COMPUTATIONAL STATISTICS: TIME SERIES AND DATA MINING

(Spine title: Plib)

(Thesis format: Monograph)

by

Shuang Ao

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

THE UNIVERSITY OF WESTERN ONTARIO

School of Graduate and Postdoctoral Studies

CERTIFICATE OF EXAMINATION

Supervisor:

.....

Dr. Charles X. Ling

Supervisory Committee:

.....

Dr. W. J. Braun

.....

Dr. A. Bing

Examiners:

.....

Dr. Q. Ring

.....

Dr. W. Fing

.....

Dr. G. Hing

The thesis by

Shuang Ao

entitled:

Computational Statistics: Time Series and Data Mining

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

.....

Date

.....

Chair of the Thesis Examination Board

Acknowlegements

Abstract

This is a really silly abstract.

Keywords: Time series analysis, data mining

Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Transfer

1.1 Issues in Transfer Learning

The motivation of transfer knowledge between different domains is to apply the previous information from the source domain to the target one, assuming that there exists certain relationship, explicit or implicit, between the feature space of these two domains [36]. Technically, previous work can be concluded into solving the following three issues: what, how and when to transfer [46].

What to transfer. Previous work tried to answer this question from three different aspects: selecting transferable instances, learning transferable feature representations and transferable model parameters. Instance-based transfer learning assume that part of the instances in the source domain could be re-used to benefit the learning for the target domain. Lim et al. proposed a method of augmenting the training data by borrowing data from other classes for object detection [30]. Learning transferable features means to learn common feature that can alleviate the bias of data distribution in target domain. Recently, Long et al. proposed a method that can learn transferable features with deep neural network and showed some impressive results on the benchmarks [32]. Parameter transfer approach assumes that the parameters of the model for the source task can be transfered to the target task. Yang et al. proposed Adaptive SVMs by transferring parameters by incorporating the auxiliary classifier trained from source domain [53]. On top of Yang's work, Ayatar et al. proposed PMT-SVM that can determine the transfer regularizer according to the target data automatically [2]. Tommasi et al. proposed Multi-KT that can utilize the parameters from multiple source models for the target classes [46]. Kuzborskij et al. proposed a similar method to learn new categories by leveraging over the known source [26].

When and how to transfer. The question *when to transfer* arises when we want to know

if the information acquired from previous task is relevant to the new one (i.e. in what situation, knowledge should not be transfered). *How to transfer* the prior knowledge effectively should be carefully designed to prevent inefficient and negative transfer. Some previous work consists in using generative probabilistic method [12] [48] [56]. Bayesian learning methods can predict the target domain by combining the prior source distribution to generate a posterior distribution. Alternatively, some previous max margin methods show that it is possible to learn from a few examples by minimizing the Leave-One-Out (LOO) error for the training model [26] [45]. Previous work shows that there is a closed-form implementation of LOO cross-validation that can generate unbiased model estimation for LS-SVM [7].

Our work correspond to the context above. In this chapter, I propose SMITLe based on parameter transfer approach with LS-SVM. I address my work on how to prevent negative transfer when the source data is not accessible. Compared to other works, I propose a novel strongly convex objective function for transfer parameters estimation. I show that SMITLe can converge at the rate of $O(\frac{\log(t)}{t})$. By optimizing this objective function, SMITLe can autonomously adjust the transfer parameters for different prior knowledge. I theoretically and empirically show that, without any data distribution assumption, the superior bound of the training loss for SMITLe is the loss of a method learning directly (i.e. without using any prior knowledge). Experiment results show that when the prior knowledge hurts the transfer procedure, SMITLe can avoid negative transfer by ignoring the unrelated prior knowledge autonomously. Extensive experiments also show that when the prior knowledge is very related (positive transfer), my method can outperform other methods by exploiting the prior knowledge greatly.

1.2 Related Work

In this part, We will introduce the framework my work based on and some related previous work. We first introduce the basic principle of LS-SVM. Based on LS-SVM, several transfer methods are introduced, including A-SVM, PMT-SVM, Multi-KT and MULTIpLe. In these methods, the first two can only adopt the knowledge from single source model and the rest can adopt the knowledge from multiple ones.

1.2.1 LS-SVM Classifier

Least Square SVM is proposed is least squares versions of support vector machines (SVM), which are a set of related supervised learning methods that analyze data and recognize patterns [43]. By replacing the hinge loss in classical SVM with L2 loss, LS-SVM classifier is obtained

by reformulating the minimization problem as:

$$\begin{aligned} \min \quad & L_{LSSVM} = \frac{1}{2}\|w\|^2 + \frac{C}{2}\sum_{i=1}^{l}\varepsilon_i^2 \\ \text{s.t.} \quad & y_i = wx_i + b + \varepsilon_i \quad \text{for} \quad i \in \{1, 2, ..., l\} \end{aligned} \tag{1.1}$$

The primal Lagrangian for this optimization problem given the unconstrained minimization problem can be written as:

$$L(w, b, \alpha, \varepsilon) = \frac{1}{2}\|w\|^2 + \frac{C}{2}\sum_{i=1}^{l}{\varepsilon_i}^2 - \sum_{i=1}^{l}\alpha_i\{wx_i + b + \varepsilon_i - y_i\} \tag{1.2}$$

Where $\alpha = [\alpha_1, ..., \alpha_l]^T$ is the vector of Lagrange multipliers. The solution to minimise this problem is give by:

$$\begin{bmatrix} K + \frac{1}{C}\mathrm{I} & 1 \\ 1^T & 0 \end{bmatrix}\begin{bmatrix} \alpha \\ b \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix} \tag{1.3}$$

Where $K \in R^{l\times l}, K_{i,j} = x_i \times x_j^T$. I is the identity matrix and $\mathbf{1}$ is a column vector with all its elements equal to 1. With:

$$\psi^{-1} = \begin{bmatrix} K + \frac{1}{C}\mathrm{I} & 1 \\ 1^T & 0 \end{bmatrix}^{-1} \tag{1.4}$$

Problem (1.2) can be solved by:

$$\begin{bmatrix} \alpha \\ b \end{bmatrix} = \psi^{-1}\begin{bmatrix} y \\ 0 \end{bmatrix} \tag{1.5}$$

1.2.2 ASVM & PMT-SVM

Adaptive SVM (ASVM) is the first work using LS-SVM for transfer learning for vision related tasks [52]. The goal of ASVM is to minimize the distance between the target hyperplane w and source one w' incorporating with the transfer parameter γ. The objective function is defined as follow:

$$\begin{aligned} \min \quad & L_{ASVM} = \frac{1}{2}\|w - \gamma w'\|^2 + \frac{C}{2}\sum_{i=1}^{l}\varepsilon_i^2 \\ \text{s.t.} \quad & y_i = wx_i + b + \varepsilon_i \quad \text{for} \quad i \in \{1, 2, ..., l\} \end{aligned} \tag{1.6}$$

Here, γ controls the amount of transfer regularization. Intuitively, the regularization term of ASVM is like a spring between w and $\gamma w'$. Equivalently, assume $\|w'\|^2 = 1$ and the regularization term can be expended as:

$$\|w - \gamma w'\|^2 = \|w\|^2 - 2\gamma\|w\|\cos\theta + \gamma^2$$

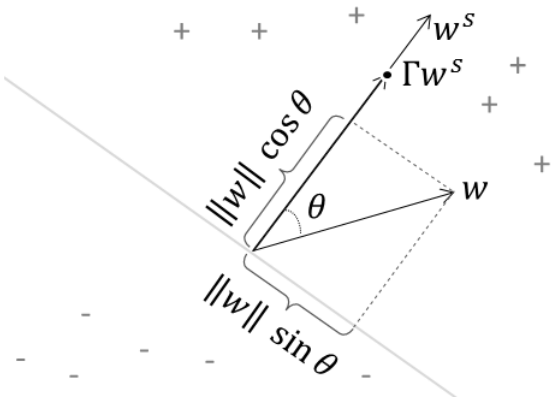


Figure 1.1: Projecting w to w' in PMT-SVM (adapted from [2]).

