CARDEA: A CONTEXT-AWARE AND INTERACTIVE VISUAL PRIVACY CONTROL FRAMEWORK

by

RUI ZHENG

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Master of Philosophy
in Computer Science and Engineering

October 2016, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

RUI ZHENG

CARDEA: A CONTEXT-AWARE AND INTERACTIVE VISUAL PRIVACY CONTROL FRAMEWORK

by

RUI ZHENG

This is to certify that I have examined the above M.Phil. thesis

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by

the thesis examination committee have been made.

ASSISTANT PROF. PAN. HUI, THESIS SUPERVISOR

PROF. QIANG YANG, HEAD OF DEPARTMENT

Department of Computer Science and Engineering

21 October 2016

ACKNOWLEDGMENTS

First I would like to thank my family for their unconditional support during the past three years. It is their optimistic attitudes as well as patience that helped me walk across all the troubled water among the years.

I would also like to express my deepest gratitude to my advisor Prof. Pan Hui for his support, guidance and encouragement, without which this thesis would not have been possible.

Special thanks to Jiayu Shu, it was a great pleasure to collaborate with her in the past year and I learned a lot from the collaboration. Same thanks gives to Dr. Tongfeng Weng for the enjoyable collaboration on random walks. Other than that, I would like to thank Haris Mughees and Hamza Zia, for countless times we happily talked about everything and I was always surprised by their knowledge, passion as well as determination.

Thanks also goes to all the members in Symlab family. I am fortunate to know so many wonderful people and work as colleague with them. They have helped me a lot in many aspects from daily life to research. Many thanks to Mr. Issac Ma and Mrs. Connie Lau for their administration work.

Last but not least, I would like to thank Prof. Dit-Yan Yeung and Prof. Chi-Keung Tang. It was great fun to take their courses, which guided me to find my interests and thesis topic. I deeply admire the strong sense of responsibility they have on both lecturing as well as supervision of their students. Their research attitudes and hard working will keep motivating me in my future endeavors.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CARDEA: A CONTEXT-AWARE AND INTERACTIVE VISUAL PRIVACY CONTROL FRAMEWORK

by

RUI ZHENG

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

ABSTRACT

The growing popularity of mobile and wearable devices with builtin cameras, the bright prospect of camera related applications such as augmented reality and lifelogging system, the increased ease of taking and sharing photos, along with advances in computer vision techniques, have greatly facilitated peoples lives in many aspects, but inevitably raised peoples concerns about visual privacy at the same time.

Motivated by the finding that peoples privacy concerns are influenced by the context, in this thesis, we propose Cardea, a context–aware and interactive visual privacy control framework that enforces privacy policies according to peoples privacy preferences. The framework provides people with finegrained visual privacy control using: *i)* personal privacy profiles, with which people can define their context–dependent privacy preferences; *ii)* natural visual indicators: face features, for devices to automatically locate individuals who request privacy protection; *iii)* hand gestures, for people to temporarily update and flexibly inform cameras of their privacy preferences. Benefited

from recent progresses in face and object recognition, Cardea offers a way for context–dependent privacy control in a natural and flexible manner, which differs from tag and marker based systems. We design and implement the framework consisting of Android client app and cloud control server, with convolutional neural networks as core of the image processing module. Our evaluation results confirm such framework is practical and effective, showing promising future for context–aware visual privacy control on mobile and wearable devices.

CHAPTER 1

CONVOLUTIONAL NEURAL NETWORKS

1.1 Deep Learning

Deep learning [1, 2] is part of a broader family of machine learning methods that focus on representation learning. Unlike rule based methods used in early days of artificial intelligence, deep learning targets at tasks that are easy for people to perform but hard for people to describe formally—problems that we solve intuitively. Former methods have proved success in problems that can be completely described by a very brief list of completely formal rules, thus easily provided ahead of time by the programmer. That led to the defeat of world chess champion Garry Kasparov by IBM's Deep Blue chess-playing system in 1997 [3]. However, rule based or knowledge based methods fail at intuitive tasks such as recognizing objects or speech. The reason behind this inability is because these tasks require immense amount of knowledge about the world, and much of this knowledge is subjective and intuitive, thus difficult to articulate in a formal way. The difficulties faced by AI systems relying on hard-coded knowledge suggest that AI systems need the ability to acquire their own knowledge, by extracting patterns from raw data, just like how human learns from experiences. This capability is known as machine learning. It does not take long for people to realize that performance of machine learning algorithms depends heavily on the representation of data fed into these algorithms. Under a good representation, factors of variation can be disentangled and non important factors will be discarded. People used to put lots of efforts on hand designing features to get a good representation for every domain specific problem. Unfortunately seeking a good representation for a problem can be as difficult as solving the problem itself. Deep learning solves this central problem in representation learning by introducing representations that are expressed in terms of other, simpler representations. Deep learning allows the computer to build complex concepts out of simpler concepts, thus achieves great power and flexibility by representing the world as a nested hierarchy of concepts. Fig 1.1 shows a hierarchy relation from AI to deep learning.

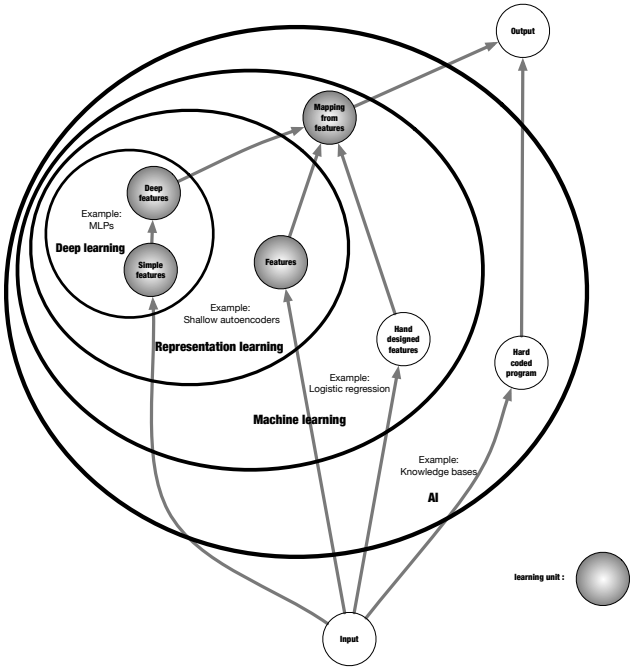


Figure 1.1: Venn diagram showing relations between deep learning, representation learning, machine learning, and AI. An example and work flow are also included for each section.

1.1.1 Three Waves

Deep learning has been rebranded many times under different names, only recently become well known as ”deep learning”. Broadly speaking, there have been three waves of development of deep learning: in 1940s-1960s known as *cybernetics* [4, 5], in 1980s-1990s known as *connectionism* [6], and the current resurgence starts from 2006 [7, 8, 9]. These three waves witness the evolving from simple perceptron, to distributed representation and stochastic gradient descent algorithm, and finally to today's various deep structures. The resurgence of deep learning benefits from the facts: computers are faster, datasets are bigger and a good initialization of model parameters. Faster computers make it possible to train deeper models, bigger datasets relieve deep models from overfitting, and enable them to learn more meaningful mid-level features that are more generalizable, and a good initialization through layer wise pretraining provides a proper prior and makes supervised training on later stages much easier.

Since the early stage of deep learning, neuroscience is regarded as an important source of inspiration. However, it is no longer a predominant guide for the field because we simply do not have

enough information about the brain to use it as a guide. It is seemed as a more general principle of learning multiple levels of composition, which can be applied in machine learning frameworks that are not necessarily neurally inspired.

1.1.2 Breakthroughs

We highlight some of the breakthroughs brought by deep learning in recent years:

- In the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2012, a convolutional neuron network won this challenge for the first time and by a wide margin, bringing down the state-of-the-art top-5 error rate from 26.1% to 15.3% [10], since then the top-5 error rate keeps dropping down each year [11, 12] and in last year dropped to 3.6% using deep residual network (ResNet) [13]. Deep networks also had spectacular successes for pedestrian detection and image segmentation [14, 15, 16] and yield superhuman performance in traffic sign classification [17].

- In March, Google DeepMind's AlphaGo defeated world champion Lee Sedol using deep reinforcement learning [18, 19]. Prior to that, they also showed that a deep reinforcement system is capable of learning to play Atari video games and reaching human-level performance on many tasks [20].

- The application of deep learning on other fields such as speech recognition [21, 22, 23] and machine translation [24, 25] all lead to great successes. The past years surfaced many new trends such as generative adversarial networks, neural Turing machines etc [26, 27, 28, 29, 30]. The years ahead are full of challenges and opportunities to improve deep learning even further and bring it to new frontiers.

1.1.3 Artificial Neural Networks

A Single Neuron

Fig 1.2 shows how to coarsely model a biological neuron: Each neuron receives input signals from its dendrites and produces output signals along its (single) axon. The axon eventually branches out

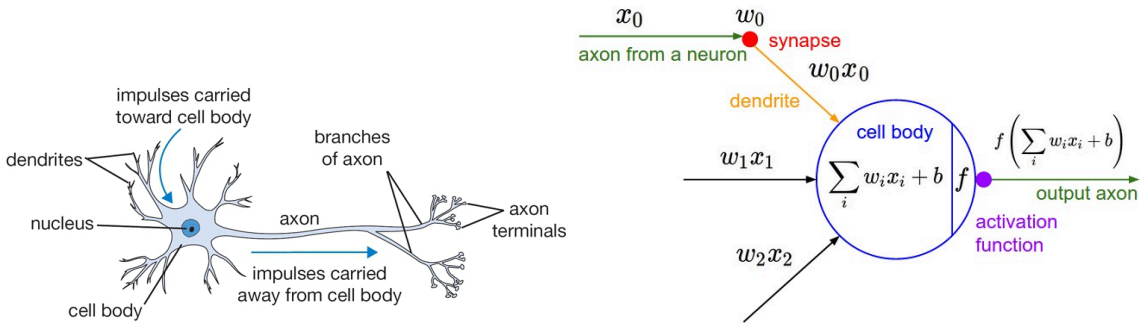


Figure 1.2: A cartoon drawing of a biological neuron (left) and its mathematical model (right). Photo taken from *Module1: Neural Networks* in [31]

and connects via synapses to dendrites of other neurons. In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. In the computational model, we assume that the precise timings of the spikes do not matter, and that only the frequency of the firing communicates information. Based on this rate code interpretation, we model the firing rate of the neuron with an activation function f, which represents the frequency of the spikes along the axon. Historically, a common choice of activation function is the sigmoid function σ, since it takes a real-valued input (the signal strength after the sum) and squashes it to range between 0 and 1. Hereafter in this thesis, we will take standard naming convention and refer a hidden unit as a computational neuron.

