger than two. In logistic regression we assume that the labels are in the binary form 0 and 1. Suppose we have the training dataset with *m* examples $\{(x_1, y_1) \ldots (x_m, y_m\}$, where x_i is an input data vector and y_i is a binary class labels {0,1}. The hypothesis is defined as below and θ is a parameter vector.

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^T x)}$$

θ is updated during training phase to reduce the value of cost function:

$$J(\theta) = -\left[\sum_{i=1}^{m} y_i \log h_\theta(x_i) + (1 - y_i)\log(1 - h_\theta(x_i))\right]$$

In softmax regression, y_i has more than 2 classes. For any input data vector x_i, the model computes a probability value $p(y = j \mid x_i)$ for each $j = 1, \ldots, k$, where *k* is the class label. The model estimates a probability by *k* different possible values. Thus, the model outputs a *k* dimensional vector for each x_i. All *k* probability values are sum into 1. So the hypothesis is extended as below for *k* different classes.

$$h_\theta(x_i) = \begin{bmatrix} p(y_i = 1|x_i;\theta) \\ p(y_i = 2|x_i;\theta) \\ p(y_i = 3|x_i;\theta) \\ \ldots \\ p(y_i = k|x_i;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} \exp(\theta_j^T x_i)} \begin{bmatrix} \exp(\theta_1^T x_i) \\ \exp(\theta_2^T x_i) \\ \exp(\theta_3^T x_i) \\ \ldots \\ \exp(\theta_k^T x_i) \end{bmatrix}$$

The cost function of softmax regression is defined as below, where 1{} is an indicator function. It produces value 1 if {} statement is true and 0 if {} statement is false.

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y_i = j\} \log \frac{\exp(\theta_j^T x_i)}{\sum_{l=1}^{k} \exp(\theta_j^T x_i)}\right]$$

Parameter $\theta = [\theta_1^T; \theta_2^T; \theta_3^T \ldots \theta_k^T]$. Each θ_i^T is updated by gradient descent as well. The gradient of $\theta_j$is:

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [x_i (1\{y_i = j\} - p(y_i = j | x_i; \theta))]$$

Then θ_j is updated through:

$$\theta_j := \theta_j - \alpha \nabla_{\theta_j} J(\theta)$$

3.7 Related work

CNN with a large number of parameters are the most complex model for image classification task. Researchers often feed the network with the large datasets and train for days or weeks. Thus, many methods try to speed up training procedure. The common way is to adopt parallel computing technology, especially under multiple GPU or GPU settings, because feature learning of each layer is similar and easy to extend. There are 2 major ways for parallel training: model parallelism and data parallelism [10] in figure 3.9. Model parallelism trains the different part of the networks on different hardware, such as training the first layer on GPU 1 and training the second layer on GPU 2. These 2 layers need to synchronize between each other. Data parallelism divides a big dataset into smaller ones and trains on different hardware at the same time. This method just simply extends one path to more paths by using more GPUs and each path only learn source images. Use more paths cannot improve classification accuracy. As shown in table 3.1, Model [10] uses 4 GPUs is not better than 1 GPU for the classification results. They all have a 42% top 1 error rate. The only benefit is reduce training time. Furthermore, the computation and networks complexity increase a lot on the parallel method, since the multiple GPUs model uses more complex feedforward and backward propagations. As shown in figure 3.10, there are 4 forward and backward propagations in fully connected layer. It is more complex than the single path model.

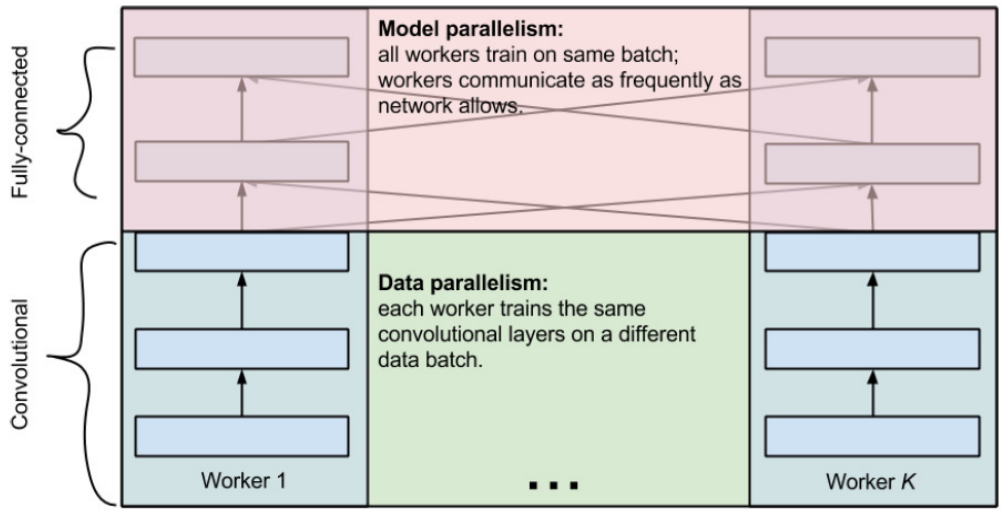

Model parallelism:
all workers train on same batch;
workers communicate as frequently as network allows.
Data parallelism:
each worker trains the same convolutional layers on a different data batch.
Fully-connected
Convolutional
Worker 1
Worker K
...

Figure 3.9 2 Parallelism training methods [10]

GPUs	Batch size	Cross-entropy	Top-1 error	Time	Speedup
1	(128, 128)	2.611	42.33%	98.05h	1x
2	(256, 256)	2.624	42.63%	50.24h	1.95x
2	(256, 128)	2.614	42.27%	50.90h	1.93x
4	(512, 512)	2.637	42.59%	26.20h	3.74x
4	(512, 128)	2.625	42.44%	26.78h	3.66x

Table 3.1 Scores for different GPU settings [10]

Another parallel method like [5] uses different resolution images that feed into different paths. After several convolution and pooling iterations, the feature maps from the output of the last convolutional layers of each path are aligned into the same size by up scaling the small ones (figure 3.12a). This method tries to learn the features from the different image pyramids, but does not speed up the learning speed. The high frequency components are still larger than the low frequency components after down sampling, such as the grass land is still complex than the sky after down sampling. This model needs an extra per pixel labeling path for training. The segmentation results show in figure 3.11b.

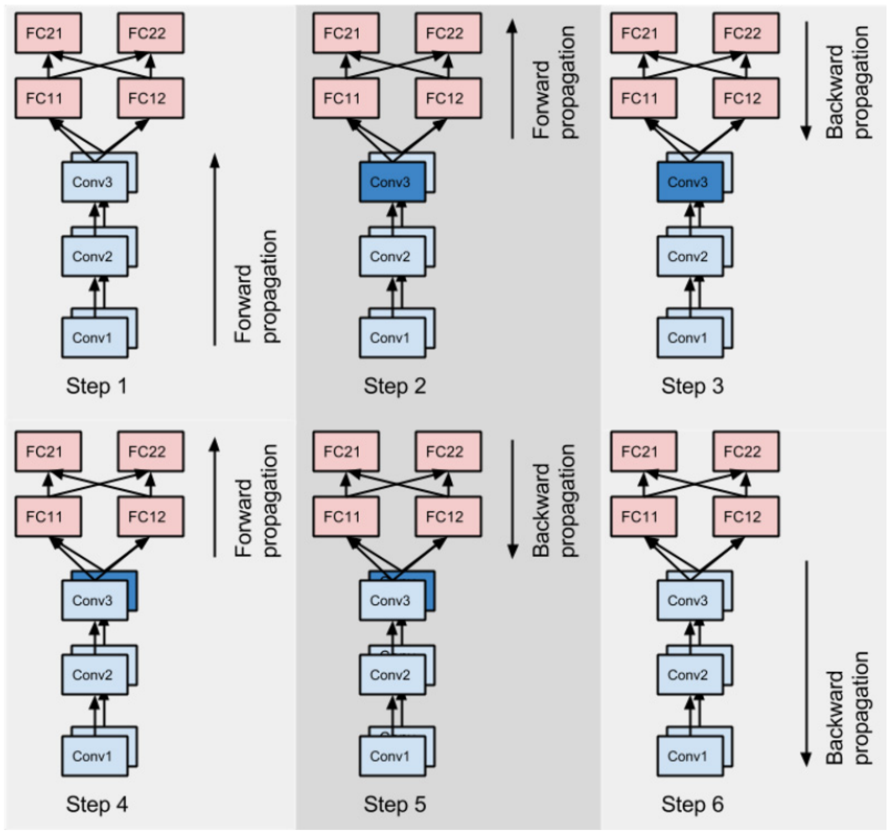
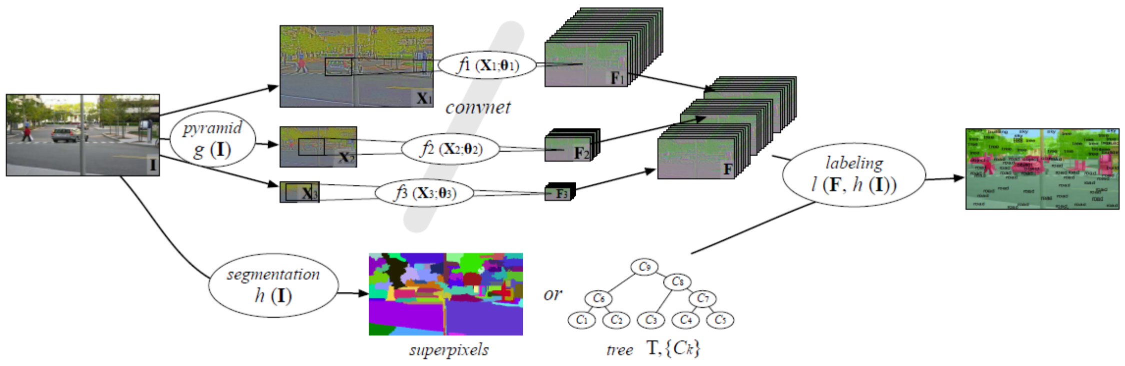
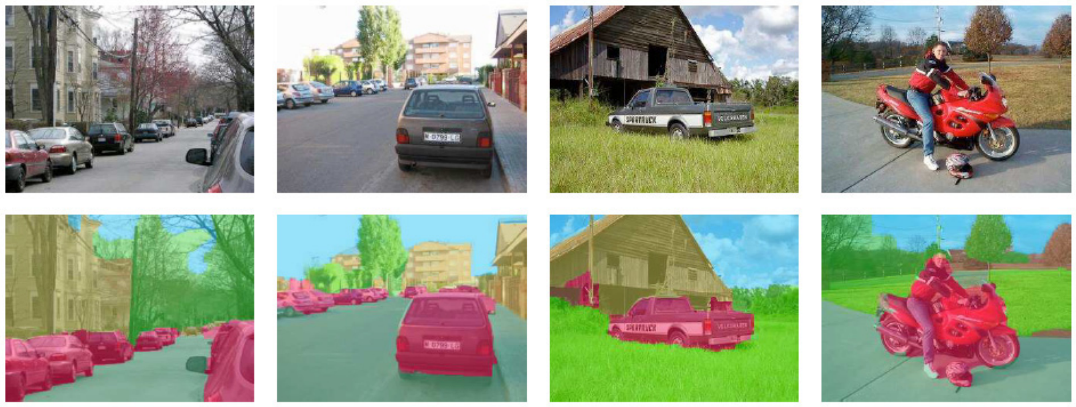


Figure 3.10 Feedforward and backward propagations of [10]



(a) 3-stage CNN on different pyramids.