Where θ is the angel between w and w'. However, the term $-\gamma \|w\| \cos\theta$ also encourages $\|w\|$ to be larger (as this reduces the cost) which prevents margin maximization. Thus γ, which defines the amount of transfer regularization, becomes a tradeoff parameter between margin maximization and knowledge transfer.

Based on this, Projective Model Transfer SVM (PMT-SVM) is proposed to solve the transfer problem by optimizing the following objective function [2]:

$$\begin{aligned} \min \quad & L_{PMT} = \frac{1}{2}\|w\|^2 + \gamma\|Pw'\|^2 + \frac{C}{2}\sum_{i=1}^{l}\varepsilon_i^2 \\ \text{s.t.} \quad & y_i = wx_i + b + \varepsilon_i \quad \text{for} \quad i \in \{1, 2, ..., l\} \\ & w^T w' \geq 0 \end{aligned} \tag{1.7}$$

Where P is the the projection matrix $P = I - \frac{w'^T \times w'}{w' \times w'^T}$. Therefore, $\|Pw\|^2 = \|w\|^2 \sin^2\theta$ is the squared norm of the projection of the w onto the source hyperplane (see Figure 1.1). As $\gamma \to 0$, the loss function (1.7) becomes a classic LS-SVM loss function. Because (1.7) is convex, it can be solved effectively by quadratic optimization.

In summary, both ASVM and PMT-SVM are designed to answer the question: how to transfer by solving some convex objective function. However, they both require a pre-defined parameter γ, which controls the amount of the knowledge to be transfered, to complete the objective function. In most cases, this parameter can only be set according to the background knowledge. Also, both methods can only adopt the knowledge from single source model which limits the their performance.

1.2.3 Multi-KT

To learning from many sources, Multi Model Knowledge Transfer (Multi-KT) is proposed to reply on multiple sources by assigning different weight for each of them [46]. Similar to the

objective function (1.6), its objective function is defined as follow:

$$\begin{aligned} \min \qquad & L_{Multi-KT} = \frac{1}{2}\left\| w - \sum_k \beta_k w'_k \right\|^2 + \frac{C}{2}\sum_{i=1}^{l} \zeta_i \varepsilon_i^2 \\ \text{s.t.} \qquad & y_i = w x_i + b + \varepsilon_i \quad \text{for} \quad i \in \{1, 2, ..., l\} \end{aligned} \tag{1.8}$$

Here, β_k is the weight assigned for the kth source model and ζ_i is defined as:

$$\zeta_i = \begin{cases} \frac{N}{2N^+} & \text{if } y_i = 1 \\ \frac{N}{2N^-} & \text{if } y_i = -1 \end{cases}$$

where N^+ and N^- are number of positive and negative examples respectively and N is the total number of examples.

The primal Lagrangian for optimization problem (1.8) can be written as:

$$L_{Multi-KT}(w, \beta, \varepsilon, \alpha) = \frac{1}{2}\left\| w - \sum_k \beta_k w'_k \right\|^2 + \frac{C}{2}\sum_{i=1}^{l} \zeta_i \varepsilon_i^2 + \sum_{i=1}^{l} \alpha_i \left[w x_i + b + \varepsilon_i - y_i \right] \tag{1.9}$$

So problem (1.8) can be solved once β is set. Different from ASVM and PMT-SVM which require background knowledge to select proper transfer parameter, Multi-KT can estimate the transfer parameter β itself by using the closed-form Leave-One-Out (LOO) error. According to [7], the closed-form LOO error is defined as:

$$y_i - \hat{y}_i = \frac{\alpha_i}{\psi_{ii}^{-1}} \qquad \text{for} \qquad i = 1, ..., l \tag{1.10}$$

Here ψ_{ii}^{-1} is its *ith* diagonal element of ψ^{-1} in (1.4).

To estimate β, a loss function, similar to hinge loss, is defined as:

$$\begin{aligned} \min \qquad & \mathcal{L} = \sum_i^l \zeta_i |1 - y_i \hat{y}_i|_+ \\ \text{s.t.} \qquad & \|\beta\| \leq 1 \end{aligned} \tag{1.11}$$

Here $|x|_+ = max(x, 0)$. Intuitively, if β is properly set, $y_i \hat{y}_i$ should be positive for each i. However, focusing only on the sign of those quantities would result in a non-convex formulation with many local minima. By adding the $|\cdot|_+$ function, formula (1.13) becomes convex and can be solved by gradient descent method.

1.2.4 MULTIpLE

MULticlass Transfer Incremental LEarning (MULTIpLE) focuses on adding a new class to a existing N class source problem while preserving the performance on the old classes [26]. To

preserving the overall performance of the classifier among $N+1$ classes, MULTIpLE contains two parts: incremental learning for existing N classes and transfer learning for the new class.

For the existing N classes, MULTIpLE uses the similar strategy as ASVM, setting the transfer parameter γ to 1. For the new class, it adopts the strategy of Multi-KT, combining knowledge from existing N-class models. As a result, the objective function for the hyperplanes in MULTIpLE is defined as:

$$\begin{aligned} \min \quad & L_{MULTIpLE} = \frac{1}{2}\sum_{n=1}^{N}\|w_n - {w'}_n\|^2 + \frac{1}{2}\left\|w_{N+1} - \sum_{k=1}^{N} w'_k\beta_k\right\|^2 + \frac{C}{2}\sum_{n=1}^{N+1}\sum_{i=1}^{l}\varepsilon_{i,n}^2 \\ \text{s.t.} \quad & \varepsilon_{i,n} = Y_{in} - x_i w_n - b_n \end{aligned} \tag{1.12}$$

Similar like Multi-KT, MULTIpLE uses LOO error in (1.10) to estimate the transfer parameter β for the new class. The objective function for β estimation is defined by [11]:

$$\begin{aligned} \min \quad & \mathcal{L}(\beta, i) = \begin{cases} \max\limits_{n \neq y_i} \left|1 - \hat{Y}_{in}(\beta) - \hat{Y}_{iy_i}(\beta)\right|_+ & : y_i \neq N+1 \\ \left|1 - \hat{Y}_{in}(\beta) - \hat{Y}_{iy_i}(\beta)\right| & : y_i = N+1 \end{cases} \\ \text{s.t.} \quad & \|\beta\| \leq 1 \end{aligned} \tag{1.13}$$

We can find the optimal β with projected subgradient descent [6].

Chapter 2

Convolutional NN

2.1 Background: brief history and recent achievments

2.1.1 Early work with Convlutional Neural Networks

The first simple version of Neural Networks (NNs) trained with supervised learning was proposed in 1960s [37][38]. Networks trained by the Group Method of Data Handling (GMDH) could be the first DL systems of the Feedforward Multilayer Perceptron type [19][39]. Later, there have been many applications of GMDH-style nets [13] [18] [24] [49].

Apart from deep GMDH networks, the Neocognitron, a hierarchical, multilayered artificial neural network, was perhaps the first artificial NN to incorporate the neurophysiological insights [14]. Inspired by Neocognitron, Convolutional NNs (CNNs) was proposed where the rectangular receptive field of a convolutional unit with given weight vector is shifted step by step across a 2-dimensional array of input values, such as the pixels of an image (usually there are several such filters). The results of the previous unit can provide inputs to higher-level units, and so on. Because of its massive weight replication, relatively few parameters may be necessary to describe its behavior.

In In 1989, backpropagation was applied (LeCun et al., 1989, 1990a, 1998) to Convolutional Neural networks with adaptive connections [28]. This combination, incorporating with Max-Pooling and speeding up on graphics cards has become an important part for many modern, competition-winning, feedforward, visual Deep Learners. Later, CNNs achieved good performance on many practical tasks such as MNIST and fingerprint recognition and was commercially used in these fields in 1990s [3] [27].

In the early 2000s, even though GPU-MPCNNs wons several official contests, many practical and commercial pattern recognition applications were dominated by non-neural machine learning methods such as Support Vector Machines (SVMs).

2.1.2 Recent achievements with Convlutional Neural Networks

In 2006, CNN trained with backpropagation set a new MNIST record of 0.39% without using unsupervised pre-training [34]. Also in 2006, an early GPU-based CNN implementation was introduced which was up to 4 times faster than CPU-CNNs [9]. Since then, GPUs or graphics cards have become more and more essential for CNNs in recent years. In 2012, a GPU implemented Max-Pooling CNNs (GPU-MPCNNs) was also the first method to achieve human-competitive performance (around 0.2%) on MNIST [10].