Activations

Depends on the activation, neurons can have different properties and impacts on the whole network. Figure 1.3 shows most commonly used activation functions:

Sigmoid Sigmoid is used historically to simulate the firing rate of a neuron but recent years falls out of favor and is rarely used now. It saturates at either tail of 0 or 1, and at these regions the gradient is almost zero, thus it kills gradients from propagating to previous layers, making deep networks not trainable. Another drawback of sigmoid activation is that neurons in later layers will not receive zero-centered inputs, this will cause all positive or negative gradients on the weights during backpropagation, which may lead to undesirable zig-zagging dynamics in the gradient updates of the weights. However, once gradients are added up across a batch

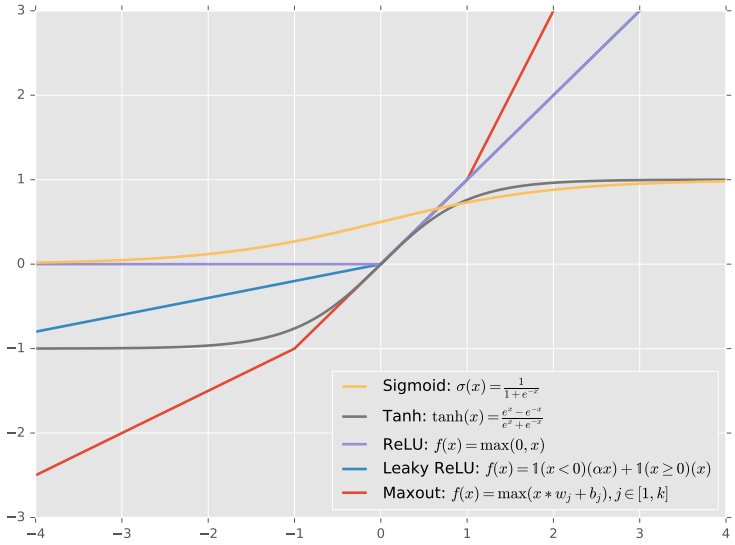


Figure 1.3: Commonly used activation functions.

of data, the final update for the weights can have variable signs, somewhat mitigating this issue.

Tanh Though similar to sigmoid non-linerality in that it also suffers from saturization and thus vanishing gradients problems, tanh is always preferred to sigmoid because its output is zero centered.

ReLU Rectified linear unit becomes popular in recent years. [10] used it in their winning 2012 ImageNet competetion and found that it greatly accelerated the convergence of stochastic gradient descent optimization process. Also comparing to other activations, it is a much more light weight operation since it is a simple thresholding operation. However, ReLU unit can ”die” if a large enough gradient changes the weights such that the neuron never activates on new data.

Leaky ReLU Similar to ReLU, but can help fix the ”dying ReLU” problem by modifying the flat side of ReLU to have a small gradient [32].

Maxout Introduced in [33], Maxout unit enjoys all the benefits of a ReLU unit, and does not have its drawbacks. Maxout activation can implement ReLU activtions and approximate any convex activation function.

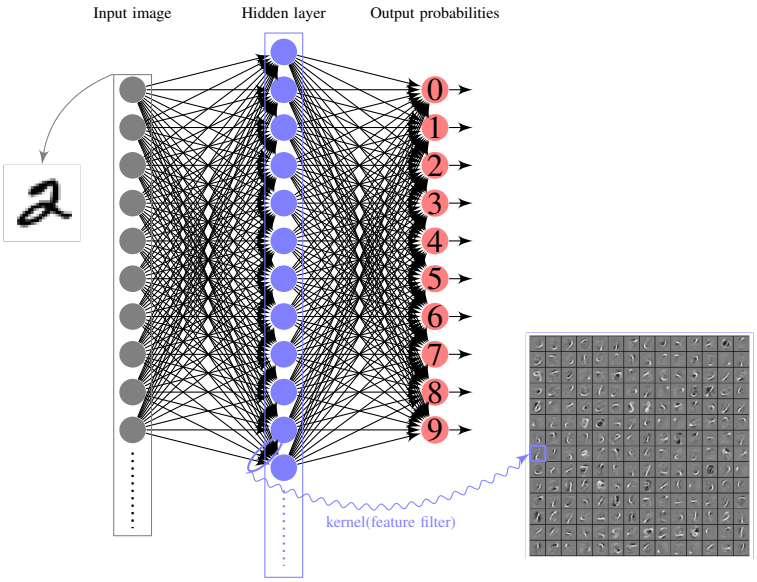


Figure 1.4: A regular feed forward neural network with 1 hidden layer. A 28x28 mnist digit image is flattened and fed into the network, it outputs probabilities of being a certain digit from 0 to 9. The learned kernel for each hidden unit is also shown.

Basic Network Structure

With appropriate loss function, a single hidden unit can function as a linear classifier. However, due to simplicity, its representation power is limited. For example, XOR operation can not be implemented with a single unit, but can be implemented with 3 units. For the network to have enough learning capacity to solve complicate tasks, hidden units are usually aggregated and stacked to form a layer by layer structure. With appropriate regularization, the network can learn powerful features and perform well for many tasks. A regular neural network is shown in Fig 1.4, a deep neural network can have many hidden layers. [34] provides an interactive play ground for tinkering with neural networks.

Backpropagation Algorithm

During the optimization process of loss function, we need to calculate the gradients with respect to all the weights and update them. A naive way is keep all the weights fixed and only tweak one weight to see how it changes loss, thus get its gradient. However, this method is very inefficient since it has to tweak order of the number of weights times to get gradients for all the weights.

The efficient way to do this is backpropagation algorithm. Detailed derivations on backpropagation algorithm can be found on tutorials [35, 31]. We highlight some points here:

- It helps to think of the mapping from input to output that the neural network represents as a computational graph. Each node in this computational graph is simple operation like $+$, $*$, $\max$, or any operation like σ that have a simple derivable gradients. In such way, any complicate mapping function is decomposed into simple unit operations, and backpropagation refers to the node by node backward propagation of gradients from loss to input using chain rule.

- There can be many paths from loss to any variable in the network, gradients flowed through all these paths backwards to this variable get added up when calculating gradient on this variable.

- In neural networks, the preactivations of a layer z and activations of previous layer o have relation $z = w^T o$ where w are the weights between these two layers, it can be seen that gradients for w are proportional to activations o of previous layer, this implies that the scale of the data has an effect on the magnitude of gradients for the weights. When input data is not scaled well, the gradient can be huge, so data preprocessing is a good practice.

Training

Below we list most of the practical concerns when training neural networks:

Data Preprocessing Common data preprocessing steps include mean subtraction, normalization, PCA whitening [36]. Mean subtraction is to zero-center the data. Normalization refers to normalizing the data dimensions so that they are of approximately the same scale. PCA whitening is to first apply PCA on the dataset, followed by a normalization step on the principle components. For most computer vision tasks with big image datasets as input, mean subtraction is enough.

Weight Initialization Mostly used weight initialization strategy is random sampling from a damped gaussian distribution. Improvements on randomly initialization are focused on

calibrating the variances with respect to number of inputs for each neuron such that its output's variable does not blow up. For ReLU units, it is recommended to draw weights from $\sqrt{\frac{2.0}{n}}\mathcal{N}(0,1)$ as suggested in [32]. A recently developed technique called batch normalization explicitly forces each neuron's activations for a minibatch to take on a unit gaussian distribution through whitening among this minibatch [37]. Batch normalization can be interpreted as doing preprocessing at every layer of the network, but integrated into the network itself in a differentiable manner. In practice networks that use batch normalization are significantly more robust to bad initialization.

Regularization $\mathcal{L}2$ regularization $\frac{1}{2}\lambda w^2$ is still the most commonly used regularization term. Other than that, $\mathcal{L}1$ regularization $\lambda|w|$ leads to sparsity in weights, it is useful when the problem is concerned with explicit feature selection. Dropout [38] is a simple but effective way of regularization, in practice it smears activations with a certain probability during training, which can be interpreted as sampling a neural network within the full neural network, and only updating the parameters of the sampled network based on the input data, therefore effectively there are many more neural nets working as an ensemble to eventually perform the classification.

Loss Functions In classification problems, most commonly used data loss functions are hinge loss and cross-entropy loss. Hinge loss is defined as $L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1)$ where L_i and y_i refer to loss and ground truth label of ith sample, f_j is the output score for class j. Hinge loss is used with SVM classifier. Cross-entropy loss is defined as $L_i = -\log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}})$, and used with Softmax classifier. For regression problems, $\mathcal{L}2$ distance $L_i = ||f - y_i||_2^2$ are normally used. Whenever possible, it is recommended to see if a regression problem can be morphed to a classification problem by quantization of outputs into bins, because classification is an easier task and gives confidences of each class.

Optimization There are many optimization options when training deep neural networks. Starting from vanilla SGD (stochastic gradient descent), integration of momentum improves convergence rate and adaptive algorithms ease the pain of tuning the learning rates. [39] gives a comprehensive overview of related algorithms.

Miscellous To find good hyperparameters such as initial learning rate, regularization strength, it is suggested to use random search over grid search [40] and stage the search from coarse to fine. Bagging of neural networks trained independently is a reliable approach to improve the performance.

1.2 Convolutional Neural Networks

Convolution neural network (CNN) is similar to regular neural network, except that it uses weight sharing to largely decrease the number of parameters, thus having the advantage of being less prune to overfitting and also enable training of deeper networks. It borrows the concept of receptive field from biological vision system, in such a way that a neuron will be only connected to a local region of previous layer. Fig 1.5 (left) shows the connections between a 3 channel input image and a neuron in the first hidden layer. Note that the connections are across the channel dimension, which means if a neuron has a receptive field of size $f_x \times f_y$, and its previous layer has D channels, then the connections this neuron has is $f_x \times f_y \times D$. These $f_x \times f_y \times D$ connections is called a kernel or filter, however people usually refer it as a filter with size $f_x \times f_y$ and normally people set $f_x = f_y$. Weight sharing is achieved when apply a convolution operation on previous layer using the same filter. The output from a filter's convolution operation is referred as a feature map, therefore a hidden layer with D channels/feature maps, is generated from D filters' convolution operations. Fig 1.5 (right) illustrates the convolution operations of two 3×3 filters applied on previous layer which has 3 feature maps each having size 5×5, with stride S and padding P both being 1. It can be easily calculated that if a layer has shape $W \times H \times D$, the convolution of K filters of size $f_x = f_y = F$ with stride S and padding P will generate K feature maps as next layer, each feature map has size $((W - F + 2P)/S + 1) \times ((H - F + 2P)/S + 1)$. And each filter will introduce $F \times F \times D$ weights parameters and 1 bias parameter. After convolution operations, it is often followed with nonlinear activations, using the same nonlinearities listed in last section.