(b) image segmentation

Figure 3.11 Parallel method for image segmentation [5]

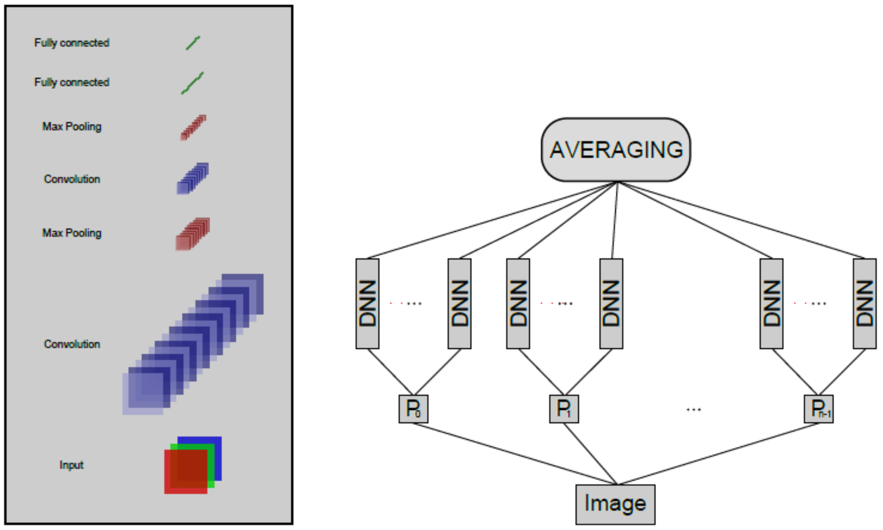


Figure 3.12 Multi-column CNN [3]

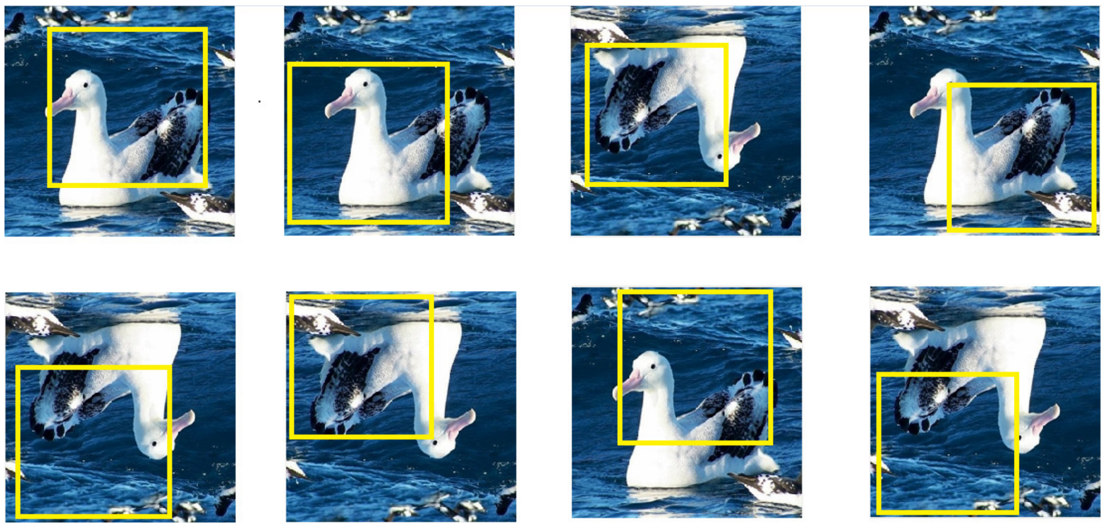
The model [3] uses multi-column networks as shown in figure 3.12. Each column feeds a processed image block *Pᵢ*. The final prediction is averaged over the output from multi-column. This model uses a typical ensemble learning method. However, the Different columns should learn the similar features, because each column uses the same way to process the input images.

Chapter 4 Comparing Different Images on CNN

4.1 Create dataset

As described in section 2.3, we use the index *C* to measure image complexity. The simple images are ideal training examples within each class, since they contains less noise and irrelevant objects. We sort all 1,300 training images of 100 classes by the index *C* defined in section 2.3 and separate them into 4 groups. The first group has 325 images with a small index. They are simple and ideal training examples. The fourth group contains the most complex images with a large index. This approach is applied to the validation set as well. Each class of the validation set has 50 images, which is not divisible by 4. Therefore, we chose the top 48 images after sorting by index C and then separate them into 4 groups. Each group has 12 validation images for each class. The first group contains the simple validation images and the last group contains the most complex validation images. We pair the *i*th group training set with the *i*th group of the validation set.

Both groups have 325 training images for each class and the total class number is 100. So we have 32,500 training images for both groups. The validation set of each group has 1,200 images. We adopt the data augment method in [11] to make more training cases through image translation and horizontal reflection. This is done by randomly cropping 224×224 regions of 256×256 source images and flipping them in them vertical direction at 50% chance in figure 4.1a. For the validation set data, we crop the center 224×224 regions of the image in figure 4.1b


(a) Random crop image for training

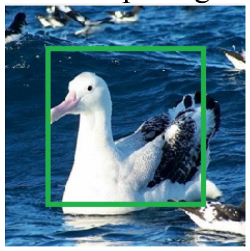

(b) Center crop for validation

Figure 4.1 Random crop the images of ImageNet [20] dataset

4.2 Experiment 1 comparing images on the network

We reproduce [29] network structure definitions (figure 4.2) and parameter settings. This has the best result for the image classification task in ILSVRC 2013. This experiment only compares different images on the same network but not compares different models. Thus, to make training faster, we do not use dropout for full connected layers. Removing drop out makes training and validation curves have very large gaps (figure 4.3), which is caused by overfitting. But dropout is applied to all experiments in Chapter 5

We run group 1 with the simple images dataset and group 4 with the complex images dataset on the above model independently for 10 cycles on NVIDIA 4GB GTX 760 GPU. Both groups have 325 mini batches for training and 12 mini batches for validation.

Each mini batch of training and validation sets contains 100 images with 100 different class labels. The image sequence in each mini batch is randomly permuted. The validation frequency is set to 10, which means it runs 1 validation mini batch after running 10 training mini batches. We run training set for 10 cycles, so the total training cases are up to 325,000 after applying the data augment method.

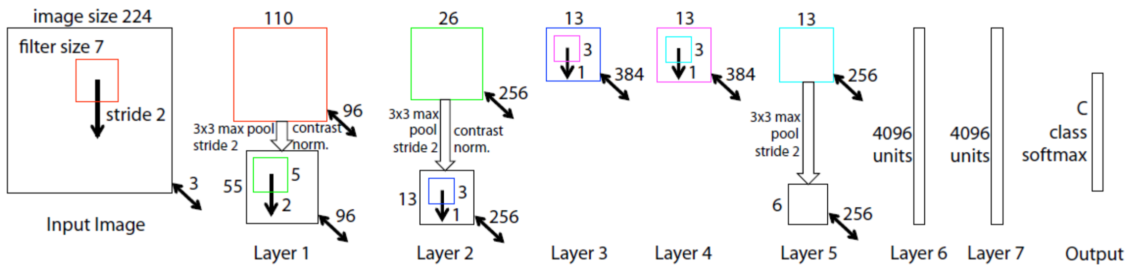
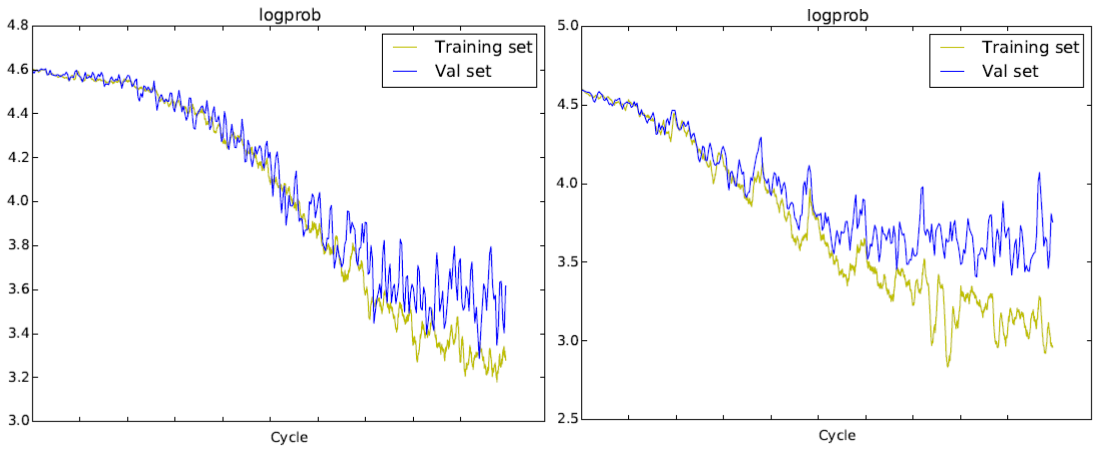


Figure 4.2 Network structure of the model [29]



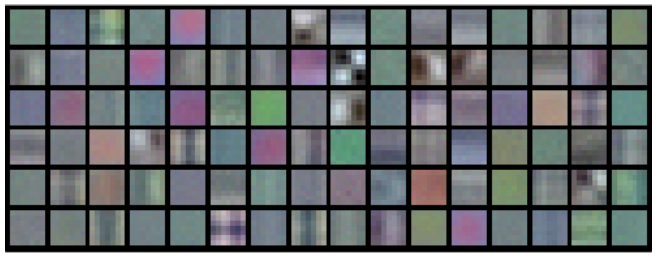
(a) simple images group 1 (b) complex image group 4

Figure 4.3 Learning curves for 2 groups of data

Logprob (min)	Group 1	Group 4
Training	3.17	2.84
Validation	3.28	3.41

Table 4.1 Logprob of 2 groups

“Logprob” is short for log probability. This value is the output from the cost function.


(a) layer 1 filter of group 1

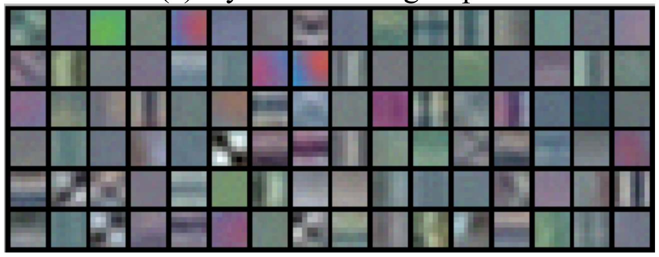

(b) layer 1 filter of group 4

Figure 4.4 Layer 1 filters of 2 groups

Figure 4.4 shows the layer 1 filters of 2 groups. Group 4 has more texture patterns than group 1, because group 4 learns from complex images. The learning curves are in Figure 4.3 and the minimum values of cost functions are in table 4.1. Group 1 with simple images has a smoother learning curve. For this group, the validation curve in the blue line has less over fitting, since it does not detach from the training curve very much. The minimum log probability of the validation curve is 3.28. For group 4 with complex images, both training and validation curves fluctuate a lot. The training and the validation curves detach very quickly. This is a very typical over fitting phenomenon. The parameters learned from the complex training images do not fit for the complex validation images well. This can be explained in another aspect; the complex images within each class are not similar to each other or have big variances. They could share the same foreground objects, but the background varies a lot. Therefore the final feature vectors have large gap. We can reach the conclusion that for CNN, the complex images are more difficult to learn.

In fact, many other image classification algorithms have the same issue. They do not perform well on complex images. We need to find out the reasons from internal steps of CNN that causes this performance.

4.3 Performance drop analysis

Let's review the convolution and pooling steps to see possible issues of the model. The training process is trying to learn the weights and biases through the forward and backward propagations. The filters in the first convolutional layer learn the basic patterns of the local region of image, such as strong colors and high frequency components edges and textures. The feature map is the outputs of convolution and is then passed through an activation function like the ReLU below.

$$f(x) = \max(0, x)$$

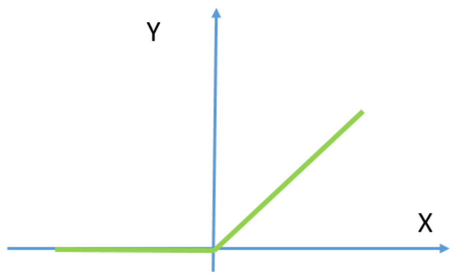
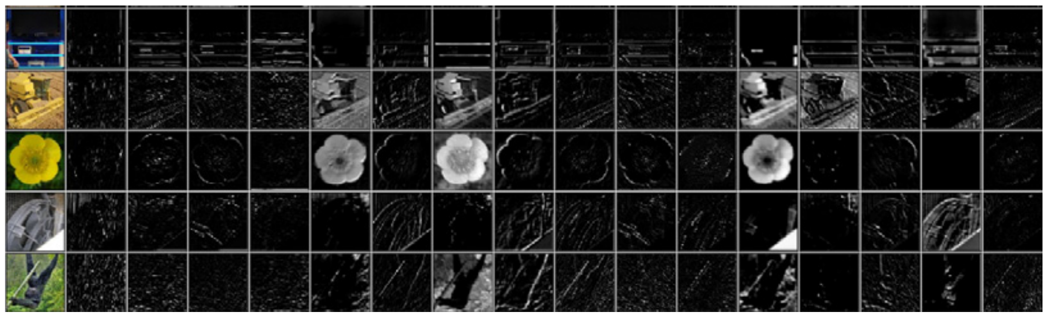
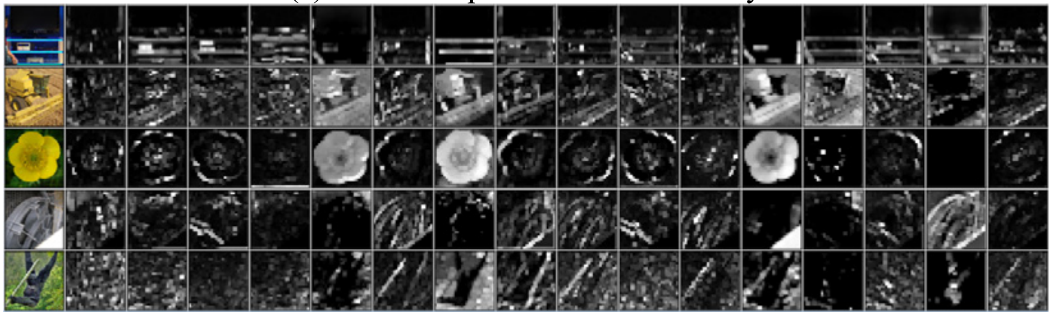
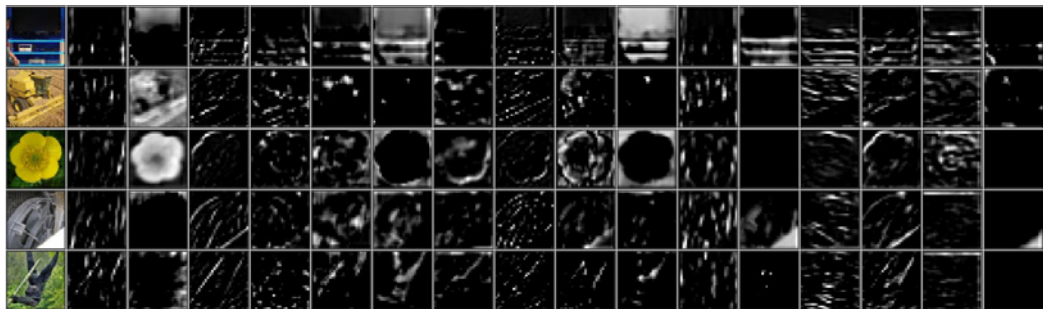