In 2012, an ensemble of GPU-MPCNNs (called AlexNet) achieved best results (top-5 accuracy at 83%) on the ImageNet classification benchmark (ILSVRC2012), which contains 1000 classes and 1.2 million images [25]. After that, excellent results have been achieved by GPU-MPCNNs in image recognition and classification. Many attempts have been made to improve the architecture of AlexNet. With the help of high performance computing system, such as GPUs and large scale distributed cluster, some improvements have been made by either making the network deeper or increasing the size of the training data (with extra training example and data argumentation). By reducing the size of the receptive field and stride, Zeiler and Fergus improve AlexNet by 1.7% on top 5 accuracy [54]. By both adding extra convolutional layers between two pooling layers and reducing the receptive field size, Simonyan and Zisserman built a 19 layer very deep CNN and achieved 92.5% top-5 accuracy [40]. After the AlexNet-like deep CNNs won ILSVRC2012 and ILSVRC2013, Szegedy et al. built a 22 layers deep network, called GoogLeNet and won the 1st prize on ILSVRC2014 for 93.33% top-5 accuracy, almost as good as human annotation[44]. Different from AlexNet-like architecture, GoogLeNet shows another trend of design, utilizing many 1×1 receptive field. Recently, Wu et. al present a image recognition system by aggressive data augmentation on the training data, achieving a top-5 error rate of 5.33% on ImageNet dataset[50]. Searchers from Google successfully trained an ingredient detector system based on GoogLeNet with 220 million images harvested from Google Images and Flickr [33].

Besides its impressive performance on those huge dataset, MPCNNs shows some impressive results by fine-tuning the existing models on small datasets.Zeiler et al. applied their pre-trained model on Caltech-256 with just 15 instances per class and improved the previous state-of-the-art in which about 60 instances were used, by almost 10% [54]. Chatfield et al used their pre-trained model on VOC2007 dataset and outperformed the previous state-of-the-art by 0.9% [8]. Zhou et al. trained AlexNet for Scene Recognition across two datasets with identical categories and provided the state-of-the-art performance using our deep features on all the current scene benchmarks [55]. Hoffman et al. fine-tuned a MPCNNs trained from ImageNet with one example per class, showing that it is possible to use a hybrid approach where

one uses different feature representations for the various domains and produces a combined adapted model [17].

2.2 CNN layers:conv/pool/norm etc

A CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. The input to a convolutional layer is a $m \times m \times r$ image where m is the height and width of the image and r is the number of channels, e.g. an RGB image has $r = 3$.

2.2.1 Convolutional Layer

Convolutional Layer is the core building block of a Convolutional Network, and its output volume can be interpreted as holding neurons arranged in a 3D volume. Natural images have the property of being "stationary", meaning that the statistics of one part of the image are the same as any other part. This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations.

Formally, given some original $h \times w$ input images I, we can train a small autoencoder from $a \times b$ kernel matrix. Also, we have to set other hyperparameters, stride s and padding p. Stride defines the number of pixels the kernel should be moved in each step around the image I and padding defines the number of rows/columns padded to the height and width of the original input (see Figure 2.1). Given a $a \times b$ kernel matrix $W^{(1)}$, bias $b^{(1)}$, padding p and stride s, we can encode the original image I as $f_{conv} = sigmod(W^{(1)}I_p + b^{(1)})$ for $I_p \in I$, giving us f_{conv} (called feature map of $W^{(1)}$), a $\left\lceil \frac{(h-a+2p)}{s} + 1 \right\rceil \times \left\lceil \frac{(w-b+2p)}{s} + 1 \right\rceil$ array of feature map. In general, for any specific input I ($h \times w \times c$ array matrix) of a convolutional layer L, assuming we have k such $a \times b \times c$ kernel matrix, its feature maps f should be a $\left\lceil \frac{(h-a+2p)}{s} + 1 \right\rceil \times \left\lceil \frac{(w-b+2p)}{s} + 1 \right\rceil \times k$ array matrix with $f^{(i)}_{conv} = sigmod(W^{(i)}I_p + b^{(i)})$ for $I_p \in I$ and $i \in 1, \ldots k$.

In real applications, small kernels (3×3, 5×5 and 7×7) are preferred by many different CNN architectures [25] [29] [40] [54]. Recent development of tiny 1×1 kernel shows an improvement on both accuracy and computational efficiency [44].

2.2.2 Pooling Layer

Pooling layer is widely used in all kinds of CNN architecture for dimensional reduction and computational efficiency. After obtaining features maps using convolutional layer, we need to use them for classification. However, applying the feature maps from convolutional layer for classification would be a computationally challenge. Consider for instance images of size

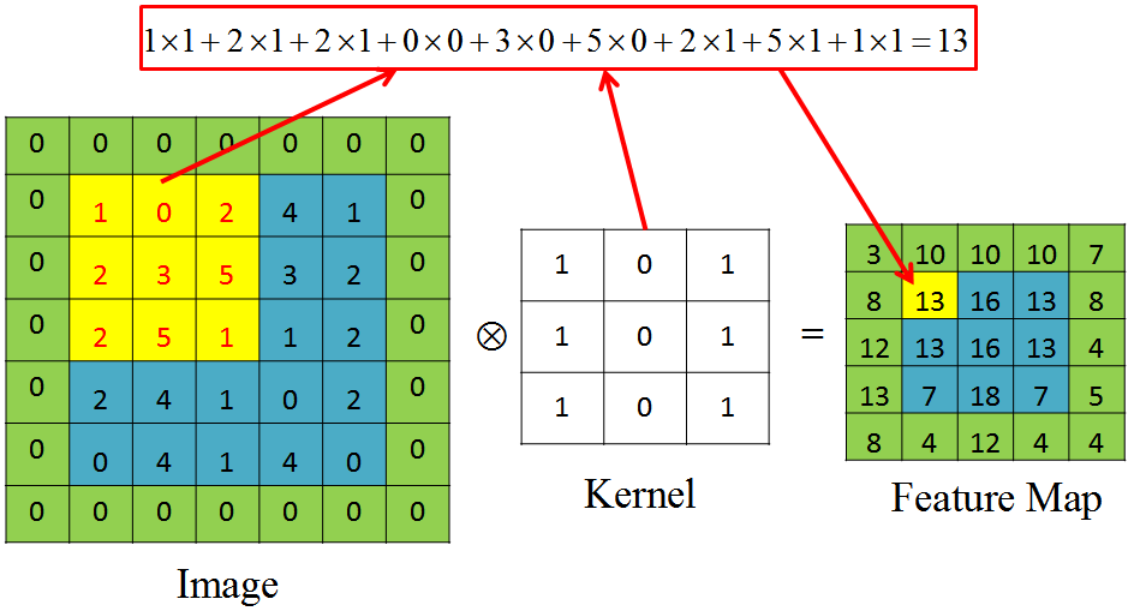


Figure 2.1: Convolution operation with 3×3 kernel, stride 1 and padding 1. $\otimes$ denotes the convolutional operator.

96×96 pixels, and suppose we have learned 400 features over 8×8 inputs. Each convolution results in an output of size $(96 - 8 + 1) \times (96 - 8 + 1) = 7921$, and since we have 400 features, this results in a vector of $892 \times 400 = 3,168,400$ features per example. Learning a classifier from over 3 millions features could lead to severe over-fitting.

Therefore, it is common to periodically insert a (Max) Pooling layer in-between successive convolutional layers in CNN architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The pooling layer works independently on the channel dimension and resize the feature map spatially. For certain $h \times w \times c$ input array matrix, a $a \times b$ Pooling layer with stride s and p padding would output a $\left\lceil \frac{(h-a+2p)}{s} + 1 \right\rceil \times \left\lceil \frac{(w-b+2p)}{s} + 1 \right\rceil \times c$ matrix array.

In general, two kinds of pooling strategy, Max Pooling and Average Pooling, are commonly used in CNN architecture (see Figure 2.2). Average pooling was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice [33] [44]. Max Pooling is been widely used in all kinds CNN architectures [5] [51].