Pooling layer is also commonly used periodically between successive convolution layers. Similar to convolution operation, it also requires a spatial extent F and stride S for pooling operation. However, pooling operations does not apply on channel dimension and the operation is very simple:

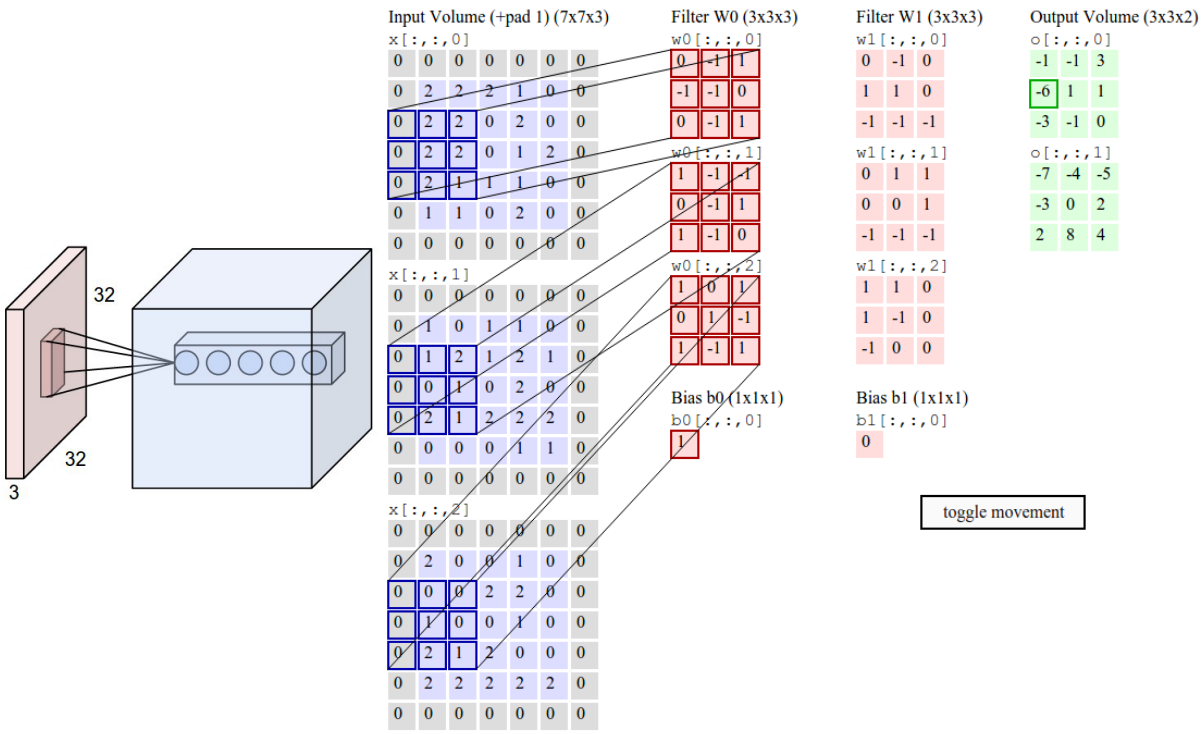


Figure 1.5: Receptive field of a hidden unit (left) and convolution operations of two 3×3 filters (right). Image credit: [31].

max/average pooling is reduce a $F \times F$ block to just a single value via $\max$/$\min$ function. A $F \times F$ pooling on top of a $W \times H$ feature map with stride S will generate a pooled feature map with size $((W-F)/S+1) \times ((H-F)/S+1)$. Though pooling layer can bring certain translation invariance to the network, it will cause too much information loss. It seems likely that future architectures will favor very few to no pooling layers. Local response normalization layer was often used with ReLU neurons but nowadays is also losing favor because in practice its contribution has been shown to be minimal.

The conventional form of a CNN follows the following form:

$$\texttt{Input} \rightarrow [[\texttt{ConvLayer} \rightarrow \texttt{ReLU}] * N \rightarrow \texttt{PoolLayer}?] * M \rightarrow [\texttt{FC} \rightarrow \texttt{ReLU}] * K \rightarrow \texttt{FC}$$

where $*$ indicates repetition, and $\texttt{PoolLayer}?$ indicates an optional pooling layer. Moreover, $0 \leq N \leq 3$, $M \geq 0$ and $0 \leq K < 3$ are common choices. A typical CNN with two feature stages is shown in Fig 1.6. It is preferred to use a stack of convolution layers with small filters rather than one convolution layer with a large filter, because the former allows to express more powerful features of

the input and at the same time with fewer parameters. LeNet [41], AlexNet [10], VGGNet [11] are representatives of this structure. However, the conventional form has been challenged a lot in recent year, many networks that achieve state-of-the-art performance have different repetitive structures such as GoogLeNet [12] and ResNet [13].

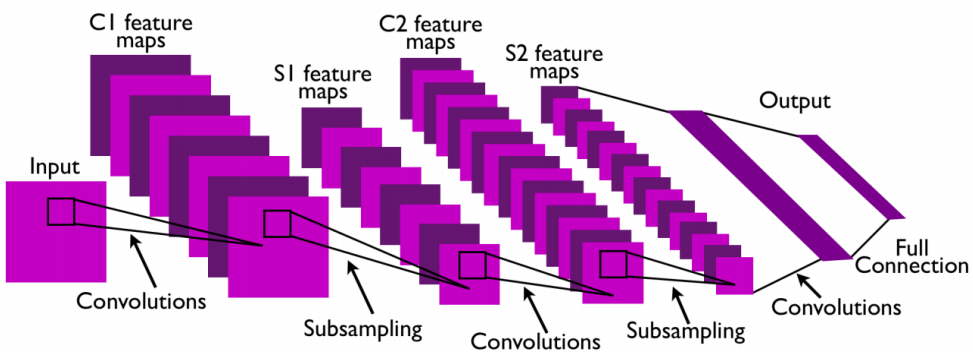


Figure 1.6: A typical CNN architecture with two feature stages.

1.3 Applications

Deep convolutional neural networks have been applied on broad areas from image recognition, video analysis to gameplay. Different from the time in 2012 when it was the major reason behind the winning of ImageNet competetion, nowadays they are basic Lego blocks that will be stacked and tweaked in every possible task. Related applications to this thesis are scene classification, face recognition and object detection. For image classification problems such as scene classification and face recognition, the training dataset is playing an equally important role as the innovation of network structure. Many times conventional CNNs trained on large datasets outperform exquisitely designed CNNs that are trained on smaller datasets by a large margin. Places2 dataset and CASIA-WebFace dataset are among the largest datasets for scene classification and face recognition. The adoption of these classification models is straightforward from their work, therefore we discuss more about how CNNs are used for object detection problems.

From R-CNN to Faster-RCNN

With a powerful CNN like AlexNet that is pre-trained on large dataset, a naive way of using it for object detection problem is classifying a sliding window. However, unlike cascading face detector in OpenCV, the classification of a window involves heavy computations and thus can not afford a sliding window approach. Instead, many methods such as selective search [42], MultiBox [43], EdgeBoxes [44] are proposed to generate region proposals. [45] proposes R-CNN, a region proposal based method that uses region proposals and their convolutional features to train classifiers and bounding-box regressors for object detection, the regional proposals are generated from other methods and features are extracted from a CNN fine-tuned for the detection classes. However, different proposals don't share computations during feature extraction as well as in prediction time, the feature extraction step is slow and requires huge storage for cached features.

He [46] proposed SPP (spatial pyramid pooling) layer to pool convolutional layers to fixed length features, thus enabling variable input size. In forwarding phase, different proposals will be able to share computations, thus it saves time for feature extraction and prediction. However, it still requires first caching features and a post training stage of classifiers and bounding box regressors.

To get rid of the post training stage, Fast R-CNN is proposed in [47] to train the detector in a single stage without caching features and post training steps. The ROI (region of interest) layer proposed in this work is a special case of SPP layer. The network integrates two loss branches which take the roles of classifier and regressor that otherwise would require the post training steps. This has the drawback of still requiring pre-generated proposals, RPN (region proposal networks) is later proposed in Faster R-CNN [48] to generate proposals by the neural network itself, thus get rid of extra proposals from other methods and become a truly end to end training and prediction framework for object detection from raw images. Cardea's gesture detection and recognition module is based on Faster R-CNN.

CHAPTER 2

CARDEA

2.1 System Design

Recalling related works in Chapter 2, what motivates the design of Cardea are the following:

- People's privacy concerns are dependent on context. Although in certain circumstances locations are strong hints of possible privacy intrusion, generally what individuals are doing and with whom are more essential and crucial factors that directly relate to privacy.
- People's privacy preferences vary from each other, thus they should be able to express their personal privacy preferences.
- People's privacy preferences may change from time to time, therefore they need a way to change such preferences easily.

To achieve these objectives, we propose following solution:

- We combine GPS location, grouped scene categories (Table 2.1) and accompanied persons as context that can better represent people's privacy concerns than previous methods. As a result we are able to provide more general as well as finer granularity privacy preference settings for users.
- We use cloud server to host individualized privacy preferences, and user's preference is bound with his facial features. In this way his raw visual information stays locally, preventing the case of visual privacy leakage from hacked servers.
- Other than a simple interface provided to users to update their privacy preferences, hand gestures like  and can be used by user to actively speak out about his preference in the capturing moment, enriching the interaction and adding more flexibilities.