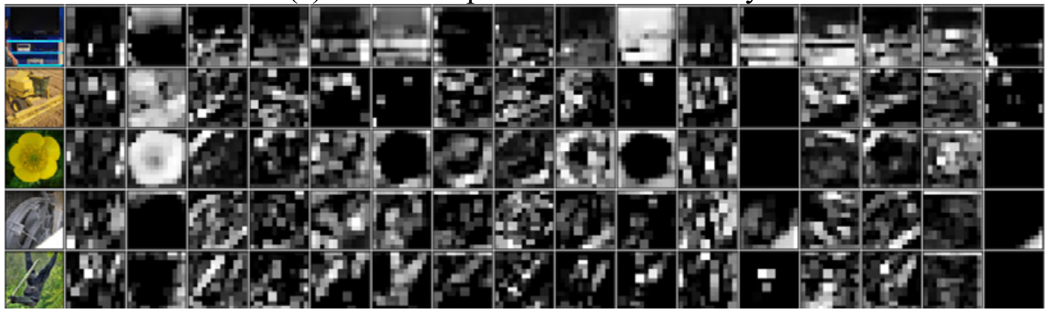
Figure 4.5 The ReLU function

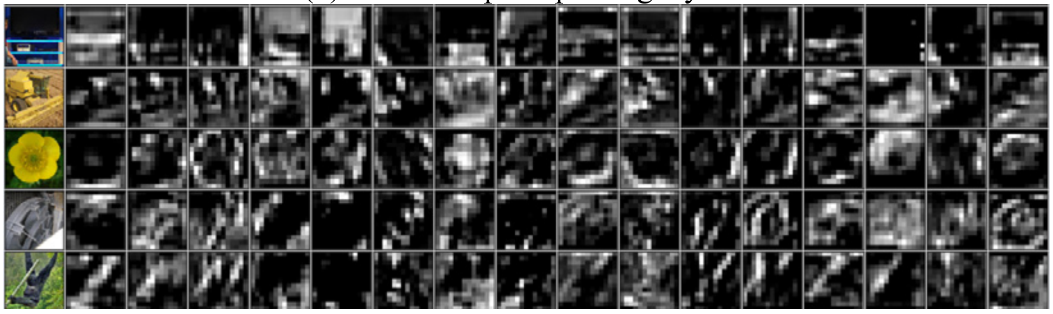
This nonlinearity function maps the negative values and zero to zero and retain all the positive values for forward propagation directly. The derivative for back propagation is a constant one for the input values larger than zero. If the filters have high similarities with the local region of image, the feature map will get a large activation value. Layer 1 filters mainly contains strong colors and edges like filters. The edges filters take a larger proportion as shown in the model [29]. Therefore, the high frequency regions in the source images are more likely to be retained in the feature maps. Pooling is a subsampling process. It chooses the max values of the sub region in the feature map. Pooling enhances the high frequency components are chosen, since the max values are retained from feature maps. After several iterations of convolution and pooling steps, the low frequency components that are related to the plain regions of the source image are dropped gradually. Figure 4.6 shows the feature maps for all intermediate layers in the model [11]. The left column contains the source images and the following images on the right are the random feature maps of the last convolutional layer. By comparing the feature maps and the source images in the last convolutional layer 5 (figure 4.6g), only the high frequency regions, such as the edges and textures, have activation values in the feature maps and all the remaining areas are zeros. The pool layer 3 extracts the max values of convolutional layer 5 and makes sparse feature maps for the high frequency components.

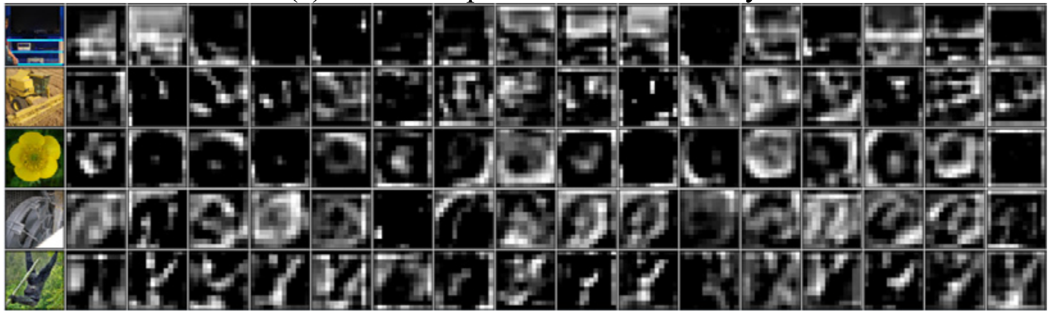

(a) feature maps of convolutional layer 1

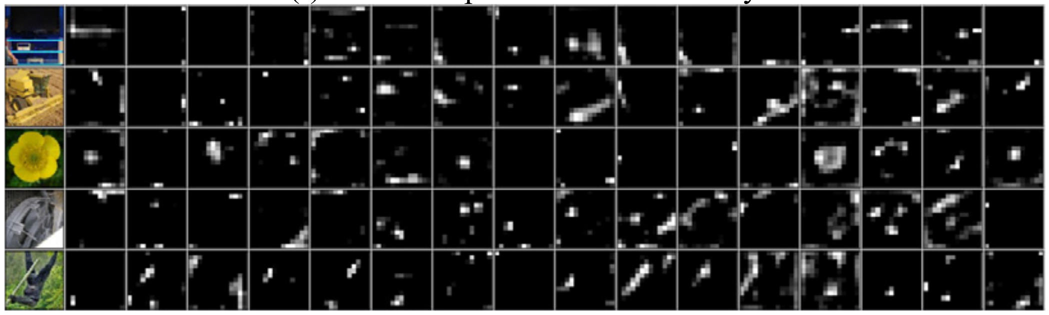

(b) feature maps of pooling layer 1


(c) feature maps of convolutional layer 2


(d) feature maps of pooling layer 2


(e) feature maps of convolutional layer 3


(f) feature maps of convolutional layer 4


(g) feature maps of convolutional layer 5

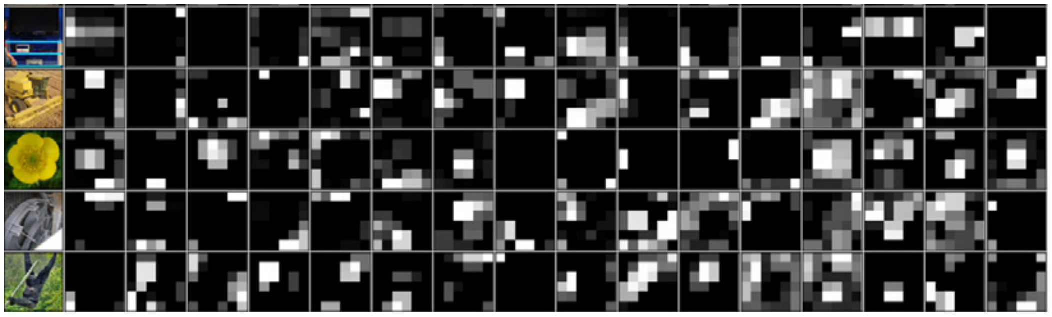

(h) feature maps of pooling layer 3

Figure 4.6 Feature maps of [11]

We need to pay attention to the last row of figure 4.6g. The source image on the left has a much clustered background. As feature maps show on the right, most of the activation values correspond to the high frequency grass background and the borders of the foreground. However, the area inside of the foreground does not have any activation values in any feature maps at all. So the foreground object is almost eliminated in the final feature vector. If the feature vector does not contain or only contains very limited information for the foreground objects, then the classification performance must decrease. The goal of image classification algorithms, including CNN is to extract valid features from images. Nevertheless, based on our observation, CNN are more sensitive to high frequency components of images. As shown in figure 4.6, the feature maps from the first to the last hidden layers gradually drop plain regions, which correspond to low frequency components. The real situation is that not all image objects in the foreground are very complex and textured. Many human made objects usually have simple shapes and less texture. If these simple objects are set in a complex background, they could be gradually eliminated during the convolution and pooling. [20] indirectly proves our conclusion. As shown in this paper, the easiest classes for the classification task are "tiger", "hen-of-wood", and "porcupine". They all have unique shapes with rich textures. The hardest classes are “ladle lope” and “letter opener”. They are very simple objects and lack textures. Once these simple objects are in the complex background, they fail to compete with complex background during convolution and pooling. As a result, the classification performance is bad. Figure 4.7 shows more details on this for the ImageNet dataset.

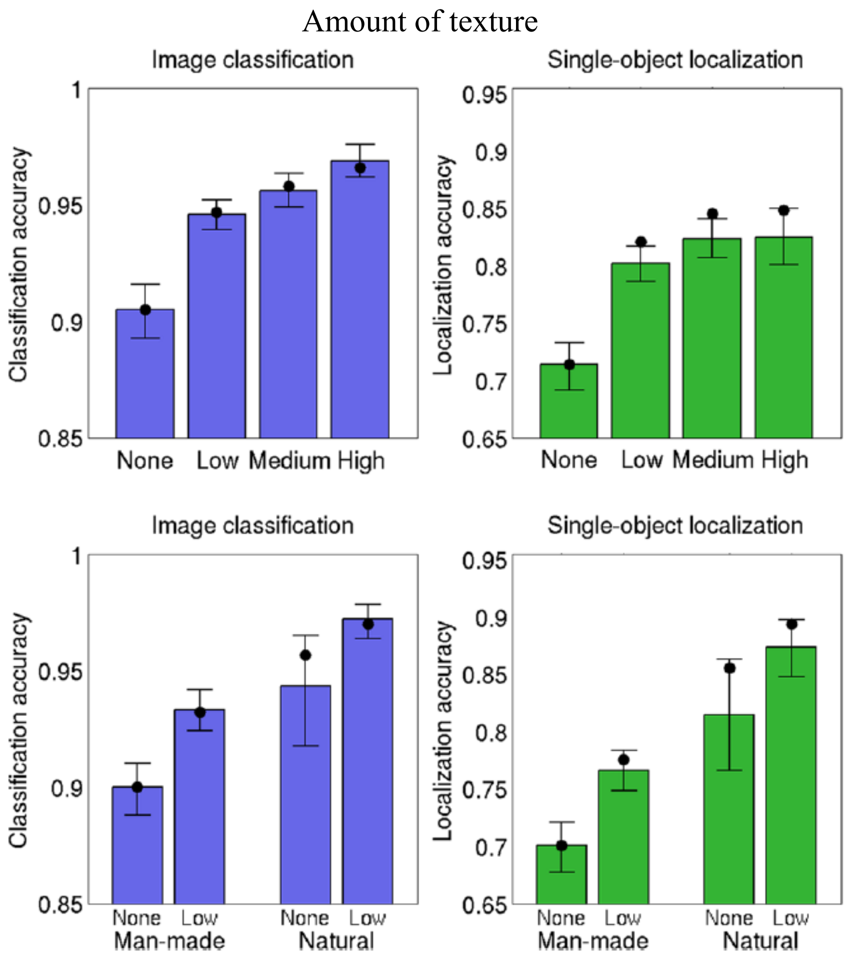


Figure 4.7 Accuracies for different amount of texture [20]