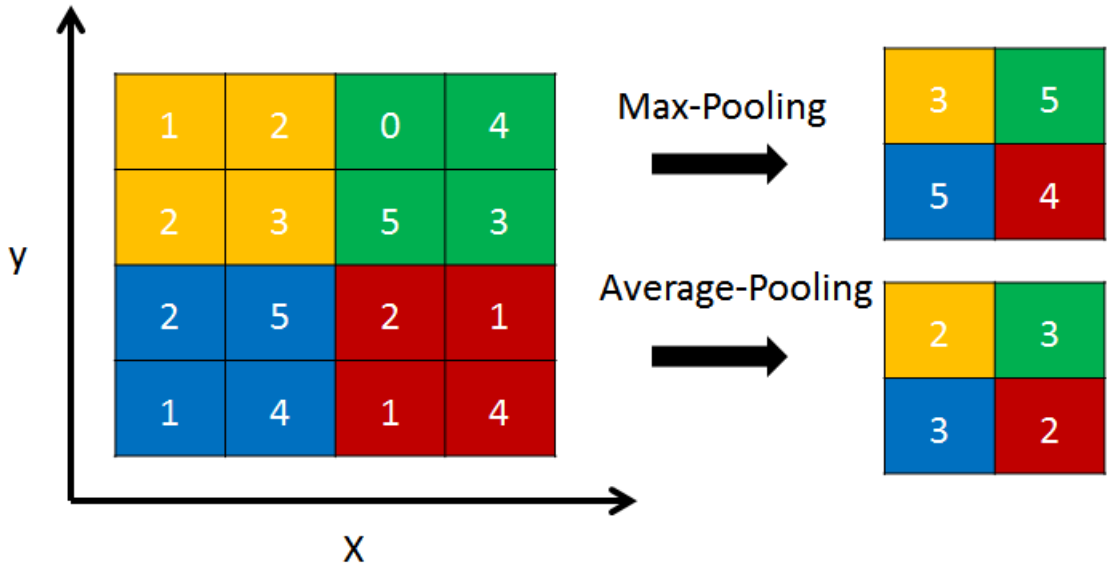


Figure 2.2: 2×2 pooling layer with stride 2 and padding 0.

2.2.3 Fully Connected Layer

Fully Connected (FC) Layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Recent work show that FC layers with Rectified Linear Units and Dropout can greatly improve the learning speed as well as avoid overfitting for deep CNNs [16] [35].

Rectified Linear Units (ReLUs) for Activation

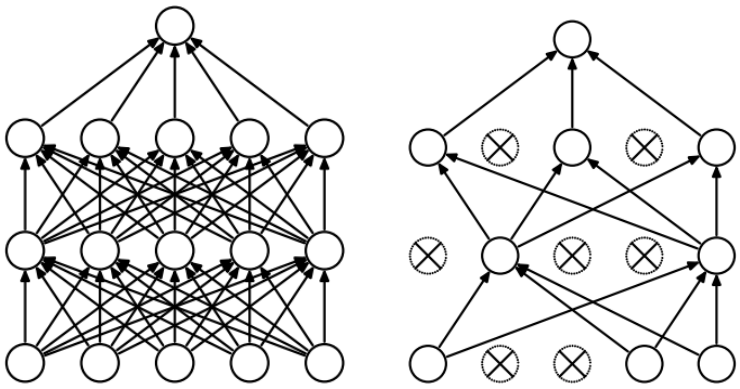
Rectified Linear Units can be considered as replacing each binary unit with sigmoid activation by an infinite number of copies that all have the same weights but have progressively more negative biases. This replacing procedure can be mathematically presented as:

$$\sum_{i}^{N} \sigma(x - i + 0.5) \approx \log(1 + e^{x}) \tag{2.1}$$

where $\sigma(x)$ is the sigmoid function. In practice, Rectified Linear Units use the function

$$f(x) = \log(1 + e^{x}) \approx \max(x, 0) \tag{2.2}$$

as the activation function for approximation [20]. With *max* function, the derivatives of the active ($x > 0$) and inactive neurons are 1 and 0 respectively. As a result, ReLUs can speed up the learning procedure greatly and improve the performance.



(a) Standard Neural Net (b) After Dropout

DropOut

In FC layer, nodes are connected to each other and this leads to a large number of parameters. Generally, larger number of parameters means more power for Neural Networks and more easily prone to overfitting. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training [42]. Technically, dropout can be interpreted as adding extra noise into the training procedure. Without actually adding noise, FC layer with dropout is tolerant of higher level of noise (20 %-50%). Randomly dropping out the nodes, for any node in FC layer, it can't rely on the other nodes to adjust its result. By eliminating the co-adaptation of hidden units, dropout becomes a technique that can be applied to any general domain and improve the performance of neural nets.

2.3 Datasets

In this section, we introduce some details about the two architectures and the food datasets used in our experiments.

2.3.1 Models

In this paper, AlexNet and GoogLeNet are their Caffe [21] implementations and all the results for a specific CNN architecture are obtained from single model.

AlexNet contains 5 layers followed by the auxiliary classifier which contains 2 fully connected layers (FC) and 1 softmax layer. Each of the first two layers can be subdivided into 3 components: convolutional layer with rectified linear units (ReLUs), local response normalization layer (LRN) and max pooling layer. Layer 3 and layer 4 contain just convolutional layer

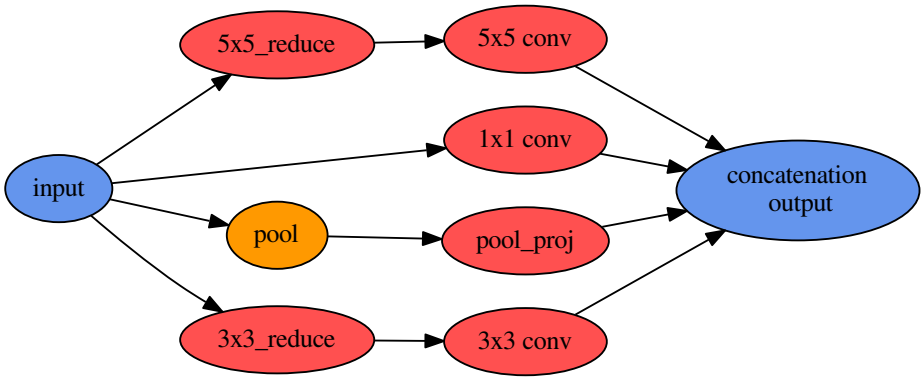


Figure 2.3: Inception Module. $n \times n$ stands for size n receptive field, $n \times n\_reduce$ stands for the 1×1 convolutional layer before the $n \times n$ convolution layer and $pool\_proj$ is another 1×1 convolutional layer after the MAX pooling layer. The output layer concatenates all its input layers.

with ReLUs while layer 5 is similar to the first two layers except for the LRN. For each of the fully connected layer, 1 ReLUs and 1 dropout [42] layer are followed.

GoogLeNet shows another trend of deep CNN architecture with lots of small receptive fields. There are 9 Inception modules in GoogLeNet and Figure 2.3 shows the architecture of a single inception module. Inspired by [31], lots of 1×1 convolutional layers are used for computational efficiency. Another interesting feature of GoogLeNet is that there are two extra auxiliary classifiers in intermediate layers. During the training procedure, the loss of these two classifiers are counted into the total loss with a discount weight 0.3, in addition with the loss of the classifier on top. More architecture details can be found from [44].

2.3.2 Food Datasets

Besides ImageNet dataset, there are many popular benchmark datasets for image recognition such as Caltech dataset and CIFAR dataset, both of which contain hundreds of classes. However, in this paper, we try to focus on a more specific area, food classification. Compared to other recognition tasks, there are some properties of the food (dishes) which make the tasks become a real challenge:

- Food doesn't have any distinctive spatial layout: for other tasks like scene recognition, we can always find some discriminative features such as buildings or trees, etc;

- Food class is a small sub-category among all the categories in daily life, so the inter-class variation is relatively small; on the other hand, the contour of a food varies depending on many aspects such as the point of the view or even its components.