As introduced in chapter 3, breakthroughs made by deep learning community in many computer vision problems such as image classification, face recognition and object detection have guided the proposed solution and shed lights on its practicability. More specifically, given a captured image, Cardea will leverage powerful convolutional neural networks for the recognition of scene context, registered users and gestures. Cardea's design is given in Fig 2.1, it is composed of the client applications and cloud server. It works based on data exchange and collaborative computing involving both client and cloud sides. The major components and interactions include:

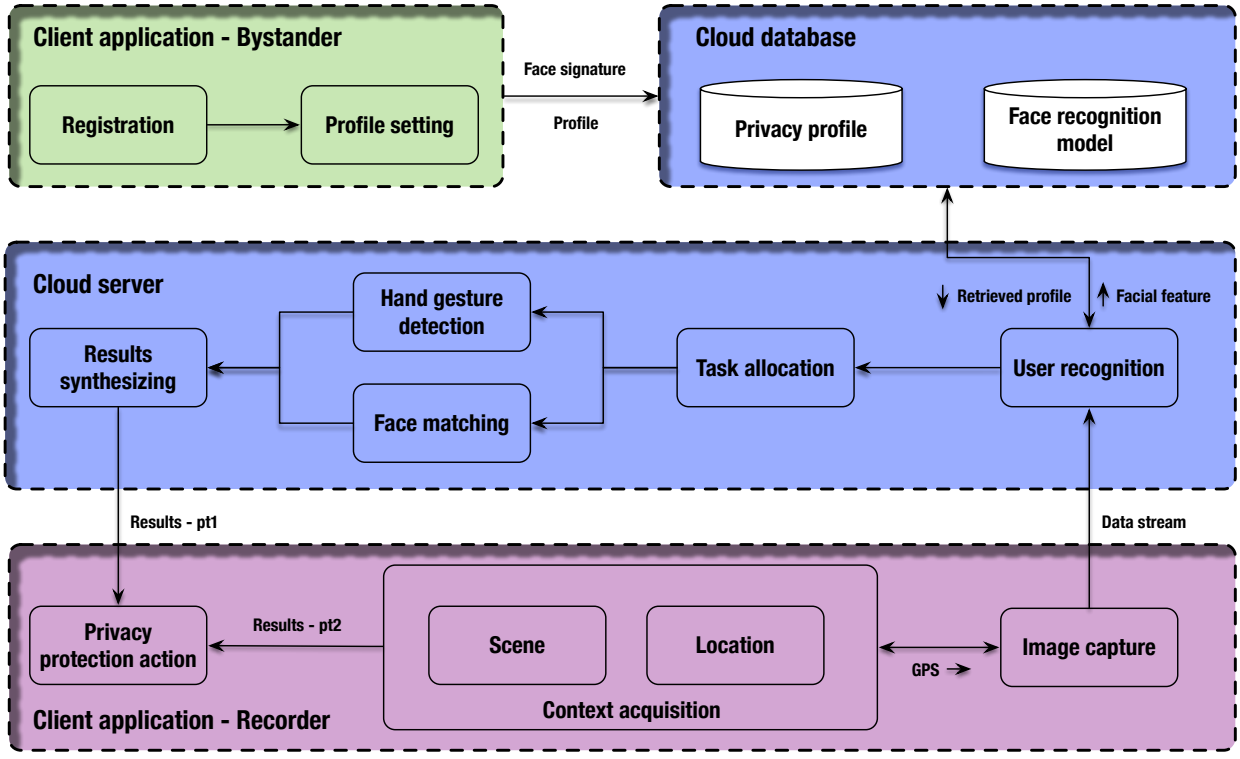


Figure 2.1: System design of Cardea.

Bystander client application: A bystander can use this application to register as Cardea user and define his privacy profile. It will capture a number of face images (about 50–60 images) and extract facial features from these images as his unique face signature. After setting up the context dependant privacy preference, his facial signature and preference will be sent to cloud server for registration and updating of face recognition model.

Recorder client application: A recorder can use this application to take images that will automatically perform privacy protection actions in compliance with all Cardea users' privacy preferences. Given a captured image, it first detects all the faces and extracts the corresponding facial features locally on device, then the features and the captured image (compressed and with all detected faces blurred) are sent to the server for face and gesture recognitions. GPS coordinate is also sent to server in this step to be compared with recognized users' location settings. During the time waiting for response from cloud, it performs scene group prediction task. Finally, predicted scene group and intermediate decision result received from server are combined to decide the actual protection action which will be enforced on the raw captured image.

Cloud server: The cloud server plays two roles: ❶ When receiving requests from Bystander applications, it will store/update users' profiles, and training/updating system's face recognition model automatically; ❷ When receiving requests from Recorder applications, it will initiate face and gesture recognition tasks, as well as partial decision making based on recognition results, and send these intermediate decision results to client for the final step of decision making.

Implementations and evaluation of each module, how Cardea allocates tasks between mobile and cloud, integration and user interactions are discussed in following sections.

2.2 Model Training

2.2.1 Scene Classification

Data Preparing and Preprocessing

For scene classification, we use pre-trained model of Places2 dataset provided by [49]. In the time Cardea project was conducted, Places2 dataset provided by the authors contained 401 categories with more than 8 million training images, and the pre-trained model was based on AlexNet structure [10]. By the time this thesis is writing, the dataset is deprecated and the new Places2 dataset contains 365 categories. And the authors provide more pre-trained models based on different network structures [50].

Table 2.1: Scene categories.

Scene Group	Scene Category
Eating	bistro/indoor, bistro/outdoor, cafeteria, coffee_shop, diner/outdoor, dining_hall, dining_room, fastfood_restaurant, food_court, restaurant, restaurant_patio, sushi_bar
Entertainment	bar, discotheque, pub/indoor
Shopping	bazaar/indoor, bazaar/outdoor, clothing_store, general_store/indoor, jewelry_shop, shoe_shop, shopping_mall/indoor, supermarket
Work	conference_center, conference_room, cubicle/office, library/indoor, office, office_cubicles, reading_room
Public	park, street
Mobility	airplane_cabin, airport_terminal, bus_interior, bus_station/indoor, subway_station/platform, train_interior, train_station/platform
Exhibition	art_gallery, museum/indoor
Religion	cathedral/indoor, cathedral/outdoor, church/indoor, church/outdoor, mosque/outdoor, pulpit, temple/east_asia, temple/south_asia
Illness	hospital, hospital_room, nursing_home
Nudity	bathroom, beach, jacuzzi/indoor, sauna, shower, swimming_pool/indoor, swimming_pool/outdoor

Note that in the dataset we used, there is a non-uniform distribution of images per category for training, ranging from 4,000 to 30,000, mimicking a natural frequency of occurrence of the scene. Among the 401 categories, we choose 59 scene categories that are close to daily life and in such scenes people may have privacy concern. In total this subset composed of 1 million training images and 2950 validation images (50 validation images for each category). We also group these 59 scene categories into 10 groups based on contextual similarity as shown in Table 2.1, such that people have similar reasons for privacy concerns in scenes that are in the same group (e.g. people don't want to be captured in bathroom and beach is both because of nudity concerns). The distribution of training images among categories and groups is shown in Fig 2.2.

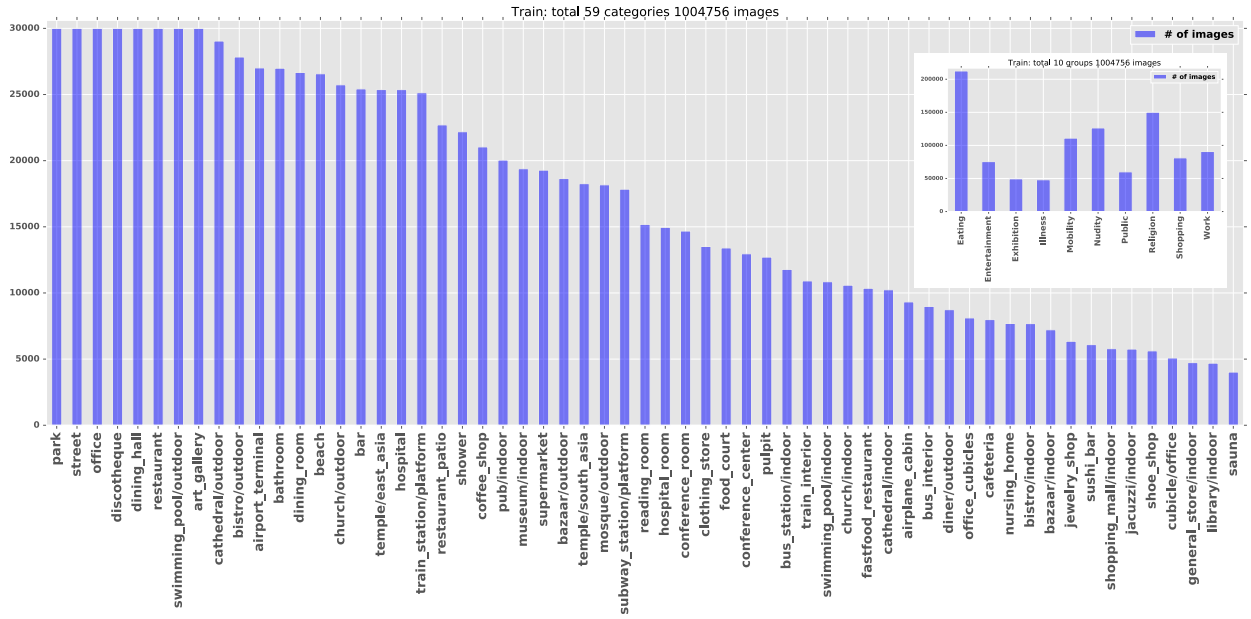


Figure 2.2: Number of images for each category and each group (inset).

Training Procedures

The training step is a standard fine-tuning process, which is extensively used in transfer learning [51, 52]:

❶ Using pre-trained model as feature extractor, we extract the features at *fc7* layer for images belonging to the 59 picked categories. Other than shuffling the features, we also augment the features such that all categories have same amount of features. Though the natural frequencies of occurrence are obviously different among different scenes, we argue that for the purpose of privacy protection, all the scene categories should be equally important, thus categories imbalance is not what we favored. The augmentation step can be implemented using weighted loss layer, but we take simple way of bootstrapping features for categories with less images. After this step, all features are cached and stored in lmdb format.

❷ Train a softmax classifier of the 59 categories using the extracted features. We choose to train a classifier for categories and then add up the output probabilities to predict the group, rather than directly train a group classifier, is because category classifier tells more about the

image, and our desired property is equal weights among scene categories rather than groups.

❸ Both feature extraction and classifier training are implemented using Caffe library [53, 54]. In this step we merge the feature extraction part of pre-trained model and the softmax classifier into a single model by copying weights. Now Caffe has the option of specifying layers with fixed weights, thus simplifying the fine-tuning and deployment process.

Other than improving the validation accuracy from 0.56 to 0.57, shuffling also makes training converges faster. With augmentation to relieve category imbalance issue, the classifier can finally achieve 0.600 validation accuracy on the 59 categories. There is no other benchmarking result specifically on the subset we choose, but recent benchmark gives 53%-56% validation accuracy on the new Places2 dataset with 365 categories [50], suggesting our model is competitive. The higher validation accuracy of our model is due to the smaller scale of classification problem we are dealing with.