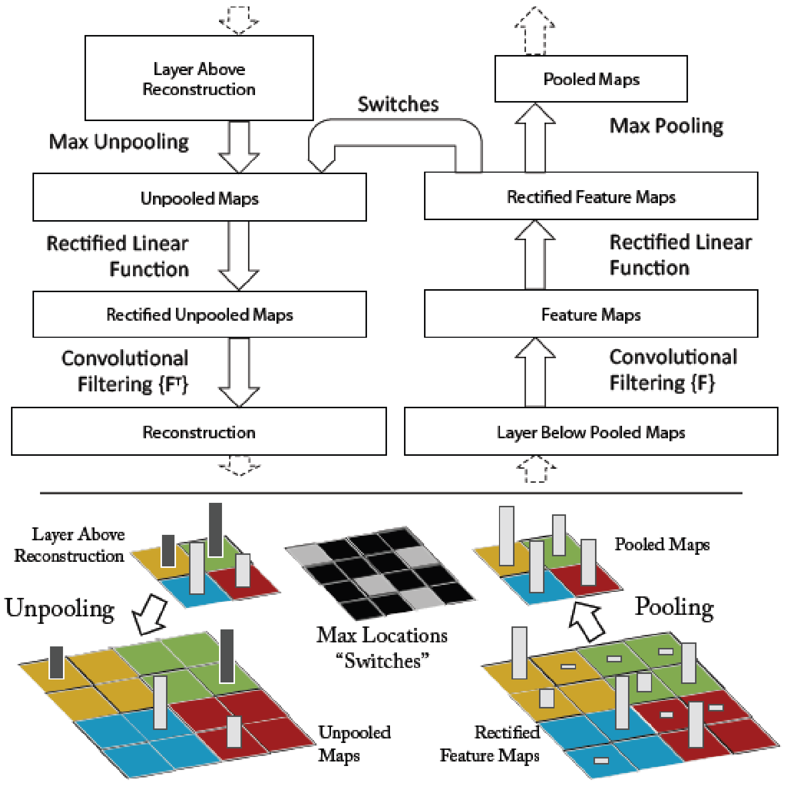
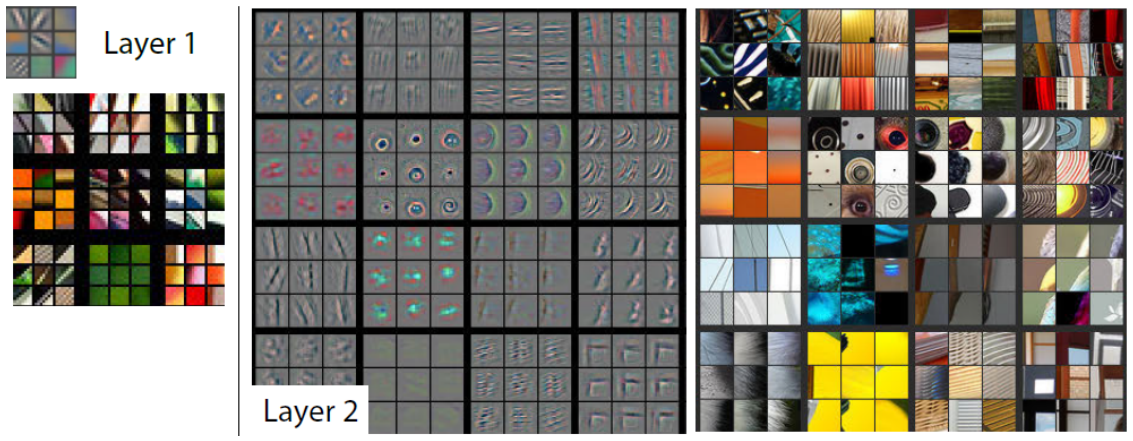
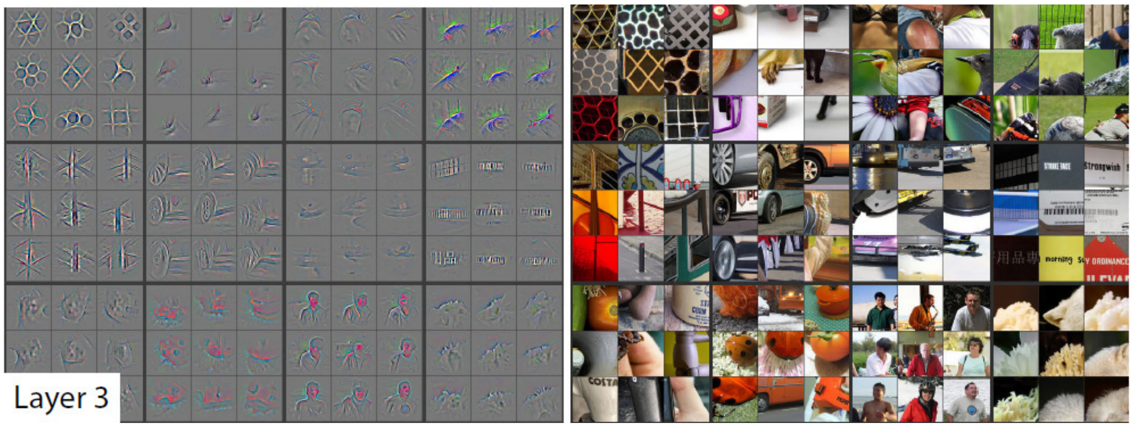


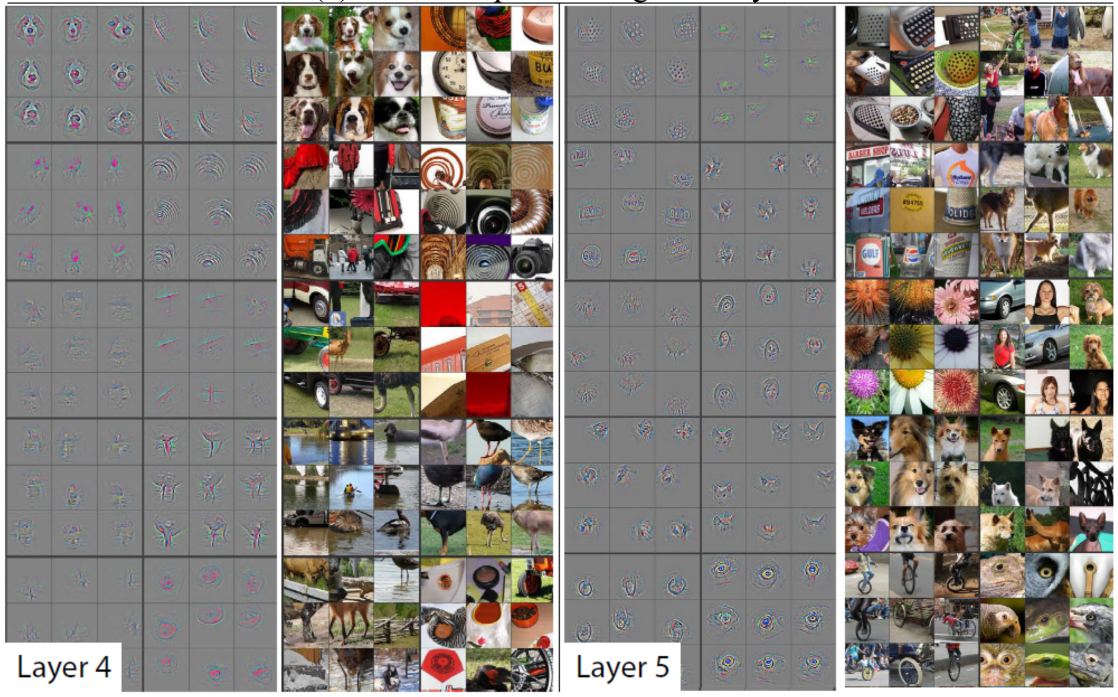
Figure 4.8 Deconvolution for the feature maps [29]



(a) feature maps and images for layer 1 and 2



(b) feature maps and images for layer 3



(c) feature maps and images for layer 4 and 5

Figure 4.9 Visualization of feature maps and corresponding image [29]

The model [29] uses a deconvolution method to visualize feature maps by un-pooling and reverse convolution as shown on the left part of figure 4.8, It finds out the most possible images in the validation set that activate a large number of neurons. We can see from the layer 3-5 the complex images are more likely to activate the neurons. The simple objects only appear in layers 1 and 2. We can conclude that the objects in the different complexities need different levels of abstraction. Complex objects are very likely to override simple ones during convolution and pooling.

4.4 Suppressing high frequency components

We found one important reason why CNN do not performed well on the complex images of each class in section 4.3. The next step is to find a solution to overcome this weakness. Many high frequency regions of images are noises and clustered backgrounds like grass and tree. So we need to suppress the high frequency components to reduce noises. 2D Gaussian filter is a valid tool, but this filter removes the details of images not only from the textured areas but also the edges. For the image classification task, the edges of the foreground objects are critical information. If we apply the Gaussian filter directly, CNN cannot learn the distinguishable features from the blurry images. Hence, we use the bilateral filter introduced in section 2.4 to process the images.

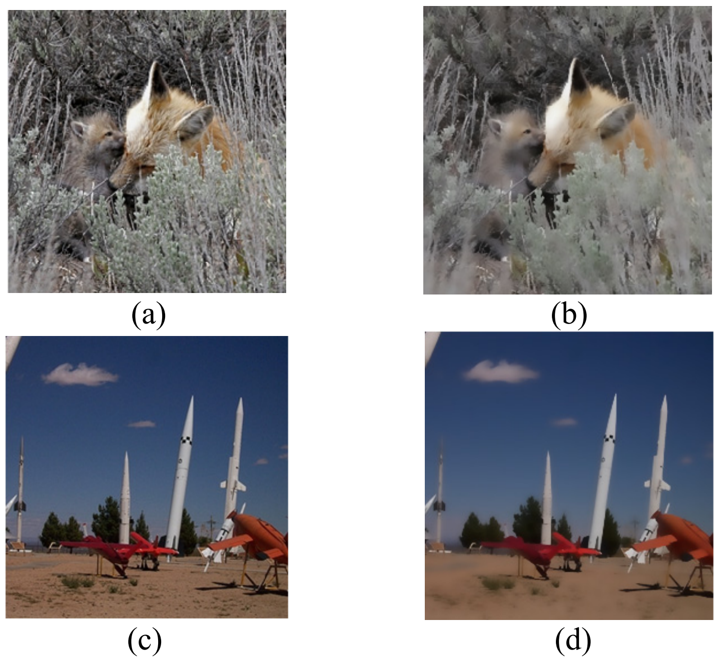

(a) (b)
(c) (d)

Figure 4.10 Examples images of [20] applied bilateral filter

We set the half kernel size to 5, the spatial domain standard deviation to 3 and the radiometric standard deviation to 0.15. These parameters make sure the high frequency components are suppressed but the whole image is not too cartoonlike. The blur filtering process reduces the high frequency signals much faster than the low frequency signals. Figure 4.10 shows examples of the source images and the bilateral filtered images. The foreground “kit fox” (figure 4.10a) has some blurry effect after filtering (figure 4.10b), but the overall shape and the contours are still clear. On the contrary, the background high frequency details reduce a lot. The background grass of the “missile” (figure 4.10c) has blurry effects after filtering (figure 4.10d), but the foreground missiles have no change. Suppressing the high frequency components of the source images makes the final feature vector contains less cluster background, so the proportion of the foreground appearing in the feature vector is increased. This means more valid features are extracted from the images.

Chapter 5 Multi-path Convolutional Neural Network

5.1 Networks design and implementation

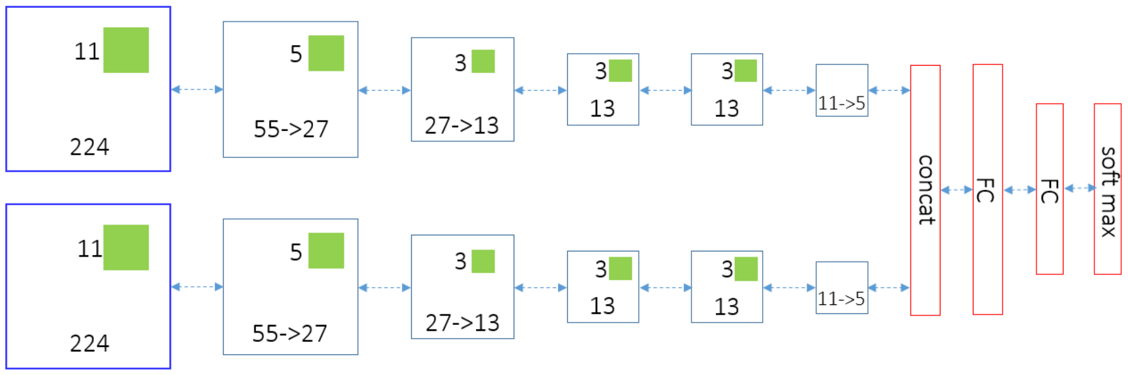


Figure 5.1 Multi-path CNN

We define a multi-path CNN in figure 5.1. It extracts features from both source images and bilateral filtered images. Figure 5.2 shows how the networks run forward and backward propagations. The first path on the top feeds the source image and the second path feeds the bilateral filtered images. This method can learn more comprehensive features than a single path model. The multi-path model constructs a longer feature vector based on the concat layer. It contains features from both paths that better represent the images. If the foreground object is simple and the background is complex, then the foreground object is more likely retained in the second path, which feeds the bilateral filtered image. In contrast, a complex foreground object like "tiger" is more likely retained in the first path, which is dominated by the high frequency components. The multi-path model can capture a broader range of foreground objects from simple to complex. Thus, it is more robust than a single path model.