These properties make food recognition catastrophic for some recognition algorithms. Therefore, the training these two architectures on the food recognition task can reveal some important aspects of themselves and help people better understand them. In this paper, we use two image datasets Food-256 [22][1] and Food-101 [4][2]. It is worthy to mention that PFID dataset is also a big public image database for classification, but their images are collected in a laboratory condition which is considerably not applicable for real recognition task.

Food-256 Dataset. This is a relatively small dataset containing 256 kinds of foods and 31644 images from various countries such as French, Italian, US, Chinese, Thai, Vietnamese, Japanese and Indonesia. The distribution among classes is not even and the biggest class (vegetable tempura) contains 731 images while the smallest one contains just 100 images. For this small dataset, we randomly split the data into training and testing set, using around 80% (25361 images) and 20% (6303 images) of the original data respectively and keep the class distribution in these two sets uniform. The collector of this dataset also provides boundary box for each image to separate different foods and our dataset is cropped according to these boundary boxes.

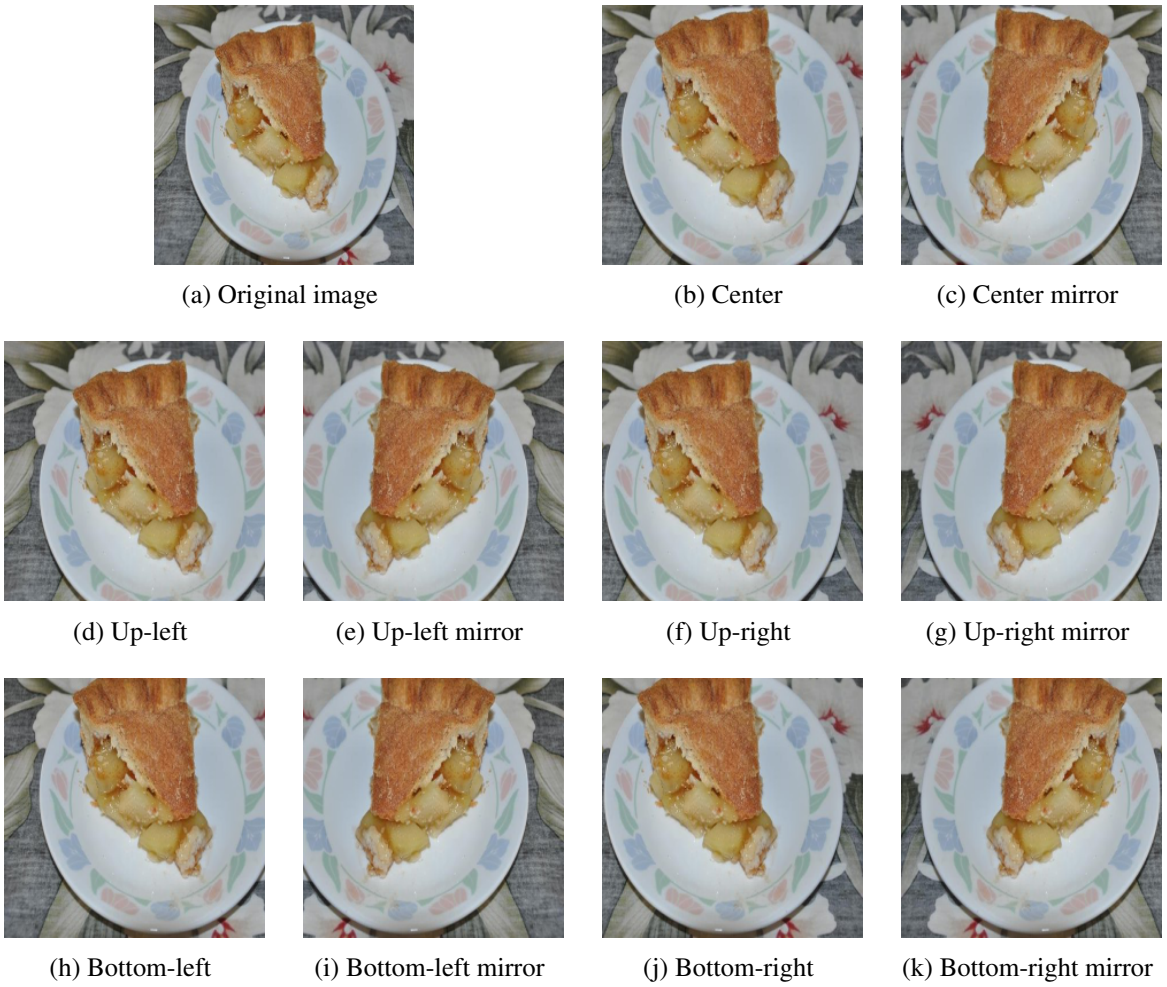
Food-101 Dataset. This dataset contains 101-class real-world food (dish) images which were taken and labeled manually. The total number of images is 101,000 and there are exactly 1000 images for each class. Also, each class has been divided into training and testing set containing 750 images and 250 images respectively by its collector. The testing set is well cleaned manually while the training set is not well cleaned on purpose. This noisy training set is more similar to our real recognition situation and it is also a good way to see the effect of the noise on these two architectures.

2.3.3 Data Argumentation

In this section, we introduce some data argumentation methods in our work to enrich our training data. Previous work shows that with intensive argumentation for the training data, the performance of CNN model can be improved [50]. Data argumentation is an efficient way to enrich the data. There are also some techniques that can applied to enlarge the dataset such as subsampling and mirroring. The original images are firstly resized to 256×256 pixels. We crop the 4 corners and center for each image according to the input size of each model and flap

[1]Dataset can be found http://foodcam.mobi/dataset.html

[2]Dataset can be found http://www.vision.ee.ethz.ch/datasets_extra/food-101



(a) Original image (b) Center (c) Center mirror

(d) Up-left (e) Up-left mirror (f) Up-right (g) Up-right mirror

(h) Bottom-left (i) Bottom-left mirror (j) Bottom-right (k) Bottom-right mirror

Figure 2.4: Crop area from original image

the 5 cropped images to obtain 10 crops. For the testing set, the prediction of an image is the average prediction of the 10 crops (see Figure 2.4).

Before cropping subsamples from the original image, we also use other argumentation methods such as color casting and vignette etc., to enrich our data and make our model less sensitive to lighting changes and other invariance (see Figure 2.5).

Compared to color shifting in [25], we use color casting to alter the intensities of the RGB channels in training images. For each image, we firstly use a random boolean parameter to determine whether its R, G, and B channel should be changed. For any channel that should be changed, we add a random integer ranging from $[-20, 20]$ to this specific channel. We also apply vignetting effect to the original image. In our implementation, we apply a 2D Gaussian kernel on the original image for vignetting. The two parameter σ_x and σ_y are randomly chosen from $[160, 200)$. We also apply some geometric transformation such as stretching and rotation,

on the original image for data argumentation. In summary, we enriched the data by 11 times, 3 times color shifting, 2 times vignetting, 4 times stretching and 1 time rotation and plus the original image.

2.4 Experimental Discuss

Training a CNN with millions of parameters on a small dataset could easily lead to horrible overfitting. But the idea of supervised pre-training on some huge image datasets could preventing this problem in certain degree. Compared to other randomly initialized strategies with certain distribution, supervised pre-training is to initialize the weights according to the model trained from a specific task. Indeed, initialization with pre-trained model has certain bias as there is no single dataset including all the invariance for natural images [1], but this bias can be reduced as the pre-trained image dataset increases and the fine-tuning should be benefit from it.

2.4.1 Pre-training and Fine-tuning

We conduct several experiments on both architectures and use different training initialization strategies for both Food-256 and Food-101 datasets. The scratch models are initialized with Gaussian distribution for AlexNet and Xavier algorithm for GoogLeNet[15]. These two initializations are used for training the original models for the ImageNet task. The ft-last and fine-tuned models are initialized with the weights pre-trained from ImageNet dataset. For the ft-last model, we just re-train the fully connected layers while the whole network is fine-tuned for the fine-tune model.