Prediction

For prediction, we get probability of a group by summing up the probabilities of all categories belonging to this group, and output the most probable group as prediction of an image. Our model's group prediction accuracy for the validation set is 82.8%. Fig 2.3 shows some prediction examples. As seen from the examples, given an image, the predicted category probabilities are usually distributed to few categories within same group, thus group prediction is resilient to perturbation coming from category prediction. The way we group categories can be seemed as a hard-coded clustering step, which makes prediction more robust to noise. The failure cases are mostly due to natural context ambiguity from a image (e.g. image with object in focus, therefore not enough hints for scene inference). Labeling the 342 non selected categories as an extra group will amplify the ambiguity issue, as doing so will distribute probabilities to the extra group and lead to wrong prediction, even for images with less ambiguity. In other words, a 59 way classifier leads to higher recall for selected scenes and grouping leads to higher accuracy. This is also reflected in confusion matrices shown in Fig 2.4, category confusion matrix shows some clustering structure which is in accordance with the groups we manually assigned. However, only using top 1 category for predic-

Figure 2.3: Scene classification prediction example. Top5 groups (green) and categories (white) are shown with their probabilities.

tion sacrifices prediction accuracy, which can be avoided by grouping as shown in group confusion matrix.

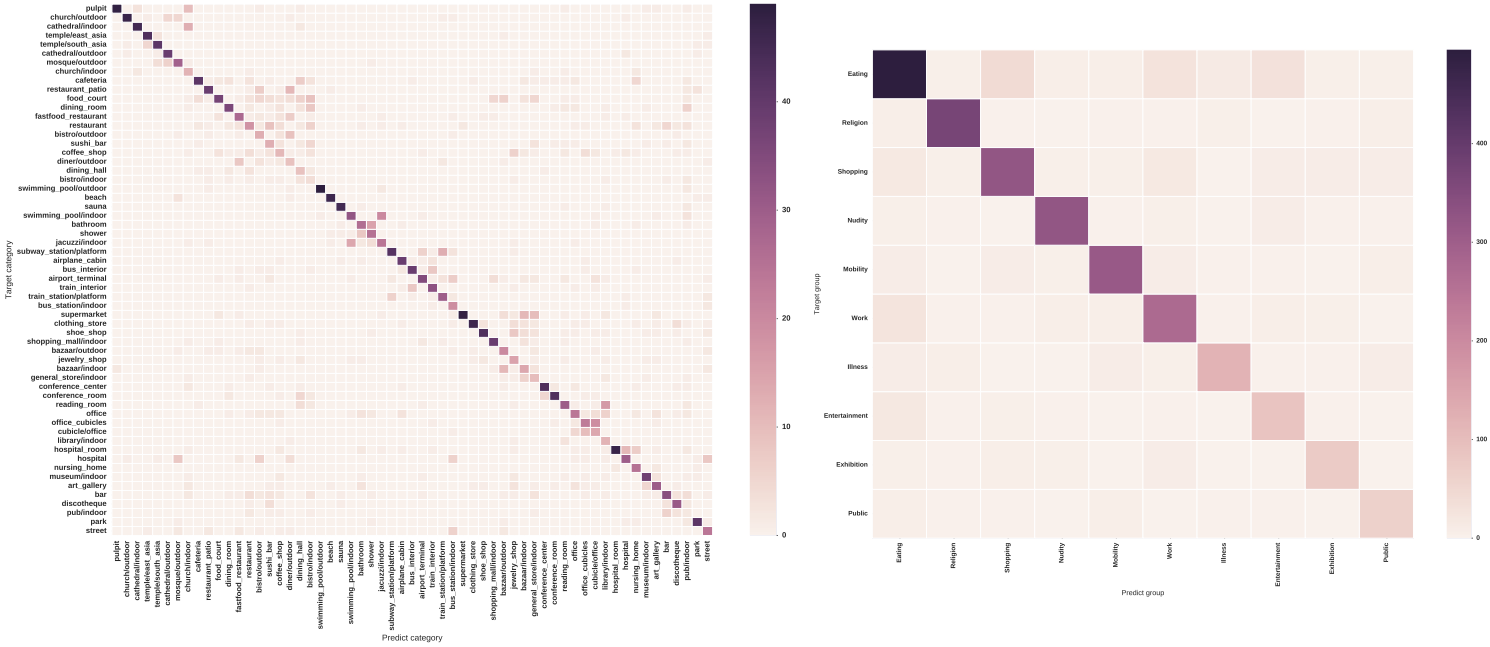


Figure 2.4: Confusion matrices for category prediction (left) and group prediction (right).

2.2.2 Face Recognition

Like scene classification, we select a pre-trained model for face recognition task. More specifically, the pre-trained model will serve as face feature extractor, and we will update the face classifier whenever new users register in Cardea and upload their face features. There are already many deep neural networks deployed in commercial products, like Megvii's Face++ [55], Facebook's Deepface [56], Google's Facenet [57], Sensetime's Deepid [58]. OpenFace [59] is an open source project that is gaining attentions in recent months, it is based on Torch [60]. Because Cardea's other modules are under Caffe framework, we limit our options on open sourced Caffe models. The models in our consideration are VGG face recognition model [61] and Lightened CNN face recognition model [62].

To compare performance of features extracted from the two models, we run t-SNE visualization [63] on the features of a small dataset we previously collected for emotion sensing. Fig 2.5 shows the t-SNE visualization result. It seems VGG feature and Lightened CNN feature have similar performance, at least on this small dataset. Though it is found that comparing to Lightened CNN model, VGG model is more robust to variations and its features show better transferability [64], the

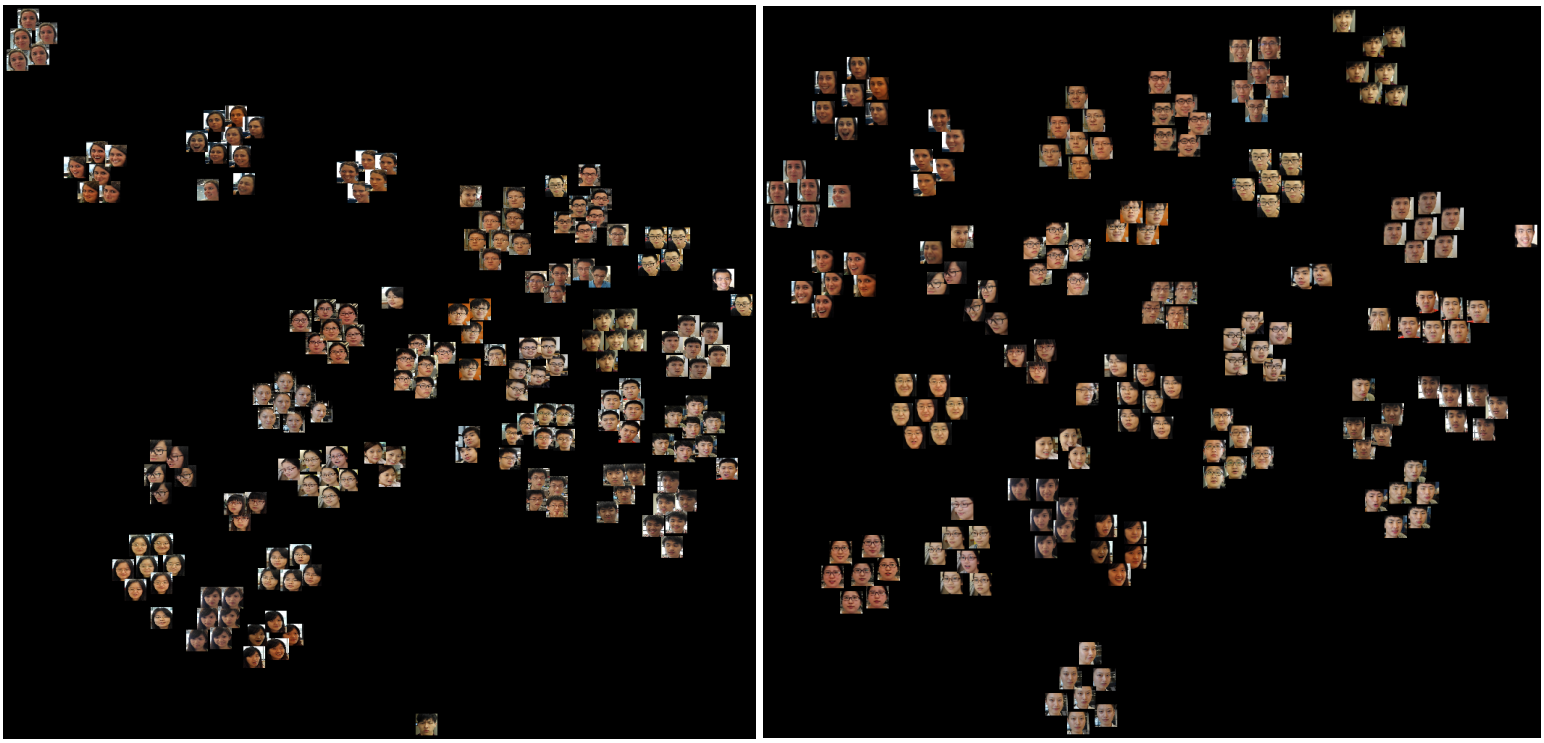


Figure 2.5: t-SNE visualization of VGG *fc8* layer features and Lightened CNN *fc1* layer features.

model size is more than 500MB, 10 times bigger than Lightened CNN model. And the released VGG model has a feature dimension of 4096, while Lightened CNN model has a feature dimension of 256. Our experiment on different Android smartphones shows it takes 10 times longer to extract VGG features. Table 2.2 shows the forwarding time we tested on different smartphones. It can be seen VGG model consumes much more memory that it can only run on phones with memory larger than 3GB. Due to above concerns, we use Lightened CNN model in our implementation.

Table 2.2: Time of single facial feature extraction and batch facial feature extraction (10 faces).