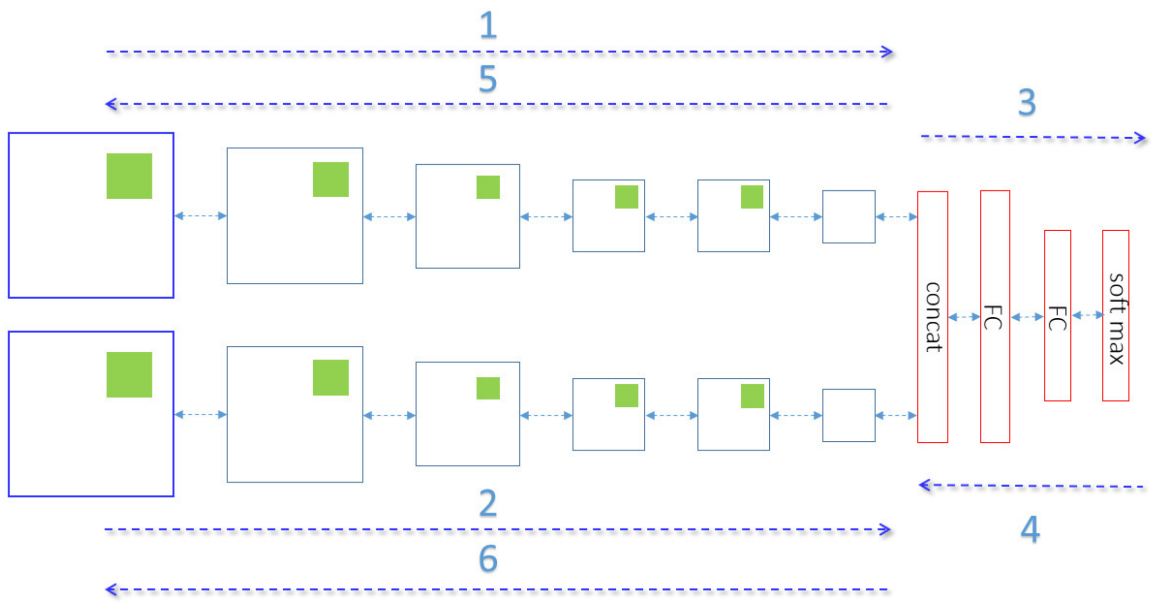


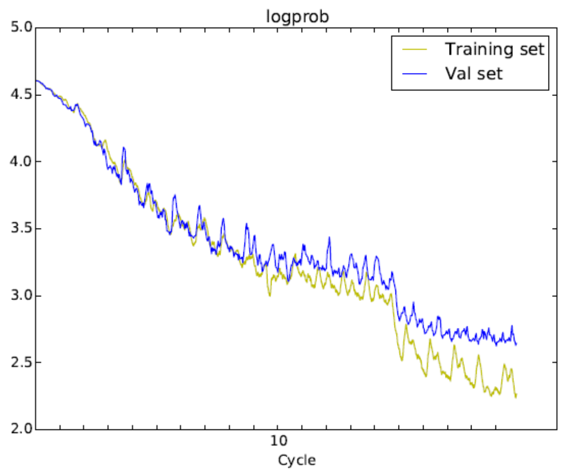
Figure 5.2 Forward and backward propagations

The detailed network design is as follows: The image layer provides 224×224×3 RGB images. From layer 1 to 5, cross feature maps normalization is applied after convolution. The first convolution layer applies 11×11 filters with the image layer and the produces 55×55 feature maps. The Pooling layer produces 27×27 feature maps from normalized feature maps. The second convolution layer applies 5×5 filters with 2 pixels border padding for the input feature maps. So it produces 27×27 feature maps. The following pooling layer produces 13×13 features maps. Layers 3 and 4 use 1 pixel border padding for the input feature maps before convolution. These 2 layers do not use the pooling layers anymore. Layer 5 applies a 3×3 filter to normalized feature maps from layer 4. The last pooling layers for layer 5 produce 5×5 feature maps. The numbers of filters from layer 1 to 5 are 48, 192, 256, 256 and 192. So the length of the concat layer is 9,600 that combines the output of pooling layers from 2 paths, with each path contributing 4,800 (192×5×5) neurons. This feature vector is longer than the traditional single path model, like 4096 neurons in [29]. The following 2 layers are full connection layers. The first full connection layer has the same length as the concat layer and the length of the second one is 100. We leverage dropout layer for 2 full connection layers for the input connections. The dropout rate is 0.5. The output of the second full connection layer is connected to a 100 way softmax regression layer which produces a distribution for 100 classes. We set the initial values of weights for all layers through a zero mean normal distribution with a standard deviation 0.01. The initial values of biases with constant zeros. The L_2 weight decay coefficient is set to 0.0005. Momentum is set to 0.9. The model maximizes a softmax regression function, which maximize the average log-probability of the image labels over prediction distribution for all the training examples.

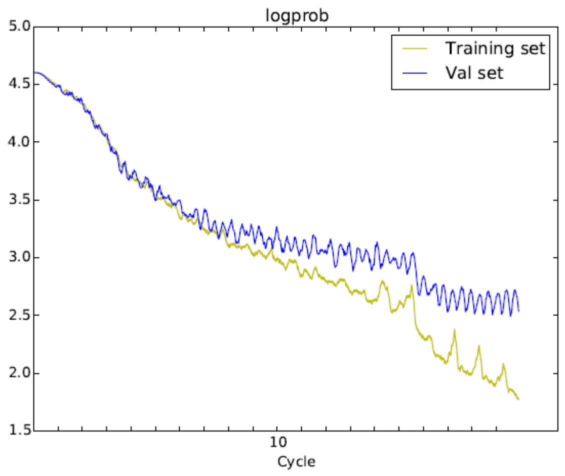
We use inherit a public CNN code library from model [11] to implement our model. This public version only supports single path with one data layer as input. We make the extension for multi-path with a more flexible data structure for input layer and designate how the networks run the forward and backward propagations. We implement dropout as well, which is used for the full connection layers to prevent overfitting. We use the Matlab to process the image data, including image transform and filtering. The mini data batch files of both training and validation sets are created by using the Matlab.

5.2 Experiment 1 Training the complex images

We feed our model with the group 4 dataset created in section 4.1, which has the most complex images for training and testing. This is a more challenging group than the other 3 groups. There are total 325 mini batch files in the training set and 12 mini batch files in the validation set. Our model has 2 paths, then each mini batch in the training and validation sets contain 200 images with the first half (1-100) images are in the source version and the second half images are in the bilateral filtered version. They are paired together and each image pair has the same class label. The total class number of each mini batch is still 100. The image sequence in each mini batch is randomly permuted. We use the data augment method as described in section 3. The validation frequency is set to 10. Our code runs on a single NVIDIA 4GB GTX 760 GPU for 20 cycles. Training a mini batch needs 3.15 seconds on average. We reduce the learning rate by multiplying 0.1 after the error rate on the validation curve goes flat.



(a) model [29]

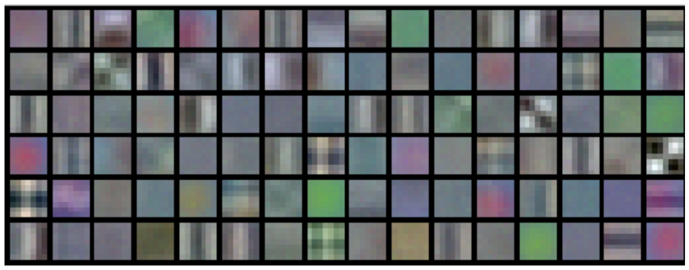


(b) our model

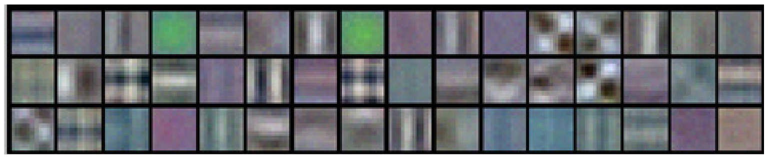
Figure 5.3 Learning curves for the complex dataset

Logprob (min)	Model [29]	Our model
Training	2.24	**1.77**
Validation	2.65	**2.40**

Table 5.1 Logprob for complex dataset



(a) filters of layer 1 of Model [29]


(b) path 1 filters of our model

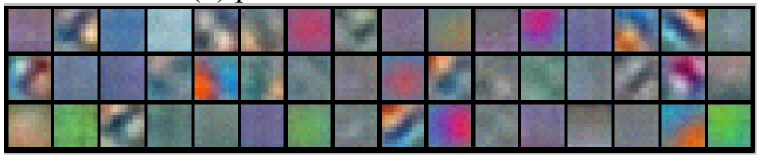

(c) path 2 filters of our model

Figure 5.4 Path 1 filters of 2 models on complex dataset

Model	Top 1	Top 5
Model [29]	66.5%	35.6%
Our model	64.2%	33.5%

Table 5.2 Top1 and Top5 error rates for complex dataset

To compare models, we run the group 4 dataset on Model [29] for 20 cycles as well. Each mini batch file in the training and validation sets for the single path model has 100 images in the source version with 100 labels. Training a mini batch for the single path model [29] needs 2.75 seconds on average. Both models use 1,200 center crop 224×224 validation images to report the scores. Figure 5.3 shows the learning curves for 2 models. Our model has smaller log probability values for both training and validation curves in table 5.1. The scores for 2 models are in table 5.2. We win in both top-1 and top-5 scores. Figure 5.4b and 5.4c show the filters of layer 1 of our model for each path. The 48 filters in the first path show very clear patterns, such as the vertical and horizontal edges and textures. Most of the filters in figure 5.4b contain high frequency components. The 48 filters in the second path show strong color spots but less edges, since the images in the second path are bilateral filtered, which reduce the high frequency components. The total

96 filters in 2 paths learn different aspects of the images. They are richer than the 96 filters learned only from traditional single path in figure 4.5a of Model [29]. Furthermore, in our model, the filters between 2 paths have no correlations, but the filters within each path show more correlations and similarities. Figure 5.4a and 5.4b have similarities, because they both learn from the source images.

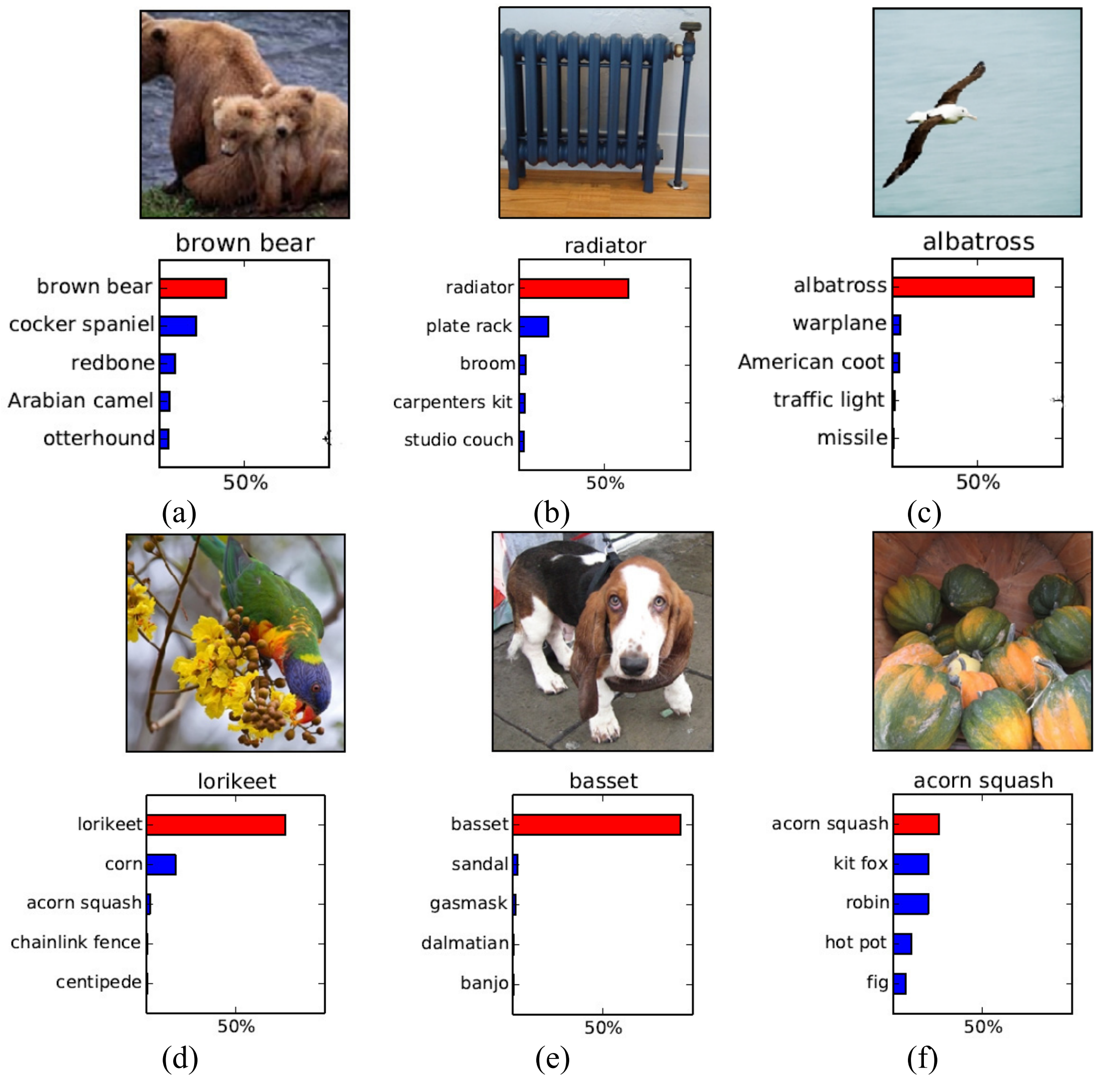


Figure 5.5 Validation images hit top1 accuracy

Figure 5.5 shows 6 predictions of the validation images of our model. They all hit top 1 accuracy. Figure 5.5a is “brown bear”, which has a similar fur pattern and color with "cocker spaniel" and "redbone". Figure 5.5b is "radiator", which is a human made object and has a simple shape with less texture. The second probable label "plate rack" is a simple human made object as well. Figure 5.5d is "lorikeet". The second probable label of this image is "corn", because of the presence of the yellow flowers, which has similarities with the "corn" class.