Table 2.1: Top-5 Accuracy in percent on fine-tuned, ft-last and scratch model for two architectures

	AlexNet		GoogLeNet	
	Food-101	Food-256	Food-101	Food-256
Fine-tune	**88.12**	**85.59**	**93.51**	**90.66**
Ft-last	76.49	79.26	82.84	83.77
Scratch	78.18	75.35	90.45	81.20

From Table 2.1 we can see that fine-tuning the whole network can improve the performance of the CNN for our task. Compared to other traditional computer vision methods (see Table

(a) Original image (b) Red casting (c) Green casting

(d) Blue casting (e) RGB casting (f) Vignette

(g) More vignette (h) Horizontal stretch (i) More horizontal stretch

(j) Vertical stretch (k) More vertical stretch (l) Rotation

Figure 2.5: Different data argumentation methods

Table 2.2: Accuracy compared to other method on Food-256 dataset in percent

	fv+linear [23]	GoogLeNet	AlexNet
Top1	50.1	**70.13**	63.82
Top5	74.4	**90.66**	85.59

Table 2.3: Top-1 accuracy compared to other methods on Food-101 dataset in percent

	RFDC[4]	MLDS(≈[41])	GoogLeNet	AlexNet
Top1 accuracy	50.76	42.63	**78.11**	66.40

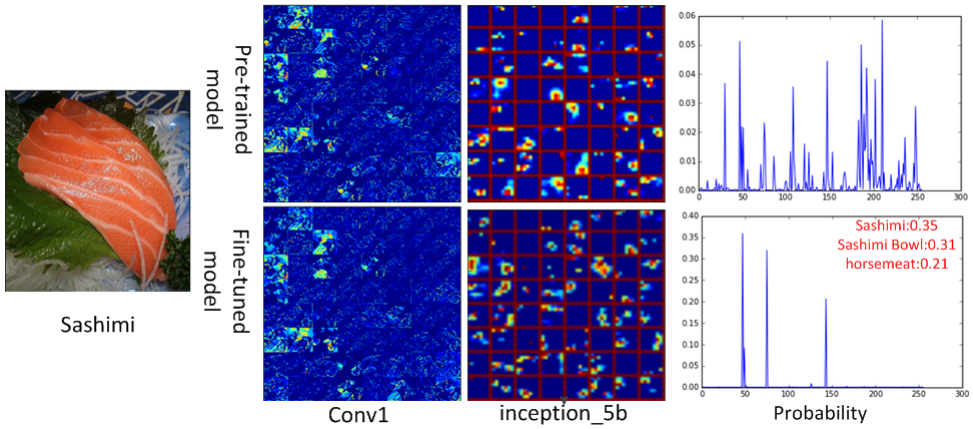


Figure 2.6: Visualization of some feature maps of different GoogLeNet models in different layers for the same input image. 64 feature maps of each layer are shown. Conv1 is the first convolutional layer and Inception_5b is the last convolutional layer.

2.2 and 2.3), GoogLeNet outperforms the other methods with large margins and we provide the state-of-the-art performance of these two food image datasets.

In Figure 2.6 we visualize the feature maps of the pre-trained GoogLeNet model and fined-tuned GoogLeNet model with the same input image for some layers. We can see that the feature maps of the lower layer are similar as the lower level features are similar for most recognition tasks. Then we can see that the feature maps in the high-level are different which leads to totally different recognition results. Since only the last layer (auxiliary classifier) of the ft-last model is optimized, we can infer that the higher level features are more important which is consistent with our intuition. Also from Table 2.1, it is interesting to see that for the Food-101 task, the accuracy of the scratch models outperforms the pre-trained models. Since Food-101 is a relatively large dataset with 750 images per class while Food-256 dataset is an imbalanced small one, this indicates that it is difficult to obtain a good deep CNN model while the data is insufficient.

From Table 2.1 we can see that GoogLeNet always performances better than AlexNet on both datasets. This implies that the higher level features of GoogLeNet are more discriminative compared to AlexNet and this is due to the special architecture of its basic unit, Inception module. Table 2.4 and 2.5 show the weights' cosine similarity of each layer between the fine-tuned models and their pre-trained models. From the results we can see that the weights in the

low layer are more similar which implies that these two architectures can learn the hierarchical features. As the low level features are similar for most of the tasks, the difference of the objects is determined by high-level ones which are the combination of these low level features. Also from Table 2.5, we can observe that, the weights of the pre-trained and fine-tuned models are extremely similar in AlexNet . This can be caused by the size of receptive filed. Since ReLUs are used in both architectures, vanishing gradients do not exist. Rectified activation function is mathematically given by:

$$h = \max(w^T x, 0) = \begin{cases} w^T x & w^T x > 0 \\ 0 & else \end{cases} \tag{2.3}$$

The ReLU is inactivated when its input is below 0 and its partial derivative is 0 as well. Sparsity can improve the performance of the linear classifier on top, but on the other hand, sparse representations make the network more difficult to train as well as fine-tune. The derivative of the filter is $\frac{\partial J}{\partial w} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w} = \frac{\partial J}{\partial y} * x$ where $\frac{\partial J}{\partial y}$ denotes the partial derivative of the activation function, $y = w^T x$ and x denotes the inputs of the layer. The sparse input could lead to sparse filter derivative for back propagation which would eventually prevent the errors passing down effectively. Therefore, the filters of the fine-tuned AlexNet is extremely similar. Compared to large receptive field used in AlexNet, the inception module in GoogLeNet employs 2 additional $n \times n\_reduced$ convolutional layers before the 3×3 and 5×5 convolutional layers (see Figure 2.3). Even though the original purpose of these two 1×1 convolutional layer is for computational efficiency, these 2 convolutional layers tend to squeeze their sparse inputs and generate a dense outputs for the following layer. We can see from Table 2.6 that the sparsity of the $n \times n\_reduce$ layers are denser than other layers within the inception module. This makes the filters in the following layer more easily to be trained for transfer learning and generate efficient sparse representations.

The unique structure of the Inception module guarantees that the sparse outputs from previous layer can be squeezed with the 1×1 convolutional layers and feed to convolutional layers with bigger receptive field to generate sparser representation. The squeeze action promises the back propagation error can be transferred more efficiently and makes the whole network more flexible to fit different recogntion tasks.

2.4.2 Learning across the datasets

From the previous experiments we can see that pre-training on the ImageNet dataset can improve the performance of the deep convolutional neural network on our specific area. In this part, we will discuss the generalization ability within the food recognition problem. Zhou et al. trained AlexNet for Scene Recognition across two datasets with identical categories [55]. But

Table 2.4: Cosine similarity of the layers in inception modules between fine-tuned models and pre-trained model for GoogLeNet

food256						
	1x1	3x3_reduce	3x3	5x5_reduce	5x5	pool_proj
inception_3a	0.72	0.72	0.64	0.67	0.73	0.69
inception_3b	0.59	0.64	0.53	0.70	0.60	0.56
inception_4a	0.46	0.53	0.54	0.50	0.67	0.38
inception_4b	0.55	0.58	0.63	0.52	0.69	0.41
inception_4c	0.63	0.64	0.63	0.57	0.68	0.52
inception_4d	0.60	0.62	0.60	0.58	0.68	0.50
inception_4e	0.60	0.61	0.67	0.61	0.68	0.50
inception_5a	0.51	0.53	0.58	0.48	0.60	0.39
inception_5b	0.40	0.44	0.50	0.41	0.59	0.40
food101						
	1x1	3x3_reduce	3x3	5x5_reduce	5x5	pool_proj
inception_3a	0.71	0.72	0.63	0.67	0.73	0.68
inception_3b	0.56	0.63	0.50	0.71	0.60	0.53
inception_4a	0.43	0.50	0.50	0.47	0.62	0.36
inception_4b	0.48	0.52	0.57	0.50	0.67	0.35
inception_4c	0.57	0.61	0.59	0.53	0.63	0.47
inception_4d	0.54	0.58	0.53	0.54	0.64	0.44
inception_4e	0.53	0.54	0.61	0.55	0.62	0.42
inception_5a	0.43	0.47	0.53	0.45	0.57	0.34
inception_5b	0.36	0.39	0.46	0.38	0.52	0.37

Table 2.5: Cosine similarity of the layers between fine-tuned models and pre-trained model for AlexNet

	conv1	conv2	conv3	conv4	conv5	fc6	fc7
food256	0.997	0.987	0.976	0.976	0.978	0.936	0.923
food101	0.996	0.984	0.963	0.960	0.963	0.925	0.933