	Xiaomi Mi 3W Snapdragon 800 2GB RAM	Galaxy Note 4 Snapdragon 805 3GB RAM	Xiaomi Mi 5 Snapdragon 820 4GB RAM
1 VGG CNN	N/A	N/A	$\sim$ 2780 ms
10 VGG CNN	N/A	N/A	$\sim$ 26740 ms
1 Lightened CNN	$\sim$ 508 ms	$\sim$ 330 ms	$\sim$ 303 ms
10 Lightened CNN	$\sim$ 6602 ms	$\sim$ 3971 ms	$\sim$ 2031 ms

Detection and Alignment

Lightened CNN model takes aligned face as input, requiring that the distance between midpoint of eyes and midpoint of mouth is 48, and y value of midpoint of eyes is 40, as shown in Fig 2.6. We use OpenCV's haar cascade [65, 66] frontal face detector. The limitation it brings to Cardea is only frontal faces will be detected and recognized. We set *minNeighbors* (the parameter specifying how many neighbors each candidate rectangle should have to retain it) to be 3 to ensure a relative high recall for face detection. To remove false positive, we further apply skin color filter (range $[0, 48, 60] - [30, 255, 255]$ in HSV color space) on retained rectangles. Following that, we use Dlib library's HOG [67, 68] based face detector as a second stage filter. Note that Dlib's face detector has higher accuracy comparing to OpenCV's face detector, but is much slower if applied directly on a high resolution image, therefore it is used as a filter on small rectangular areas. Dlib's facial landmarks detector [69] is also used in later face alignment stage, it can detect 68 facial landmarks [70, 71]. With the detected landmarks and required alignment condition about inputs to the CNN model, we can calculate the homography matrix that is finally used to align faces. The steps for detection and alignment is shown in Fig 2.6, and we implemented it as a JNI library for Android platform [72].

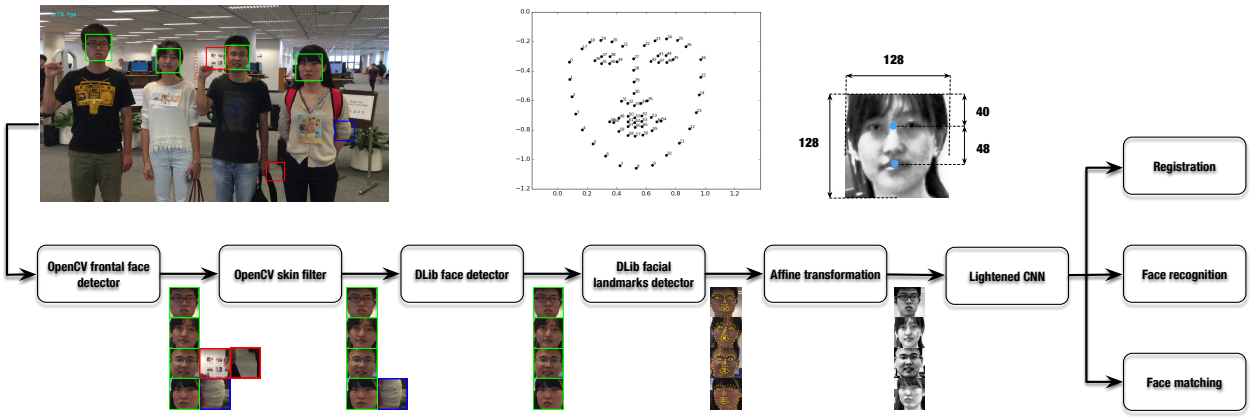


Figure 2.6: Face detection and alignment workflow.

Recognition

All the facial features uploaded by registered users are used to train a classifier in the cloud server, using LIBSVM library [73]. During prediction, we enable probability estimations $p_i, i \in 1, \cdots, N$, where p_i is the probability of being user i. For each facial feature, if $\max_i p_i \leq T_p$, then we treat it as from an unknown person who hasn't registered in Cardea, otherwise it is from the user who has the highest probability and his privacy preference will be fetched for further processing. Threshold T_p is an empirical parameter, a proper value of T_p makes sure registered users are recognized correctly, and non-registered bystanders are recognized as unknown person. It is dependant on the face database scale, but the proper value can always be found through cross validation.

Matching

Face matching occurs when a recognized user A has also specified and uploaded features of person B with whom he doesn't want to be captured, it is to determine whether B also appears in this captured image. Note that B is not necessarily a registered user of Cardea. It is required that n_B, the number of B's facial features uploaded by A should be more than 10. A simple way is pointing the camera on B's photo but from different angles. Then for every detected face C other than A in the image, its feature f_C will be compared with n_B features of B uploaded by A. `Cosine` similarity is used as distance metric. Among n_B distances between f_C and B's features, we can calculate the ratio r of distances which are shorter than a threshold T_d, if the ratio r is higher than a threshold T_r, then C and B are the same person, thus B appears with A in the same image and A's privacy will be protected. By tuning, we find $T_d \in (0.3, 0.5)$ and $T_r \in (0.5, 0.8)$ shows good enough performance. In Fig 2.7, we plot the distribution of distance between same person's Lightened CNN features and different person's Lightened CNN features. The features are extracted from all the faces in ORL face database [74], which consists of 400 images from 40 distinct subjects, 10 images per subject. Each subject has photos with different variations, such as: with/without glasses, open/closed eyes, and different facial expressions. It is obviously seen that distances between features of same person and features of different persons are well seprated, especially for the case of `cosine` similarity.

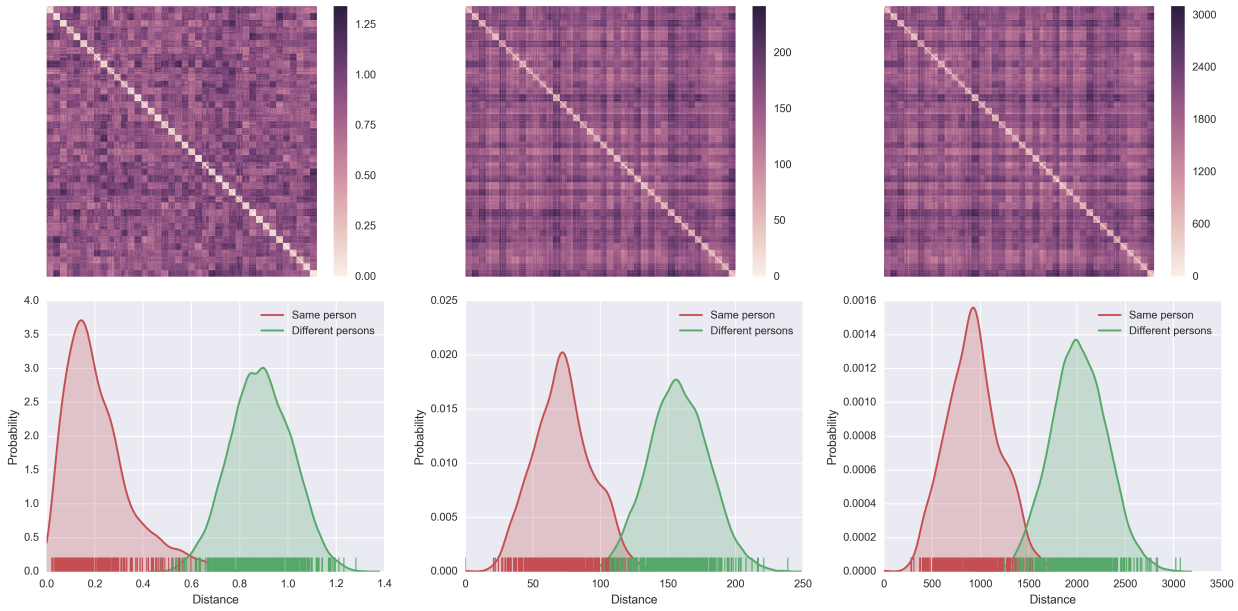


Figure 2.7: Distance matrix (top) and distance distribution (bottom) of Lightened CNN features using `cosine` similarity (left), l_1 norm (center) and l_2 norm (right).

2.2.3 Gesture Recognition

In Cardea, "yes" (✌) and "no" (✋) gestures have the highest priorities and are used to temporarily overwrite privacy preferences. To recognize gestures in real world images, the first step is detection of hands, and it turns out to be the most challenging part in this sub task. Skin color based hand detector will fail dramatically in images with cluttered background. A much more robust method will be using multiple proposals [75] based on hand shape, context, and skin color. However, it takes an extremely long time to detect hands in one image. Finally, we choose to use state-of-the-art detection framework faster R-CNN [48] to train a gesture detector in an end-to-end manner.

Data Preparing and Preprocessing

VGG group has shared a comprehensive dataset of hand images collected from various different public image data set sources in [75, 76]. It contains 5628 images, which is composed of 4069 training images, 738 validation images, 821 testing images respectively, each image is with annotations of hand bounding boxes. However, this dataset can only let us train a hand detector. To

achieve the goal of recognizing gestures, there are two solutions in our consideration:

- First train a hand detector using this dataset, then train another hand gesture classifier using other commonly used gesture datasets and pipe them together.
- Take this dataset as a subset of images with "natural" gestures, then prepare extra images with "yes" and "no" gestures including annotations by ourselves, and train a "natural/yes/no" gesture detector end-to-end.

The first solution is not an end-to-end solution, and the specific "yes" and "no" gestures may not be included in those standard gesture datasets, then we will still need to prepare our specific gesture dataset like in second solution. Therefore, we choose the second solution, based on the observation and also assumption that annotated hands in VGG's hand dataset are in natural relaxing modes, thus will not be treated as "yes" or "no" gestures. The annotations of VGG dataset are tilted rectangles shown as yellow ones in Fig 2.8, we re-annotate the dataset using bounding boxes of the original annotations shown as blue rectangles.

We crawled 527 images with "yes" hand gestures, and 363 images with"no" hand gestures. Note that in a crawled image, it may contain different types of hand gestures as shown in Fig 2.8, which is not a problem so long as gesture types are annotated correctly (*remind* that all gestures in VGG dataset are treated as "natural" class). These images are crawled from Google and Flickr image search with keywords such as "victory sign", "stop gesture", "palm gesture" and so on, many of them are focused on the hands thus don't contain many background pixels. We rescale these images in different scales and then pad zeros on rescaled images. In Faster-RCNN python implementation [77], an input image is rescaled to around 1000×1000 before fed to region proposal network. If without padding data augmentation step, the bounding boxes of hand gestures will be huge in many crawled images that are focused on hands, which makes the learned model not able to detect small hand gestures and also not perform well on the regression of large hand gestures. Another reason for the padding step is to counter data imbalance of three classes. After augmentation, we have a dataset of 13843 images, including 5628 images from VGG dataset, 4712 augmented images mostly with "yes" gestures and 3503 augmented images mostly with "no" gestures. Fig 2.8

shows some sample images with annotations from this composed dataset. We wrote a tool [78] to annotate the crawled images.