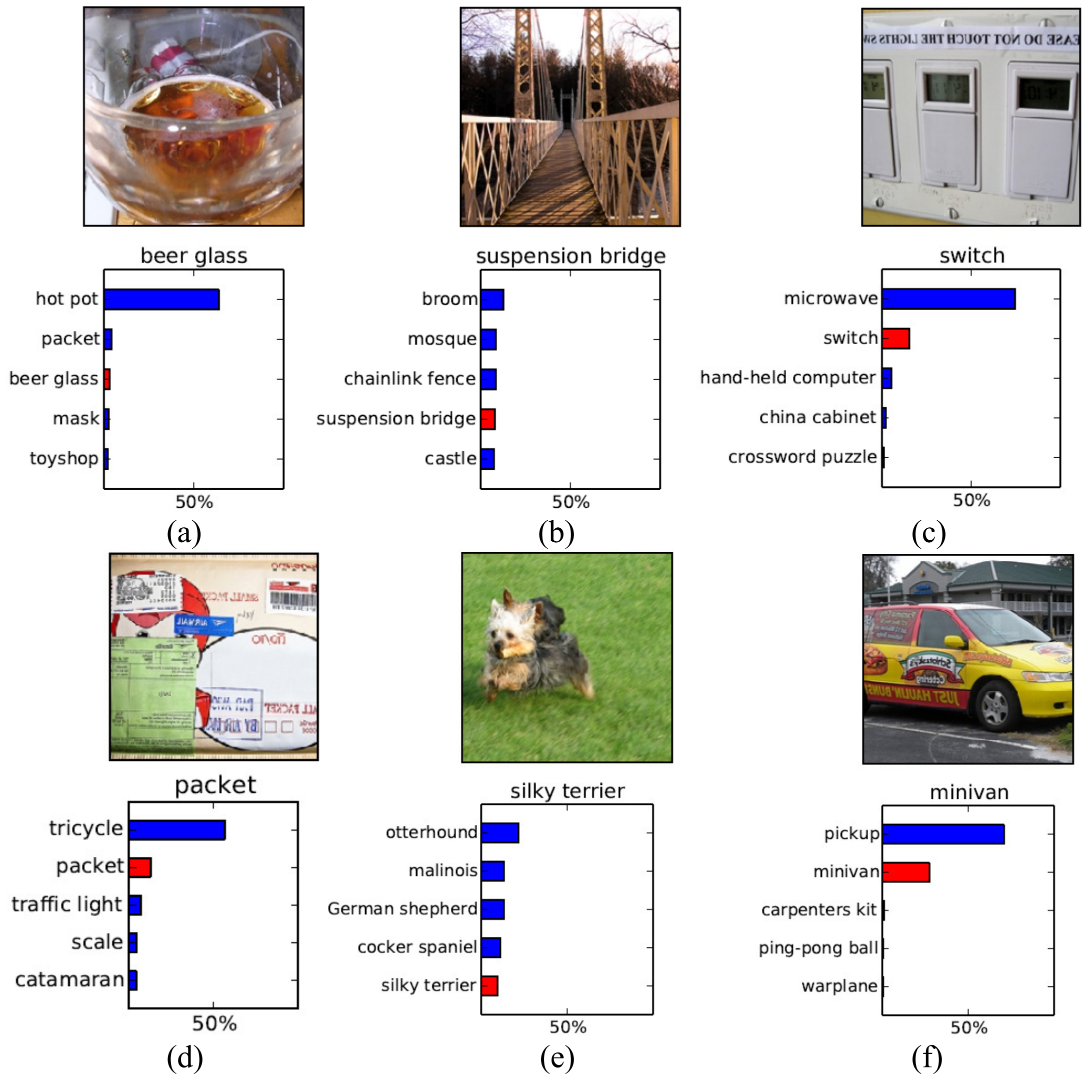


Figure 5.6 Validation images hit within top5

Figure 5.6 shows 6 validation images hit within top 5. Figure 5.6c is "switch", which is a human made object with simple shape and less texture. The contour of "microwave oven" looks like "switch" taken from very near the camera. Figure 5.6e is "silky terrier", which has a complex grassland background. The top 5 prediction labels for this image are all dogs, such as "otter hound", "malinois", "german shepherd", "cocker spaniel" and "silky terrier". These 5 classes share high similarities between each other. Thus the prediction probable values have no big differences in blue bars.

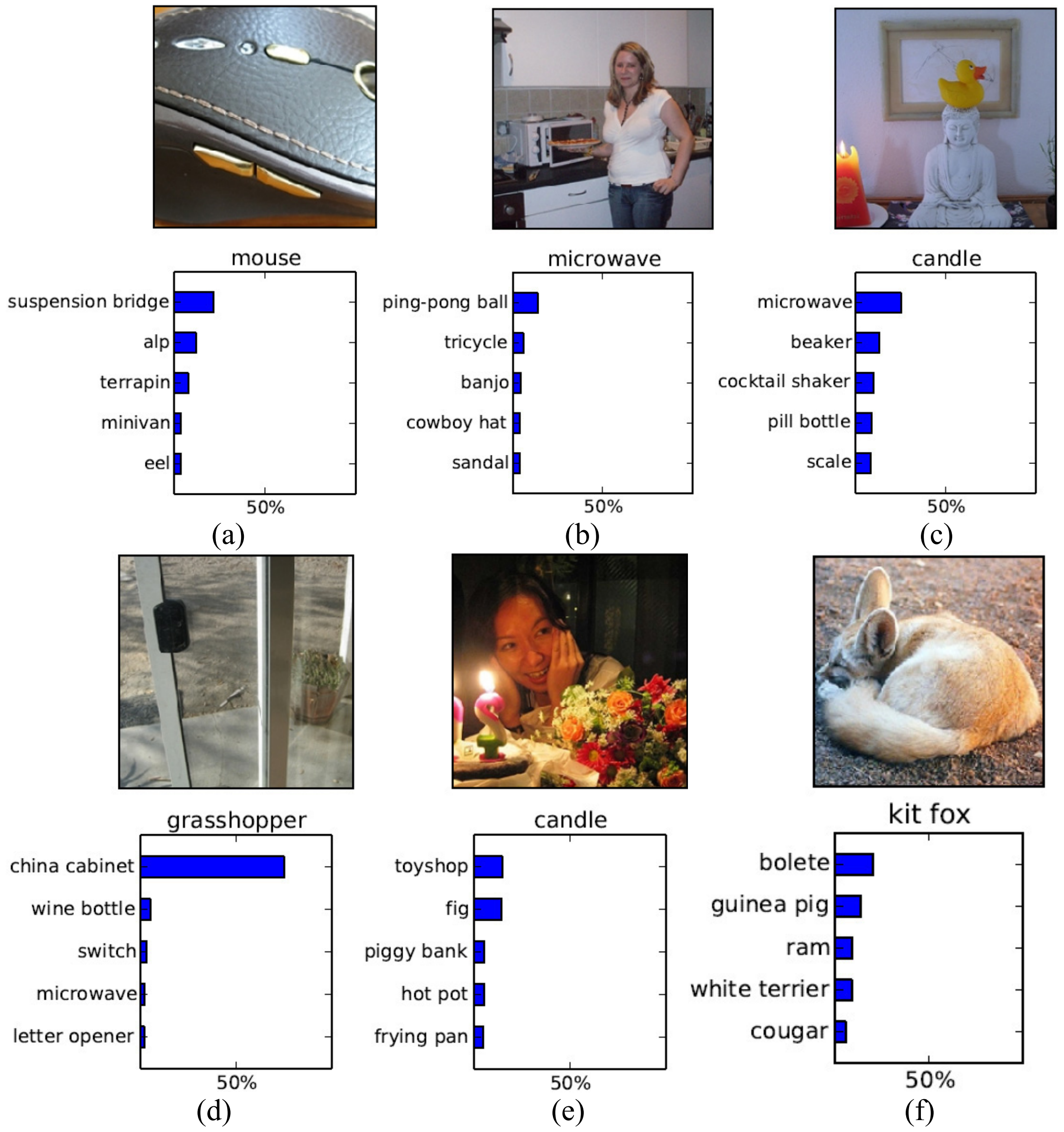


Figure 5.7 Validation images with wrong labels with in top 5

Figure 5.7 shows 6 validation images with the wrong labels within the top 5 predictions. Figure 5.7a is “mouse”, which is very near the camera. Figure 5.7b is “microwave”, which takes very small area of the whole image. Figure 5.7c-e are “candle” and “grasshopper”. They have the same issue that foreground objects take a very small area of the whole images. The other validation examples like these are less likely get right predictions. Figure 5.7c is “kit fox”, which has a special pose and the center crop means the head of the fox is not included in the image layer. Thus, it does not get the right label.

5.3 Experiment 2 Model generalization

We do a further test to see the model generalization ability. Our model uses group 4 with complex images for training. Now we test the model by using the simplest validation images of group 1. We apply this test method to the model [29] to compare the performance. They both use 1,200 center crop 224×224 validation images of group 1 with simple images to report the scores. The results are listed in table 5.3. We win in both top 1 and top 5 scores again. Top 1 result has a 4.4% improvement.

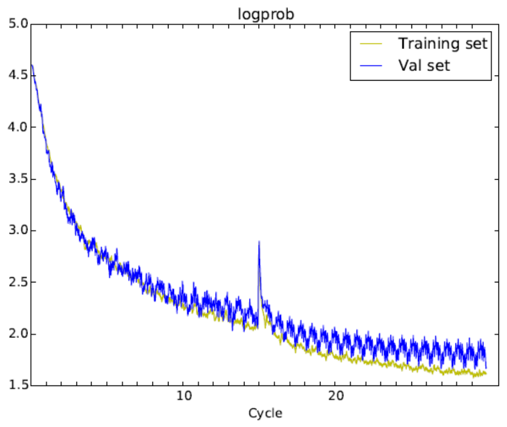
Model	Top 1	Top 5
Model [29]	67.6%	36.6%
Our model	63.2%	34.9%

Table 5.3 Top1 and Top5 error rates for simple images

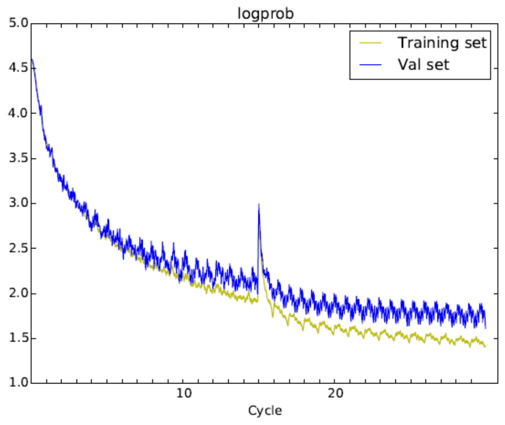
5.4 Experiment 3 Training all images of 100 classes

The previous 2 experiments only use one quarter of the training images of 100 classes. Now we use the full training and validation images of 100 classes. Each class has 1,300 images for training and 50 for testing. We still adopt the data augmentation method in the section 4.1. The data structure of mini batch file for our 2 paths model is same as in the experiment 1. There are total 1,300 mini batches for training and 50 mini batches for validation. All training mini data batch files run for 30 cycles. Then the total training examples are up to 7,800,000 (1,300×100×30×2) for our 2 paths model.

To compare models, we run the full 100 class datasets on Model [29] for 30 cycles as well. The mini data batch file in training and validation sets contains 100 images with 100 different classes for the single path model. Then the total training examples are up to 3,900,000 (1,300×100×30) for the single path model. Both models still run on the single NVIDIA 4GB GTX 760 GPU independently. We reduce the learning rate from 0.01 to 0.001 at the 15th cycle during training. Both models use 5,000 center crop 224×224 regions of validation images for testing. We can see the peak values of learning curves in figure5.8. They should be caused by the momentum dominates the parameter updates after training restart at 15th cycle. Our program do not save the update (delta) of each parameter into the disk, but save the current values (one version) of the parameters only. Parameter updates (delta) are only saved in the memory during the program running. This is for performance purpose, as we have a huge amount of parameters. After training restart and the program does not know the previous updates, then we hack the code for the previous updates as a constant number 1. The momentum is 0.9, which is larger than the learning rate 0.001. Therefore, the momentum term (0.9×1) dominates the current update, and then the cost function over shoot a large value in the high dimensional non-smooth surface. This phenomenon does not always happen. The peak value does not impact the final result, as the parameter updates (delta) get the true values once train the second mini batch and the value of the cost function reduces very quickly.