Table 2.6: Sparsity of the output for each unit in GoogLeNet inception module for training data from Food101 in percent

	1x1	3x3_reduce	3x3	5x5_reduce	5x5	pool_proj
inception_3a	69.3 ± 1.3	69.6 ± 1.1	80.0 ± 1.0	64.1 ± 2.2	75.8 ± 1.6	76.2 ± 5.4
inception_3b	92.8 ± 0.9	76.5 ± 0.9	94.7 ± 0.9	71.6 ± 2.3	94.4 ± 0.5	94.7 ± 1.6
inception_4a	90.9 ± 0.9	70.0 ± 1.2	93.8 ± 1.1	63.3 ± 4.0	91.9 ± 1.8	95.1 ± 2.0
inception_4b	71.9 ± 1.6	67.5 ± 1.2	75.4 ± 1.0	58.5 ± 2.6	78.9 ± 1.6	85.6 ± 3.6
inception_4c	75.1 ± 2.4	72.6 ± 1.3	81.0 ± 2.0	66.3 ± 6.1	79.7 ± 3.6	88.1 ± 3.3
inception_4d	87.3 ± 2.7	78.0 ± 2.2	88.0 ± 1.6	67.9 ± 3.1	88.9 ± 2.8	93.0 ± 2.2
inception_4e	91.8 ± 1.1	62.3 ± 2.2	91.0 ± 2.5	49.5 ± 3.7	94.0 ± 1.0	92.3 ± 1.5
inception_5a	78.7 ± 1.6	66.5 ± 1.7	82.3 ± 2.6	59.9 ± 3.2	86.4 ± 2.3	87.1 ± 2.6
inception_5b	88.2 ± 2.3	86.8 ± 1.6	83.3 ± 4.4	84.0 ± 3.1	81.4 ± 5.3	94.7 ± 1.5

for more complex situation, such as two similar datasets with a little overlapped categories, we are very interested in exploring whether deep CNN can still successfully handle. Therefore, we conduct the following experiment to stimulate a more challenging real world problem: transferring the knowledge from the fine-tuned Food-101 model to a target set, Food-256 dataset. To make the experiment more practical, we limit the number of samples per category from Food-256 for training, because if we want to build a our own model using deep CNN for a specific task, the resource is always limited and it is exhausted to collect hundreds of labeled images for each category.

The Food-101 and Food-256 datasets share about 46 categories of food even though the images in the same category may vary across these two datasets. The types of food in Food-101 are mainly western style while most types of food in Food-256 are typical Asian foods. We compare the top-5 accuracy trained from different size of subset for Food-256 on different pre-trained model and the results are shown in Table 2.7. The ImageNet columns denote using the model pre-trained from ImageNet dataset as the pre-trained model and Food101_ft columns denote using the fine-tuned Food-101 model (the same one in Table 2.1) as the pre-trained model.

From the result of Table 2.7 we can see that, with this further transfer learning, both CNNs can achieve around 95% of the accuracy trained on full dataset while just utilizing about half of them (50 per class, 12800 of 25361 images). This indicates that when there is not enough labeled data, with its strong generalization ability, deep CNN trained from general task can still achieve satisfying result and perform even better when an additional relevant dataset is involved. This encouraging result may attract more people to use deep CNN for their specific

task and continue to explore the potential of the existing architecture as well as designing new ones.

Table 2.7: Top5 Accuracy for transferring from Food101 to subset of Food256 in percent

	AlexNet		GoogLeNet	
instances per class	ImageNet	Food101_ft	ImageNet	Food101_ft
20	68.80	75.12	74.54	77.77
30	73.15	77.02	79.21	81.06
40	76.04	80.23	81.76	83.52
50	78.90	81.66	84.22	85.84
all	85.59	87.21	90.66	90.65

Bibliography

[1] Pulkit Agrawal, Ross Girshick, and Jitendra Malik. Analyzing the performance of multilayer neural networks for object recognition. In *Computer Vision–ECCV 2014*, pages 329–344. Springer, 2014.

[2] Yusuf Aytar and Andrew Zisserman. Tabula rasa: Model transfer for object category detection. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2252–2259. IEEE, 2011.

[3] Pierre Baldi and Yves Chauvin. Neural networks for fingerprint recognition. *Neural Computation*, 5(3):402–418, 1993.

[4] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *Computer Vision–ECCV 2014*, pages 446–461. Springer, 2014.

[5] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 111–118, 2010.

[6] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[7] Gavin C Cawley. Leave-one-out cross-validation based model selection criteria for weighted ls-svms. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 1661–1668. IEEE, 2006.

[8] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*, 2014.

[9] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[10] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[11] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292, 2002.

[12] Jesse Davis and Pedro Domingos. Deep transfer via second-order markov logic. In *Proceedings of the 26th annual international conference on machine learning*, pages 217–224. ACM, 2009.

[13] Stanley J Farlow. *Self-organizing methods in modeling: GMDH type algorithms*, volume 54. CrC Press, 1984.

[14] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[15] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[16] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[17] Judy Hoffman, Eric Tzeng, Jeff Donahue, Yangqing Jia, Kate Saenko, and Trevor Darrell. One-shot adaptation of supervised deep convolutional models. *arXiv preprint arXiv:1312.6204*, 2013.

[18] Saburo Ikeda, Mikiko Ochiai, and Yoshikazu Sawaragi. Sequential gmdh algorithm and its application to river flow prediction. *Systems, Man and Cybernetics, IEEE Transactions on*, (7):473–479, 1976.

[19] Alexey Grigorevich Ivakhnenko and Valentin Grigorévich Lapa. *Cybernetic predicting devices*. CCM Information Corporation, 1965.

[20] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[21] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.

[22] Y. Kawano and K. Yanai. Automatic expansion of a food image dataset leveraging existing categories with domain adaptation. In *Proc. of ECCV Workshop on Transferring and Adapting Source Knowledge in Computer Vision (TASK-CV)*, 2014.

[23] Yoshiyuki Kawano and Keiji Yanai. Foodcam-256: A large-scale real-time mobile food recognitionsystem employing high-dimensional features and compression of classifier weights. In *Proceedings of the ACM International Conference on Multimedia*, MM '14, pages 761–762, New York, NY, USA, 2014. ACM.

[24] Tadashi Kondo and Junji Ueno. Multi-layered gmdh-type neural network self-selecting optimum neural network architecture and its application to 3-dimensional medical image recognition of blood vessels. *International Journal of innovative computing, information and control*, 4(1):175–187, 2008.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[26] Ilja Kuzborskij, Francesco Orabona, and Barbara Caputo. From n to n+ 1: Multiclass transfer incremental learning. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3358–3365. IEEE, 2013.

[27] B Boser Le Cun, John S Denker, D Henderson, Richard E Howard, W Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer, 1990.

[28] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[29] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[30] Joseph Jaewhan Lim. *Transfer learning by borrowing examples for multiclass object detection*. PhD thesis, Massachusetts Institute of Technology, 2012.

[31] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[32] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France*, pages 97–105, 2015.

[33] Jonathan Malmaud, Jonathan Huang, Vivek Rathod, Nick Johnston, Andrew Rabinovich, and Kevin Murphy. What's cookin'? interpreting cooking videos using text, speech and vision. *arXiv preprint arXiv:1503.01558*, 2015.

[34] Christopher Poultney Marc'Aurelio Ranzato, Sumit Chopra, and Yann Lecun. Efficient learning of sparse representations with an energy-based model. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS (NIPS 2006*. Citeseer, 2006.

[35] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[36] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

[37] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[38] Frank Rosenblatt. Principles of neurodynamics. 1962.

[39] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[41] Saurabh Singh, Abhinav Gupta, and Alexei A Efros. Unsupervised discovery of mid-level discriminative patches. In *Computer Vision–ECCV 2012*, pages 73–86. Springer, 2012.

[42] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[43] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[44] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[45] Tatiana Tommasi, Francesco Orabona, and Barbara Caputo. Safety in numbers: Learning categories from few examples with multi model knowledge transfer. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3081–3088. IEEE, 2010.