Figure 2.8: Training hand gesture dataset composed of VGG hand dataset (top) and augmented crawled dataset (middle and bottom). Blue, red and green annotations denotes "natural", "no" and "yes" gestures.

Training Procedures

Using the composed dataset, we fine-tune the *conv3_1* and up layers of VGG16 pre-trained model provided by Faster-RCNN library, jointly with region proposal layers and detection layers that are not part of VGG16 pre-trained model. Features from *conv5_3* layer of VGG16 network are shared

between region proposal network (RPN) and Fast-RCNN [47] detection network, RPN uses them to generate proposals, and region of interest (ROI) pooling layer in detecting network uses them for bounding box regression and classification. There are two methods to train a shared feature extraction network. One is an alternating optimization method with following steps: ❶ Train an RPN M_1 initialized from VGG16 pre-trained model M_0, ❷ Generate training proposals P_1 using RPN M_1, ❸ Train Fast R-CNN model M_2 on proposals P_1 initialized from M_0, ❹ Train RPN M_3 from M_2 without changing convolutional layers, ❺ Generating proposals P_2 using RPN M_3, ❻ Train Fast R-CNN model M_4 on proposals P_2 initialized from M_3 without changing convolutional layers, ❼ Add M_3's RPN layers to Fast R-CNN model M_4. Another method is an approximate joint optimization method by training with stochastic gradient descent as usual, which is easier, faster and achieves similar performance [77], so we use the second training procedure.

Prediction

During prediction, we set Non-Maximum Suppression (NMS) threshold as 0.4 and confidence level threshold as 0.7. Figure 2.9 shows some examples of gesture detection and recognition results in natural environment. It can be seen that the trained model can handle cluttered background such as in shopping environment, and indoor dark lighting condition. It is interesting to notice that bounding boxes for "natural" hands are bigger, reflecting the fact that we select re-annotated the VGG dataset for "natural" class using bounding boxes of the original annotations. The model has a good recall in terms of hand detection, however, its gesture recognition is sensitive to motion blur, palm angles and gesture size. We think the good recall of hands comes from the comprehensive VGG dataset, and the not so good recognition result is because the "yes/no" dataset we composed does not have a good quality because gestures are focused in many images, especially for "no" gestures, which is reflected by the observation that "yes" gesture recognition performs better than "no" gesture.

Figure 2.9: Examples of gesture detection and recognition.

2.3 System Integration

2.3.1 Deployment on Android

Ideally, for a privacy control framework, we prefer a design that does not require cloud server and all the algorithms run locally on mobile devices. In the current design and implementation, cloud server exists mainly for two reasons: ❶ Storage center for profiles and hosting face recognition model; ❷ RPN in gesture recognition task is written in Python language, thus gesture recognition can not run on Android smartphones easily. However, these are not hard restrictions, possible improvements are discussed in Chapter 5.

The deployment of Caffe models for scene classification and facial feature extraction on smartphones is based on Caffe-android-library [79], we modified its code [80] to support loading multiple Caffe models, batch feature extraction, and only forwarding to a specified layer during feature extraction, which saves useless computations from fully connected layers. There are other libraries for deploying neural networks on mobile, such as Torch-android [81] and MXNet [82]. The de-

ployed scene classification model (based on AlexNet structure) has a size of 230MB, which is not small. However, its prediction is very fast, and can be easily fitted into the time slot when client is waiting response from cloud server. The facial feature extraction model (based on Lightened CNN structure) has a relatively bearable size of 33MB. Comparing to AlexNet, Lightened CNN model has smaller filter sizes, but with many more feature maps, therefore in run time, Lightened CNN model consumes about 1GB memory, which makes it not able to run on smartphones with less than 2GB memory as shown in Table 2.2. Possible ways to decrease model size and optimization of resource consumptions are also discussed in next chapter. Other lighter models we deployed on android include OpenCV face cascading model (less than 1MB) for face detection, and Dlib shape predictor (90MB) for facial landmarks detection.

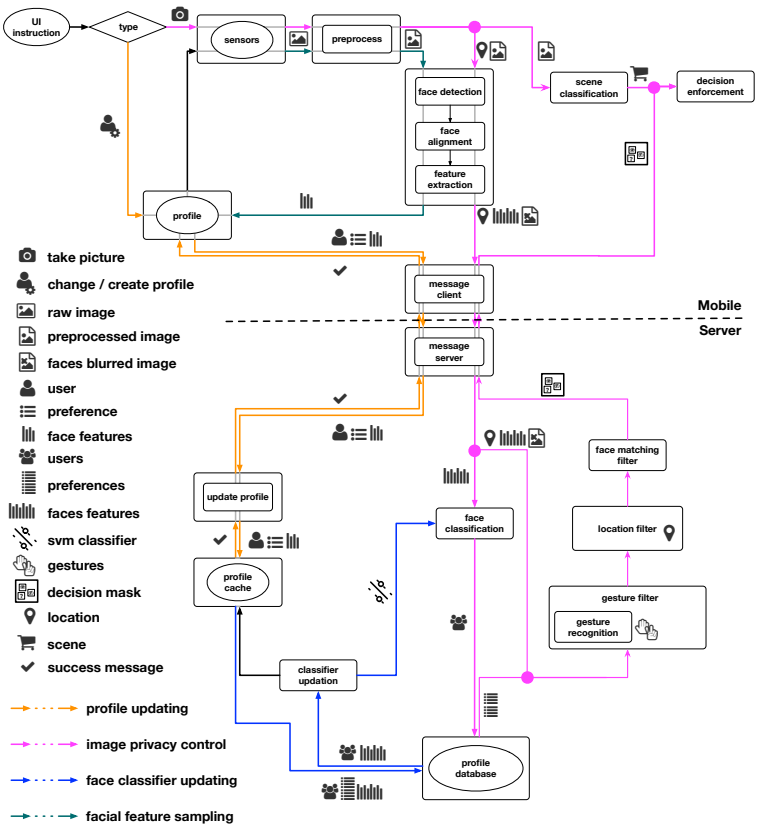
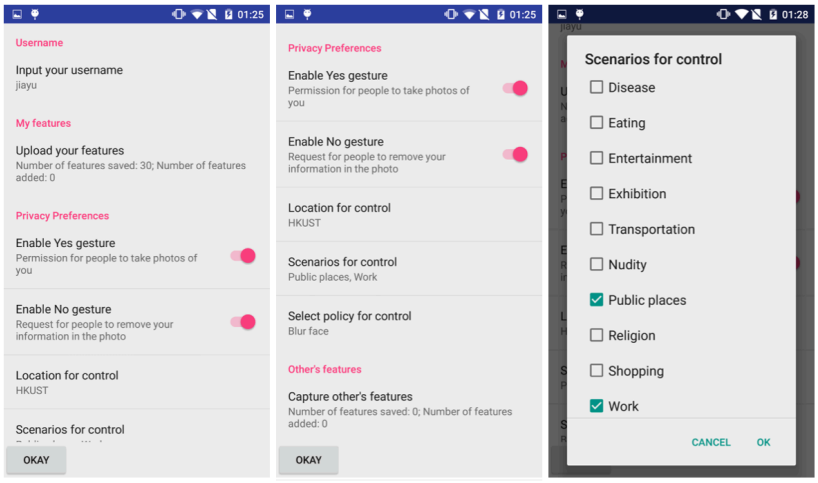


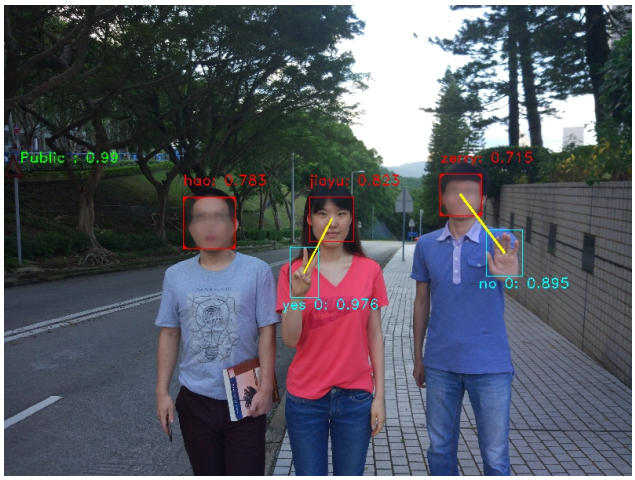
Figure 2.10: Dataflow of Cardea

2.3.2 Dataflow and Integration

Fig 2.10 shows the detailed structure and dataflow of Cardea, which is mainly composed of the following steps (a demo video about the usage can be found in [83]):



(a) Registration and profile updating interface



(b) Privacy protection example

Figure 2.11: Cardea user interface and privacy protection results. In (a), *jiayu* registers as a Cardea user by extracting and uploading her face features. She specifies HKUST, two scene groups for privacy protection. She also enables "yes" and "no" gestures. In (b), a picture is taken in HKUST. 3 registered users, one "yes" and one "no" gesture are recognized. The scene is correctly predicted as "Public". *jiayu*'s face is not blurred due to her "yes" gesture. Prediction probabilities are also shown in (b).

Registration and Profile updating: The interface provided to a bystander for registration and updating of his profile is shown in Fig 2.11a. He is able to select one or more scene categories, one location for control, as well as enable gestures or not. In two cases his facial features will be packaged with his privacy preferences: one is in registration time and the other is when he wants to update his facial features in the cloud. This registration and updating message will be send to server, if it is an updating message without feature updating, then his user profile in cloud will be updated immediately and he will receive a "success" notification, otherwise this message will be first buffered in a profile cache, and only after the next successful face classifier updating will he receive the "success" notification.

Face classifier updating: In the server, the face classifier will be updated intermittently. For every time interval ΔT, if there is cached messages that brings new facial features, it will ❶ block the

queue of prediction messages from doing face recognition, ❷ merge profile cache with profile database, ❸ retrain a face classifer and ❹ unblock queued prediction messages.

Image capturing: When a recorder uses Cardea to take a image, extracted facial features, GPS coordinate as well as face-blurred image will be packaged as prediction message and sent to server for processing. In the server side, after faces recognition and profiles retrieval, it will start the cloud part of decision making process for every face bounding box, after which what will be returned to the client side is a decision mask that specifies among all the detected faces, which should be blurred, which should be kept and which should be further determined based on the scene results calculated on the client side. In client side, while waiting for response from server it will calculate the scene result. With received decision mask, it makes final decisions on every face and enforces the privacy protection actions complied with everyone's preference.