(a) model [29]

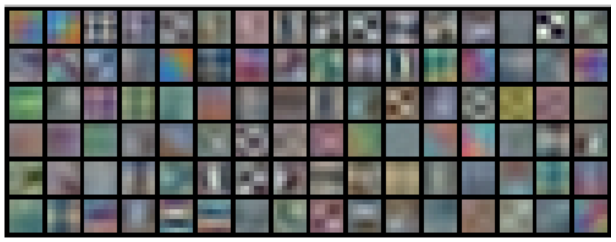


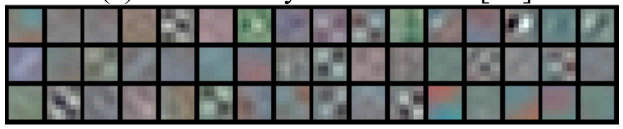
(b) our model

Figure 5.8 Learning curves on the full dataset

Logprob (min)	Model [29]	Our model
Training	1.59	**1.33**
Validation	1.66	**1.60**

Table 5.4 Logprob of 2 models


(a) filters of layer 1 of model [29]


(b) path 1 filters of our model

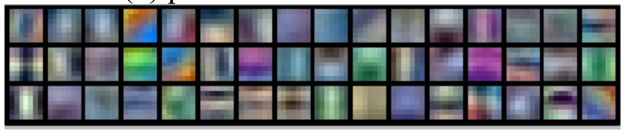

(c) path 2 filters of our model

Figure 5.9 Layer 1 filters of 2 models

Model	Top 1	Top 5
Model [29]	48.3%	20.51%
Our model	46.6%	19.03%

Table 5.5 Top1 and Top5 error rates for full dataset

Figure 5.8 shows the learning curves for 2 models and table 5.4 shows the minimum values of the learning curves for both training and validation sets. Our model hit the smaller values on both training and validation curves. We win both top 1 and top 5 scores again as shown in table 5.5. As the scores are obtained after training for 30 cycles on both models, then error rate is larger than the model [29] on 1,000 classes for 70 cycles. It is fair we train both models for 30 cycles to compare the performance and the validation curves in both models already become flat. So the base line is 30 cycles. Figure 5.9 shows the filters of layer 1 of 2 models. The filters in path 2 of our model have very smooth edges and are without very clear texture pattern, since we applied bilateral filters to the input images.

5.5 Experiment 4 Training single path with bilateral filter images

We want to prove that using multi-path and different versions of the images can get better distinguishable features and better prediction results. In the experiment of section 5.4, we use a single path model to train the full dataset and the result is not better than our model. How about we use a single path model to train the bilateral filtered images on the full dataset? We use this method to train and test the bilateral filtered images in a single path model as section 5.4 for the original version of images.

Figure 5.10 below shows the 96 filters learned from the bilateral filtered images. Compared with the experiment in section 5.4 of the single path model, which feeds source images, the filters in the current experiment show less texture patterns and have smooth edges only. We need to see the performance impact of training bilateral filtered images. Figure 5.11 shows the learning curves and table 5.6 shows the minimum value of the training and validation learning curves. We find the values in table 5.6 are larger than the values in section 5.4. The results become worse. Therefore, we can conclude that using single path and bilateral filtered images is not a good choice. This could be caused by high frequency components being lost. Some complex objects with rich texture become blurry after filtering and are not easy to distinguish from each other.

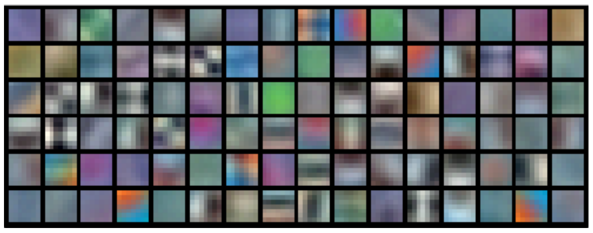


Figure 5.10 Filters of bilateral filtered images

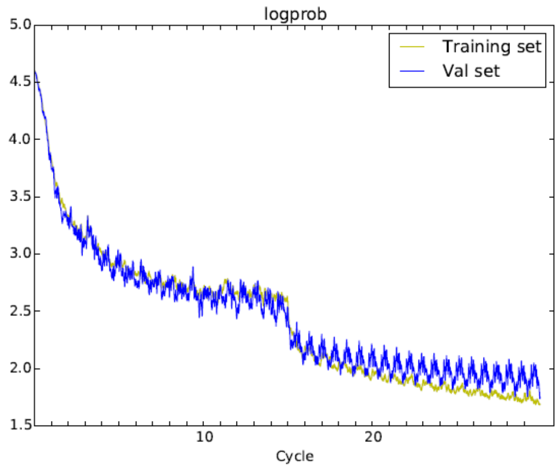


Figure 5.11 Learning curve for bilateral filtered images

Logprob (min)	Model [29]
Training	1.70
Validation	1.73

Table 5.6 Logprob of single path model for bilateral filtered images

5.6 Experiment 5 Training source and Gaussian filtered images

To prove that we apply the bilateral filter to the images in the second path is a right choice, we run another test by applying the Gaussian filter to the images in the second path. The first path still uses the source images. We use a 5×5 Gaussian filter with zero mean and standard deviation is equal to 1. All the training and validation images in the second path are applied with this filter. We still use 100 classes full dataset. We still run the 2-path model for 30 cycles and adjust the learning rates from 0.01 to 0.001 at the 15th cycle.

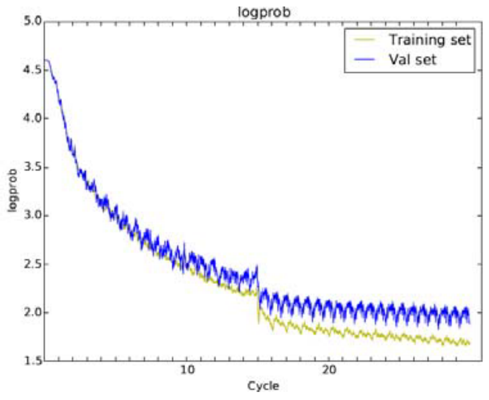
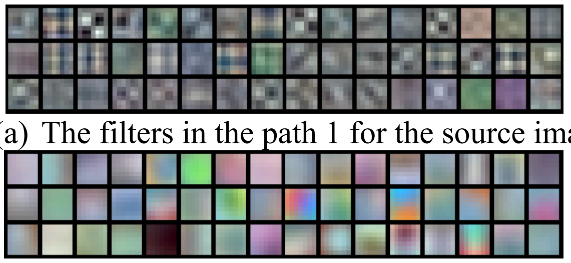


Figure 5.12 Learning curves for the source and Gaussian filtered images

Logprob (min)	Our model
Training	1.64
Validation	1.82

Table 5.7 Logprob of 2-path model for the source and Gaussian filtered images

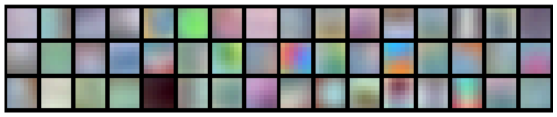


(a) The filters in the path 1 for the source images

(b) The filters in the path 2 for the Gaussian filtered images

Figure 5.13 Layer 1 filters for 2-path

As shown in the figure 5.13b, the learned filters in the second path from the Gaussian filtered images are very blurry. By contrast, the learned filters in the second path from the bilateral filtered images in figure 5.9c are very distinguishable. For the comparison purpose, we show them again in the same figure 5.14.


(a) Copy of figure 5.13b

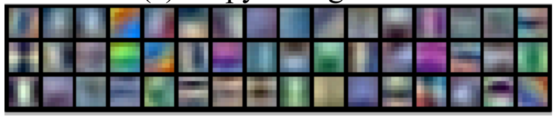

(b) Copy of figure 5.9c

Figure 5.14 Comparing filters in 2 experiments

Figure 5.12 shows the learning curve for the current experiment. Table 5.7 shows the minimum values in the learning curves. The result 1.82 in the validation curve is worse than the result 1.60 in the validation curve of 2-path model, which trains the source and bilateral filtered images. It is even worse than the result 1.73 in the validation curve of the single path model, which trains bilateral filtered images only. This experiment get the worst results among all experiments, which use 100 classes full dataset. Therefore, we can conclude that Gaussian filter is not a proper choice. It removes too much details from the images.

Chapter 6 General discussion and future work

6.1 Conclusion

In this thesis, we conducted an advanced research of CNN with image analysis and find out the reasons of performance drop on complex images. To resolve the issue, we leverage image processing technology to suppress high frequency components but keep the overall object shape and contour. We design and implement multi-path networks that extract more distinguishable features from both source images and bilateral filtered images than the single path model that feeds the source images only. We use 4 different experiments to show our method has better result for predictions. Therefore, we can conclude that multi-path CNN show more advantages than the simple path model for the image classification task.

6.2 Limitation

Our 2 paths model needs larger GPU memory than the single path model. The memory size of the low or middle end GPU is smaller than the host (CPU) memory. GPU needs to load the input images, parameters, and feature maps in hidden layers for training and testing. Take the full connection layer as an example, the number of the parameters for the first full connection layer in our model is 92,160,000 (9,600×9,600), since the previous concat layer has 9,600 neurons as well. All the above data are in the float precision. The really memory consumption is larger than the size of GPU memory. Therefore, the data need to move between the GPU and host memories. The host memory play a role as a temporary storage for GPU. Figure 6.1 shows the memory fluctuation in GPU. The data allocations in both GPU and host memories need time and then impact the training speed.

Furthermore, 2 paths model has more complex forward and backward propagations than the single path model. Thus, the training time is 14.5% = (3.15-2.75)/2.75 longer than the single path model.

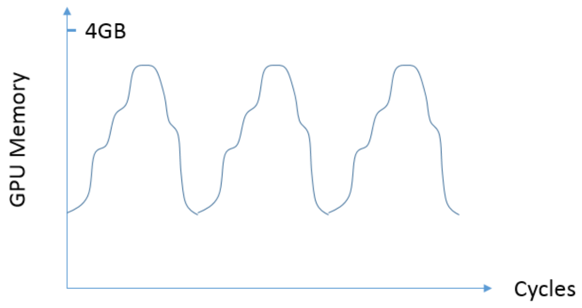


Figure 6.1 Illustration of memory fluctuation in GPU

In the chapter 5, we use 5 experiments to prove that our model can extract more distinguishable features from the source and bilateral filtered images. Our model has better results for the predictions on the validation set. However, there are still some images cannot be hit within top 5 predictions. In other words, using bilateral filter cannot resolve all the issues. After observing the images with the wrong predictions, we find some common properties of these images. Some objects are too small, such as "grasshopper" and "candle"; some objects are covered by other objects, such as "microwave" and some objects are in very special pose, such as "kit fox". All these conditions impact the results.

6.3 Future work

To take our work further we can, for example, add more paths and different paths using different depths and parameter settings (figure 6.2). We suppose simple and complex objects need different levels of abstraction. Simple objects should use less convolution and pooling steps. We could also try multiple GPU configurations (figure 6.3), which let different paths run different GPUs to speed up the training procedure.