[46] Tatiana Tommasi, Francesco Orabona, and Barbara Caputo. Learning categories from few examples with multi model knowledge transfer. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(5):928–941, 2014.

[47] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[48] Xuezhi Wang, Tzu-Kuo Huang, and Jeff Schneider. Active transfer learning under model shift. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1305–1313, 2014.

[49] Marcin Witczak, Józef Korbicz, Marcin Mrugalski, and Ron J Patton. A gmdh neural network-based approach to robust fault diagnosis: Application to the damadics benchmark problem. *Control Engineering Practice*, 14(6):671–683, 2006.

[50] Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. Deep image: Scaling up image recognition. *arXiv preprint arXiv:1501.02876*, 2015.

[51] Jianchao Yang, Kai Yu, Yihong Gong, and Tingwen Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1794–1801. IEEE, 2009.

[52] Jun Yang, Rong Yan, and Alexander G Hauptmann. Adapting svm classifiers to data with shifted distributions. In *Data Mining Workshops, 2007. ICDM Workshops 2007. Seventh IEEE International Conference on*, pages 69–76. IEEE, 2007.

[53] Jun Yang, Rong Yan, and Alexander G Hauptmann. Cross-domain video concept detection using adaptive svms. In *Proceedings of the 15th international conference on Multimedia*, pages 188–197. ACM, 2007.

[54] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

[55] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 487–495. Curran Associates, Inc., 2014.

[56] Tianyi Zhou and Dacheng Tao. Multi-task copula by sparse graph regression. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 771–780. ACM, 2014.

Appendix A

Proofs of Theorems

A.1 Proof of Theorem1

Proof For simplification, let $\delta_i = 1$ if $i = N + 1$ and 0 otherwise, and $\theta_{ij} = \alpha''_{ij}\left(1 - \delta_j\right)/\psi_{ii}^{-1}$. Eq. (**??**) can be written as:

$$\xi_i(\gamma,\beta) = \max_n \left\{ \varepsilon_{ny_i} - 1 + \frac{\left(\alpha'_{iy_i} - \alpha'_{in}\right)}{\psi_{ii}^{-1}} + \theta_{in}\gamma_n - \theta_{iy_i}\gamma_{y_i} + \left(\delta_n - \delta_{y_i}\right)\sum_k \frac{\alpha''_{ik}\beta_k}{\psi_{ii}^{-1}} \right\} \tag{A.1}$$

When $\gamma = \beta = \mathbf{0}$, from Eq. (A.1) we can get:

$$\bar{\xi}_i = \max_n \left[\varepsilon_{ny_i} - 1 + \frac{\left(\alpha'_{iy_i} - \alpha'_{in}\right)}{\psi_{ii}^{-1}} \right]$$

To obtain the optimal value of γ and β, we have to seek the saddle point of the Lagrangian problem in (**??**) by finding the minimum for the prime variables $\{\gamma, \beta, \xi\}$ and the maximum for the dual variables η. To find the minimum of the primal problem, we require:

$$\frac{\partial L}{\partial \xi_i} = 1 - \sum_n \eta_{in} = 0 \rightarrow \sum_n \eta_{in} = 1$$

Similarly, for γ and β, we require:

$$\begin{aligned} \frac{\partial L}{\partial \gamma_n} &= \lambda_1\gamma_n + \sum_i \eta_{in}\theta_{in} - \sum_{i,n=y_i}\left(\sum_q \eta_{iq}\right)\theta_{in}\gamma_n \\ &=_1 \lambda_1\gamma_n + \sum_i \eta_{in}\theta_{in} - \sum_i \varepsilon_{ny_i}\theta_{in} = 0 \\ &\Rightarrow \gamma_n^* = \frac{1}{\lambda_1}\sum_i \left(\varepsilon_{ny_i} - \eta_{in}\right)\theta_{in} \end{aligned} \tag{A.2}$$

In $=_1$ we use the facts that $\sum_n \eta_{in} = 1$ and use ε_{ny_i} to replace it.

$$\begin{aligned} \frac{\partial L}{\partial \beta_n} &= \lambda_2 \beta_n + \left[\sum_{i,n} \frac{\eta_{in} \alpha''_{in}}{\psi_{ii}^{-1}} \left(\delta_n - \delta_{y_i} \right) \right] = 0 \\ &\Rightarrow \beta_n^* = \frac{1}{\lambda_2} \sum_{i,n} \frac{\eta_{in} \alpha''_{in}}{\psi_{ii}^{-1}} \left(\delta_{y_i} - \delta_n \right) \end{aligned} \tag{A.3}$$

As the strong duality holds,the primal and dual objectives coincide. Plug Eq (A.2) and (A.3) into Eq. (**??**), we have:

$$\sum_{i,n} \eta_{in} \left[1 - \varepsilon_{ny_i} + \hat{Y}_{in} (\gamma^*, \beta^*) - \hat{Y}_{iy_i} (\gamma^*, \beta^*) - \xi_i^* \right] = 0$$

Expand the equation above, we have:

$$\sum_{i,n} \eta_{in} \left[\varepsilon_{n,y_i} - 1 + \frac{\left(\alpha'_{iy_i} - \alpha'_{in} \right)}{\psi_{ii}^{-1}} - \xi_i \right] = \lambda_1 \sum_r \left\| \gamma_r^* \right\|^2 + \lambda_2 \sum_r \left\| \beta_r^* \right\|^2 \geq 0$$

Rearranging the above, we obtain:

$$\sum_{i,n} \eta_{in} \left[\varepsilon_{n,y_i} - 1 + \frac{\left(\alpha'_{iy_i} - \alpha'_{in} \right)}{\psi_{ii}^{-1}} \right] \geq \sum_{i,n} \eta_{in} \xi_i = \sum_i \xi_i \tag{A.4}$$

The left-hand side of Inequation (A.4) can be bounded by:

$$\begin{aligned} &\sum_{i,n} \eta_{in} \left[\varepsilon_{ny_i} - 1 + \frac{\left(\alpha'_{iy_i} - \alpha'_{in} \right)}{\psi_{ii}^{-1}} \right] \\ &\leq \sum_i \left(\sum_n \eta_{in} \max_r \left\{ \varepsilon_{ry_i} - 1 + \frac{\left(\alpha'_{iy_i} - \alpha'_{ir} \right)}{\psi_{ii}^{-1}} \right\} \right) \\ &= \sum_i \left(\sum_n \eta_{in} \bar{\xi}_i \right) = \sum_i \bar{\xi}_i \end{aligned} \tag{A.5}$$

A.2 Configuration of GoogLeNet

Table A.1: Configuration of GoogLeNet

type	patch size / stride	output size	depth	#1x1	#3x3 reduce	#3x3	#5x5 reduce	#5x5 pool	proj
convolution	7x7/2	112x112x64	1						
max pool	3x3/2	56x56x64	0						
convolution	3x3/1	56x56x192	2		64	192			
max pool	3x3/2	28x28x192	0						
inception(3a)		28x28x256	2	64	96	128	16	32	32
inception(3b)		28x28x480	2	128	128	192	32	96	64
maxpool	3x3/2	14x14x480	0						
inception(4a)		14x14x512	2	192	96	208	16	48	64
inception(4b)		14x14x512	2	160	112	224	24	64	64
inception(4c)		14x14x512	2	128	128	256	24	64	64
inception(4d)		14x14x528	2	112	144	288	32	64	64
inception(4e)		14x14x832	2	256	160	320	32	128	128
maxpool	3x3/2	7x7x832	0						
inception(5a)		7x7x832	2	256	160	320	32	128	128
inception(5b)		7x7x1024	2	384	192	384	48	128	128
avg pool	7x7/1	1x1x1024	0						
dropout (40%)		1x1x1024	0						
linear		1x1x1000	1						
softmax		1x1x1000	0						

Curriculum Vitae

Name:	Shuang Ao
Post-Secondary Education and Degrees:	La La School La La Land 1996 - 2000 M.A. University of Western Ontario London, ON 2008 - 2012 Ph.D.
Honours and Awards:	NSERC PGS M 2006-2007
Related Work Experience:	Teaching Assistant The University of Western Ontario 2008 - 2012

Publications:

La La