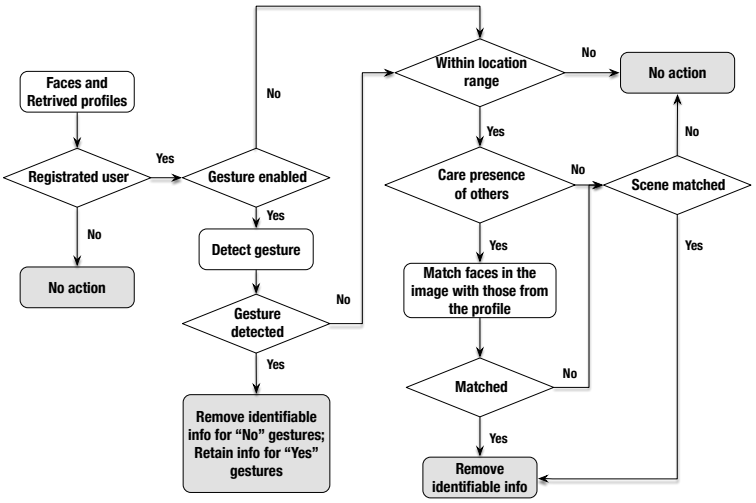


Figure 2.12: Steps of decisions about actions to be applied on detected faces.

Decision making: Fig 2.12 gives the detailed decision steps. Note that in this process, we need to match a detected gesture with the right face or people who issued the gesture, currently we simply take the nearest face of a detected gesture as the person who issued this gesture, thus it requires the user to put his hand near his face when sending "yes/no" gestures. As seen from the figure, when the detected face is recognized as a registered user, and context including location, scene, presence of other people and gesture matched with the user's profile, this detected face will be removed. An

example is shown in Fig 2.11b. For other cases, the face will be kept but it may cause removal of other faces in the face matching step.

Concurrent requests: To enable concurrent requests from different client apps, we implement multiple "recognition" workers to process the queued messages from all the clients, recognition results from different workers will be collected and put into a result queue, followed by multiple "mailing" workers to make decision masks and mail the decision masks back to corresponding blocked client threads. We tested with a server configuration - "Intel i7-5820K CPU, 16GB RAM, GeForce 980Ti Graphic Card(6GB RAM)" and the campus's Wi-Fi network, the time from start sending message to receive server's response is $1-2s$, depending on the message type. The total time from capturing moment to the enforcement of protection actions is $2-4s$, depending on how many faces detected. Most time is spent on facial feature extraction and message data transmission. Note that processing in the server side is relatively fast, gesture recognition of one image takes $200-300ms$, while the time spent on SVM face classifier and face matching is negligible. With 6GB GPU RAM, the server can serve 3 gesture recognition workers at the same time, supporting 5-10 concurrent requests. The implementation of Cardea is hosted in [84].

2.4 Evaluation

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[2] Ian Goodfellow Yoshua Bengio and Aaron Courville. "Deep Learning". Book in preparation for MIT Press. 2016. URL: `http://www.deeplearningbook.org`.

[3] Feng-Hsiung Hsu. *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press, 2002.

[4] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[5] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[6] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. DTIC Document, 1985.

[7] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Comput.* 18.7 (July 2006), pp. 1527–1554.

[8] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507.

[9] Yoshua Bengio et al. "Greedy layer-wise training of deep networks". In: *NIPS*. 2007.

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

[11] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014).

[12] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[13] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015).

[14] Pierre Sermanet et al. “Pedestrian detection with unsupervised multi-stage feature learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 3626–3633.

[15] Clement Farabet et al. “Learning hierarchical features for scene labeling”. In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1915–1929.

[16] Camille Couprie et al. “Indoor semantic segmentation using depth information”. In: *arXiv preprint arXiv:1301.3572* (2013).

[17] Dan CireAn et al. “Multi-column deep neural network for traffic sign classification”. In: *Neural Networks* 32 (2012), pp. 333–338.

[18] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.

[19] *AlphaGo versus Lee Sedol*. URL: `http://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol`.

[20] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[21] George Dahl, Abdel-rahman Mohamed, Geoffrey E Hinton, et al. “Phone recognition with the mean-covariance restricted Boltzmann machine”. In: *Advances in neural information processing systems*. 2010, pp. 469–477.

[22] Li Deng et al. “Binary coding of speech spectrograms using a deep auto-encoder.” In: Citeseer. 2010.

[23] Geoffrey Hinton et al. “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97.

[24] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[25] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[26] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 2672–2680.

[27] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[28] Alex Graves, Greg Wayne, and Ivo Danihelka. “Neural turing machines”. In: *arXiv preprint arXiv:1410.5401* (2014).

[29] Kelvin Xu et al. “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. In: *Proceedings of The 32nd International Conference on Machine Learning*. 2015, pp. 2048–2057.

[30] Chris Olah and Shan Carter. *Attention and Augmented Recurrent Neural Networks*. 2016. URL: http://distill.pub/2016/augmented-rnns/.

[31] *CS231n: Convolutional Neural Networks for Visual Recognition.* 2015. URL: http://cs231n.github.io/.

[32] Kaiming He et al. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1026–1034.

[33] Ian J Goodfellow et al. “Maxout networks.” In: *ICML (3)* 28 (2013), pp. 1319–1327.

[34] *Neural Network Playground*. URL: http://playground.tensorflow.org/.

[35] Michael Nielson. *Neural Networks and Deep Learning*. 2016. URL: http://neuralnetworksanddee
com/index.html.

[36] *PCA whitening*. URL: http://ufldl.stanford.edu/tutorial/unsupervised/PCAWhitening/.

[37] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[38] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting.” In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[39] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. URL: http://ufldl.stanford.edu/tutorial/unsupervised/PCAWhitening/.

[40] James Bergstra and Yoshua Bengio. “Random search for hyper-parameter optimization”. In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[41] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[42] Jasper RR Uijlings et al. “Selective search for object recognition”. In: *International journal of computer vision* 104.2 (2013), pp. 154–171.

[43] Dumitru Erhan et al. “Scalable object detection using deep neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 2147–2154.

[44] C Lawrence Zitnick and Piotr Dollr. “Edge boxes: Locating object proposals from edges”. In: *European Conference on Computer Vision*. Springer. 2014, pp. 391–405.

[45] Ross Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[46] Kaiming He et al. “Spatial pyramid pooling in deep convolutional networks for visual recognition”. In: *European Conference on Computer Vision*. Springer. 2014, pp. 346–361.

[47] Ross Girshick. “Fast r-cnn”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1440–1448.

[48] Shaoqing Ren et al. “Faster R-CNN: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[49] *Places2 Dataset Project*. URL: `http://places2.csail.mit.edu/index.html`.

[50] *Release of Places365-CNN*. URL: `http://github.com/metalbubble/places365`.

[51] Ali Sharif Razavian et al. “CNN features off-the-shelf: an astounding baseline for recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014, pp. 806–813.

[52] Jason Yosinski et al. “How transferable are features in deep neural networks?” In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[53] *Caffe*. URL: `http://caffe.berkeleyvision.org/`.

[54] Yangqing Jia et al. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: *arXiv preprint arXiv:1408.5093* (2014).

[55] Erjin Zhou et al. “Extensive facial landmark localization with coarse-to-fine convolutional network cascade”. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2013, pp. 386–391.

[56] Yaniv Taigman et al. “Deepface: Closing the gap to human-level performance in face verification”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1701–1708.

[57] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “Facenet: A unified embedding for face recognition and clustering”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 815–823.

[58] Yi Sun et al. “Deepid3: Face recognition with very deep neural networks”. In: *arXiv preprint arXiv:1502.00873* (2015).

[59] Brandon Amos, Bartosz Ludwiczuk, and Mahadev Satyanarayanan. *OpenFace: A general-purpose face recognition library with mobile applications*. Tech. rep. CMU-CS-16-118, CMU School of Computer Science, 2016.

[60] *Torch7*. URL: `http://torch.ch`.

[61] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. “Deep face recognition”. In:

[62] Xiang Wu, Ran He, and Zhenan Sun. “A Lightened CNN for Deep Face Representation”. In: *arXiv preprint arXiv:1511.02683* (2015).

[63] Laurens van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.

[64] Mostafa Mehdipour Ghazi and Hazim Kemal Ekenel. “A Comprehensive Analysis of Deep Learning Based Representation for Face Recognition”. In: *arXiv preprint arXiv:1606.02894* (2016).

[65] *OpenCV*. URL: `http://opencv.org`.

[66] Paul Viola and Michael Jones. “Rapid object detection using a boosted cascade of simple features”. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001, pp. I–511.

[67] *Dlib C++ Library*. URL: `http://dlib.net`.

[68] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 1. IEEE. 2005, pp. 886–893.

[69] *Real-Time Face Pose Estimation*. URL: `http://blog.dlib.net/2014/08/real-time-face-pose-estimation.html`.

[70] *Dlib facial landmarks*. URL: `http://openface-api.readthedocs.io/en/latest/openface.html`.

[71] *Dlib facial landmark coordinates*. URL: `http://openface-api.readthedocs.io/en/latest/_modules/openface/align_dlib.html`.

[72] *Face alignment JNI library*. URL: http://github.com/ZhengRui/FaceAlignmentJNI.

[73] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: a library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 2.3 (2011), p. 27.

[74] *ORL face database*. URL: http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html.

[75] Arpit Mittal, Andrew Zisserman, and Philip HS Torr. "Hand detection using multiple proposals." In: Citeseer.

[76] *VGG: Hand dataset*. URL: http://www.robots.ox.ac.uk/~vgg/data/hands/index.html.

[77] *Faster-RCNN (Python Implementation)*. URL: http://github.com/rbgirshick/py-faster-rcnn.

[78] *Image annotation tool*. URL: http://github.com/ZhengRui/ImgAnnotaPyQt4.

[79] *Caffe Android Library*. URL: http://github.com/sh1r0/caffe-android-lib.

[80] *Caffe Android Library (Clone)*. URL: http://github.com/ZhengRui/caffe-android-lib.

[81] *Torch-7 for Android*. URL: https://github.com/soumith/torch-android.

[82] *MXNet on Mobile Device*. URL: http://mxnet.readthedocs.io/en/latest/how_to/smart_device.html.

[83] *Cardea demo video*. URL: http://drive.google.com/file/d/0B4z8qjK8O_uUc0o2RjZWYktiMTg/view.

[84] *Cardea project*. URL: https://github.com/ZhengRui/cardea.