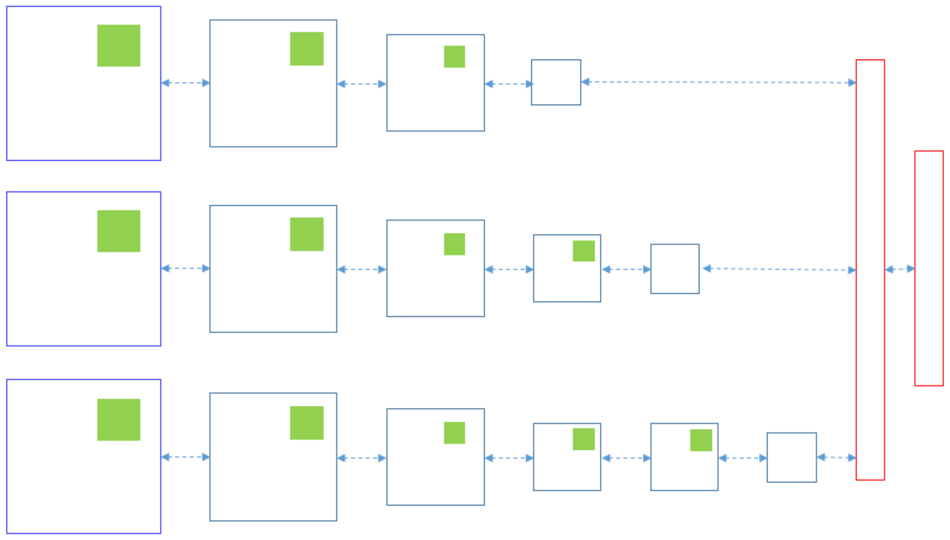


Figure 6.2 Multi-path model with different depths

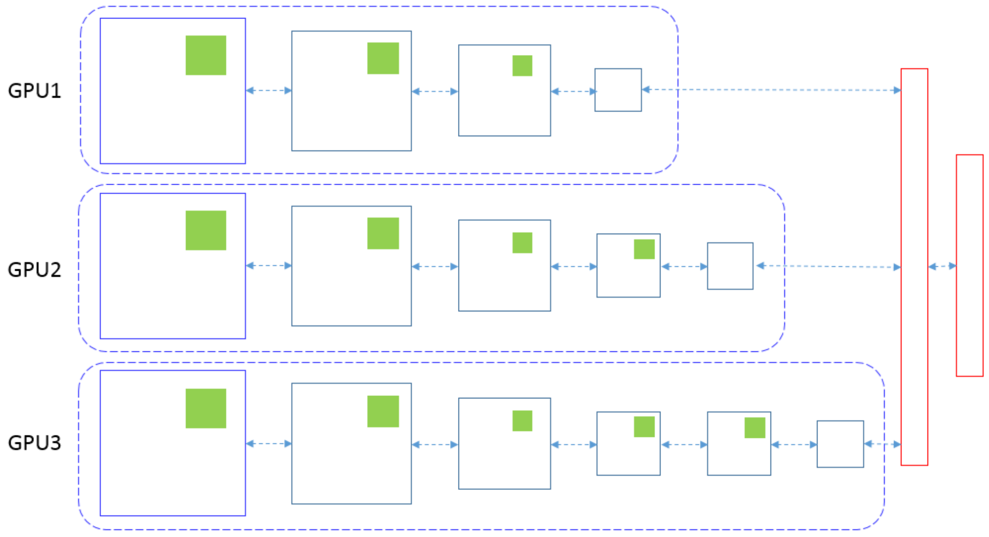


Figure 6.3 Multi-path model on different GPU

Appendix A: the Network and Parameter Configurations

The network and parameter configurations are list below. Layer name is in parenthesis, such as [conv1.1] represents the first convolutional layer of the first path. The number before the dot is the path index and the number after the dot is the layer index. The key words of the parameters are list in the table 7.1. The default values of the parameters and the network structure are list in the table 7.2. For CNN, the default values of the numerical parameters are based on experience. We adopt some default values for the numerical parameter from the model [29]. We add a mark (Y) under the key words of numerical parameters, which are inherited from the model [29]. We also adopt learning rates at 0.01 and 0.0001, L_2 weights decay rate at 0.0005 and Momentum at 0.9 from the model [11]. All the rest parameter values and the network structure are fixed after we repeated tries. This is more like an engineering job that try to find a workable model.

Names	**Descriptions**
type:	The type of current layer, such as convolutional layer or pooling layer.
data:	Data layer for input images and class labels.
dataIdx:	Data matrix index in mini batch file.
conv:	Convolutional layer.
pool:	Pooling layer. “max” is for the max pooling.
fc:	Fully connected layer
dropout:	Dropout layer

eltconcat:	Element wise concatenation. Just combine input data as one feature vector.
softmax	Softmax layer for classification. It needs the output from the network and class labels.
inputs:	The name of previous layer.
channels:	The number of channels, such as the image with 3 channels then set to 3.
filter:	The number of filter in the current layer.
filterSize:	The size of filter. If this value is set to 7, then the filter is 7×7 square.
padding: (Y)	The number of pixels pad to the border of images or feature maps.
stride: (Y)	The number of pixels shifted for the filter during convolution.
initW: (Y)	The standard deviation of the Gaussian function for weight parameters to random initialize. In general this value is set to 0.01. The mean value is 0 and it is hard coded.
initB: (Y)	The initial value of bias term. It is always set to 0.
neuron	The type of neuron, such as sigmoid and Relu.
sizeX: (Y)	The size of local region for pooling. If the value is 3, then pooling choose max value from 3×3 region from the current feature map.
cmrnorm:	Cross feature map normalization. This is to make sure the value of the input images and feature maps are in the same range, such as between 0 and 255.
sharedBiases:	The Boolean value to set if each filters in the current layer share bias term. 1 is for true.

Table A.1 key words for the network parameter

```
# data layer
[data1] # source image
type=data
dataIdx=0

[data2] # bilateral filtered image
type=data
dataIdx=1

[labels] # class label
type=data
dataIdx=2
```

```
# Path 1
#------------first layer--------------
 [conv1.1]
 type=conv
 inputs=data1
 channels=3
 filters=48
 padding=0
 stride=2
 filterSize=7
 initW=0.01
 initB=0
 sharedBiases=1
 neuron=relu

 [rnorm1.1]
 type=cmrnorm
 inputs=conv1.1
 channels=48
 size=5

 [pool1.1]
 type=pool
 pool=max
 inputs=rnorm1.1
 sizeX=3
 stride=2
 channels=48
 neuron=relu

 #------------second layer--------------
 [conv1.2]
 type=conv
 inputs=pool1.1
```

```
# Path 2
#------------first layer--------------
 [conv2.1]
 type=conv
 inputs=data2
 channels=3
 filters=48
 padding=0
 stride=2
 filterSize=7
 initW=0.01
 initB=0
 sharedBiases=1
 neuron=relu

 [rnorm2.1]
 type=cmrnorm
 inputs=conv2.1
 channels=48
 size=5

 [pool2.1]
 type=pool
 pool=max
 inputs=rnorm2.1
 sizeX=3
 stride=2
 channels=48
 neuron=relu

 #------------second layer--------------
 [conv2.2]
 type=conv
 inputs=pool2.1
```

```
channels=48
filters=192
padding=0
stride=2
filterSize=5
neuron=relu
initW=0.01
initB=0
sharedBiases=1

[rnorm1.2]
type=cmrnorm
inputs=conv1.2
channels=192
size=5

[pool1.2]
type=pool
pool=max
inputs=rnorm1.2
sizeX=3
stride=2
channels=192
neuron=relu

#------------third layer--------------
[conv1.3]
type=conv
inputs=pool1.2
filters=256
padding=1
stride=1
filterSize=3
channels=192
initW=0.01
initB=0
sharedBiases=1
neuron=relu

[rnorm1.3]
type=cmrnorm
inputs=conv1.3
channels=256
size=5

#------------fouth layer--------------
```

```
channels=48
filters=192
padding=0
stride=2
filterSize=5
neuron=relu
initW=0.01
initB=0
sharedBiases=1

[rnorm2.2]
type=cmrnorm
inputs=conv2.2
channels=192
size=5

[pool2.2]
type=pool
pool=max
inputs=rnorm2.2
sizeX=3
stride=2
channels=192
neuron=relu

#------------third layer--------------
[conv2.3]
type=conv
inputs=pool2.2
filters=256
padding=1
stride=1
filterSize=3
channels=192
initW=0.01
initB=0
sharedBiases=1
neuron=relu

[rnorm2.3]
type=cmrnorm
inputs=conv2.3
channels=256
size=5

#------------fouth layer--------------
```

```
[conv1.4]
type=conv
inputs=rnorm1.3
filters=256
padding=1
stride=1
filterSize=3
channels=256
initW=0.01
initB=0
neuron=relu

[rnorm1.4]
type=cmrnorm
inputs=conv1.4
channels=256
size=5

#------------fifth layer--------------
[conv1.5]
type=conv
inputs=rnorm1.4
filters=192
padding=0
stride=1
filterSize=3
channels=256
initW=0.01
initB=0
neuron=relu

[rnorm1.5]
type=cmrnorm
inputs=conv1.5
channels=192
size=3


[pool1.5]
type=pool
pool=max
inputs=rnorm1.5
sizeX=3
stride=2
channels=192
neuron=relu
```

```
[conv2.4]
type=conv
inputs=rnorm2.3
filters=256
padding=1
stride=1
filterSize=3
channels=256
initW=0.01
initB=0
neuron=relu

[rnorm2.4]
type=cmrnorm
inputs=conv2.4
channels=256
size=5

#------------fifth layer--------------
[conv2.5]
type=conv
inputs=rnorm2.4
filters=192
padding=0
stride=1
filterSize=3
channels=256
initW=0.01
initB=0
neuron=relu

[rnorm2.5]
type=cmrnorm
inputs=conv2.5
channels=192
size=3


[pool2.5]
type=pool
pool=max
inputs=rnorm2.5
sizeX=3
stride=2
channels=192
neuron=relu
```

```
[concat] # concatenate 2 paths together.
type=eltconcat
inputs=pool1.5,pool2.5

[dropout1]
type=dropout
inputs=concat

[fc6]
type=fc
outputs=9600
inputs=dropout
initW=0.01
initB=0
neuron=relu

[dropout2]
type=dropout
inputs=fc6

[fc7]
type=fc
outputs=100
inputs=dropout
initW=0.01
initB=0
neuron=relu

[softmax]
type=softmax
inputs=fc7,labels
```

Table A.2 The network and parameter configurations

Bibliography

[1] V. Aurich and J. Weule, "Non-linear gaussian filters performing edge preserving diffusion," in Mustererkennung 1995, Springer, 1995, pp. 538-545.

[2] J.A. Calvin, "Regression Models for Categorical and Limited Dependent Variables," Technometrics, vol. 40, pp. 80-81, 1998.

[3] D. Ciresan, U. Meier and J. Schmidhuber, "Multi-column deep neural networks for image classification," in Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, pp. 3642-3649, 2012.

[4] I. Daubechies, Ten lectures on wavelets, SIAM, 1992, .

[5] C. Farabet, C. Couprie, L. Najman and Y. LeCun, "Learning hierarchical features for scene labeling," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 35, pp. 1915-1929, 2013.

[6] K. He, J. Sun and X. Tang, "Guided image filtering," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 35, pp. 1397-1409, 2013.

[7] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R.R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," ArXiv Preprint arXiv:1207.0580, 2012.

[8] D.H. Hubel and T.N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," J.Physiol.(Lond.), vol. 195, pp. 215-243, 1968.

[9] K. Jarrett, K. Kavukcuoglu, M. Ranzato and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in Computer Vision, 2009 IEEE 12th International Conference on, pp. 2146-2153, 2009.

[10] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," ArXiv Preprint arXiv:1404.5997, 2014.

[11] A. Krizhevsky, I. Sutskever and G.E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, pp. 1097-1105, 2012.

[12] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard and L.D. Jackel, "Backpropagation applied to handwritten zip code recognition," Neural Comput., vol. 1, pp. 541-551, 1989.

[13] Y. LeCun, K. Kavukcuoglu and C. Farabet, "Convolutional networks and applications in vision," in Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on, pp. 253-256, 2010.

[14] H. Lee, R. Grosse, R. Ranganath and A.Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in Proceedings of the 26th Annual International Conference on Machine Learning, pp. 609-616, 2009.

[15] S.G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," Pattern Analysis and Machine Intelligence, IEEE Transactions on, vol. 11, pp. 674-693, 1989.

[16] V. Nair and G.E. Hinton, "Rectified linear units improve restricted boltzmann machines," in Proceedings of the 27th International Conference on Machine Learning (ICML-10), pp. 807-814, 2010.

[17] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," International Journal of Computer Vision, vol. 81, pp. 24-52, 2009.

[18] S. Paris, P. Kornprobst, J. Tumblin and F. Durand, Bilateral filtering: Theory and applications, Now Publishers Inc, 2009, .

[19] S. Raman and S. Chaudhuri, "Bilateral filter based compositing for variable exposure photography," in Proceedings of Eurographics, 2009.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla and M. Bernstein, "Imagenet large scale visual recognition challenge," International Journal of Computer Vision, pp. 1-42, 2014.

[21] P.Y. Simard, D. Steinkraus and J.C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," pp. 958, 2003.

[22] E.P. Simoncelli, W.T. Freeman, E.H. Adelson and D.J. Heeger, "Shiftable multiscale transforms," Information Theory, IEEE Transactions on, vol. 38, pp. 587-607, 1992.

[23] S.M. Smith and J.M. Brady, "SUSAN—A new approach to low level image processing," International Journal of Computer Vision, vol. 23, pp. 45-78, 1997.

[24] C. Solomon and T. Breckon, Fundamentals of Digital Image Processing: A practical approach with examples in Matlab, John Wiley & Sons, 2011, .

[25] R. Szeliski, Computer vision: algorithms and applications, Springer Science & Business Media, 2010, .

[26] R. Tibshirani, "Regression shrinkage and selection via the lasso," Journal of the Royal Statistical Society.Series B (Methodological), pp. 267-288, 1996.

[27] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in Computer Vision, 1998. Sixth International Conference on, pp. 839-846, 1998.

[28] S.C. Turaga, J.F. Murray, V. Jain, F. Roth, M. Helmstaedter, K. Briggman, W. Denk and H.S. Seung, "Convolutional networks can learn to generate affinity graphs for image segmentation," Neural Comput., vol. 22, pp. 511-538, 2010.

[29] M.D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in Computer Vision–ECCV 2014, Springer, 2014, pp. 818-833.

[30] I. Sutskever, J. Martens, G. Dahl and G. Hinton, "On the importance of initialization and momentum in deep learning," in Proceedings of The 30th International Conference on Machine Learning, pp. 1139-1147, 2013.

[31] S. Amari, "Backpropagation and stochastic gradient descent method," Neurocomputing, vol. 5, pp. 185-196, 1993.