Deep Feature-based Face Detection on Mobile Devices

Sayantan Sarkar[1], Vishal M. Patel[2] and Rama Chellappa[1]
[1] Center for Automation Research, University of Maryland, College Park, MD 20742
{ssarkar2, rama}@umiacs.umd.edu
[2] Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ 08854
vishal.m.patel@rutgers.edu

Abstract

We propose a deep feature-based face detector for mobile devices to detect user's face acquired by the front-facing camera. The proposed method is able to detect faces in images containing extreme pose and illumination variations as well as partial faces. The main challenge in developing deep feature-based algorithms for mobile devices is the constrained nature of the mobile platform and the non-availability of CUDA enabled GPUs on such devices. Our implementation takes into account the special nature of the images captured by the front-facing camera of mobile devices and exploits the GPUs present in mobile devices without CUDA-based frameworks, to meet these challenges.

1. Introduction

Current methods of authenticating users on mobile devices are mostly PIN or pattern based, which provides authentication only during the initial login. Password-based methods are susceptible, because people sometimes set passwords that are easy to guess or are repetitive [1] and pattern-based systems are vulnerable to smudge attacks [2]. Once the attacker successfully bypasses the initial authentication barrier, the phone has no way of blocking or denying the attacker. Continuous authentication systems deal with this issue by continuously monitoring the user identity after the initial access to the mobile device based on how the user interacts with the mobile device. Examples of such systems include touch gesture-based systems [3], [4], [5], face-based systems [6], [7], [8], gait-based systems [9], stylometry-based methods [10], speech and face-based method [11] [12] and sensor-based methods [13], [14]. It has been shown that face-based recognition can be very effective for continuous authentication [11], [7], [15], [8].

Face detection is a very important step in face-based authentication systems. There has been substantial progress in detecting faces in images, which have impressive performances on challenging real-world databases [16]. But such databases are predominantly composed of general surveillance or media type images and not specifically of images captured using front-facing cameras of smartphones. As we shall discuss later, face images captured using the front-facing cameras of mobile devices possess some unique features that can be used as powerful prior information to simplify the task of face detection on mobile platforms. This paper proposes a deep convolutional neural network (DCNN)-based face detection scheme for mobile platforms.

1.1. Motivation

State of the art face detection techniques are based on DCNNs [17], [18]. Variations of DCNNs have been shown to perform well in various datasets like Face Detection Dataset and Benchmark (FDDB) [19] and Annotated Face in-the-Wild (AFW) [20]. Though DCNN-based methods can run on serial processors like CPUs, they are prohibitively slow without parallel processors like GPUs. Mobile devices and consumer electronics products like cameras often have in-built face detection systems, but since they do not have much computational horsepower, simpler detection algorithms are implemented on them, which do not have as high a performance as DCNN-based methods but can run on low power mobile platforms. Thus, there is a tradeoff between high performance and hardware and power constraints. This paper seeks to reconcile the two competing objectives and studies the feasibility and effectiveness of DCNN-based face detection methods in mobile platforms. Clearly, the most powerful DCNN-based face detectors that are designed to run on desktop environments will not be a good candidate for a DCNN-based detector for mobile platforms. Below are a few differences between the two tasks.

1. Differences in hardware and software setup:

    - The de facto hardware requirement for DCNNs is a powerful Nvidia GPU. Clearly, mobile GPUs are much less powerful, hence the algorithms need to be simpler.

    - Most DCNN frameworks use a CUDA backend, but since most mobile GPUs are not made by Nvidia, they do not support CUDA. Hence, a more portable software stack is needed.

2. Differences in dataset:

    - Generic face databases may have images with multiple small faces while the front-facing camera captures face images when the user is using the phone and hence may have one large face image. Therefore, we can restrict ourselves to detecting a single face only. Also, given the typical distance at which the user interacts with his or her phone, we can make assumptions about the maximum and minimum sizes of the captured faces.
    - The images captured by the front-facing camera usually have the user's face in a frontal pose. Extreme pose variations are rare and one can focus on detecting faces with minor pose variations.
    - Faces captured by the front-facing camera, however, tend to be partial. A mobile face detector should be equipped to detect partial faces, which is not the focus of many generic face detectors.

This paper makes the following contributions:

- Exploiting the unique nature of the face detection problem on mobile platforms, we design an effective, simplified DCNN-based algorithm for mobile platforms that need not be as powerful as general face detectors, but is fine-tuned to work in a mobile setting.
- Most of the existing implementations of DCNNs use a CUDA backend, but most mobile GPUs are not Nvidia GPUs, hence they do not support CUDA. We develop libraries (in OpenCL and RenderScript) to implement DCNN-based algorithms on GPUs without resorting to CUDA, so that the algorithm is portable across multiple platforms.

Rest of the paper is organized as follows. We first survey related works that have influenced the current algorithm and discuss their advantages and disadvantages. Section 2 introduces the algorithm in full details and ends with a discussion on the salient features of the algorithm. Section 3 explores the details of the actual implementation of the algorithm on a mobile platform. Section 4 presents evaluation results of the algorithm on two datasets, UMD-AA and MOBIO. Finally we draw some conclusions about the algorithm and suggest some future directions.

1.2. Related Work

Cascade classifiers form an important and influential family of face detectors. Viola-Jones detector [21] is a classic method, which provides realtime face detection, but works best for full, frontal, and well lit faces. Extending the work of cascade classifiers, some authors [22] have trained multiple models to address pose variations. An extensive survey of such methods can be found in [16].

Modeling of the face by parts is another popular approach. Zhu *et al.* [20] proposed a deformable parts model that detected faces by identifying face parts and modeling the whole face as a collection of face parts joined together using 'springs'. The springs like constraints were useful in modeling deformations, hence this method is somewhat robust to pose and expression changes.

As mentioned before, current state-of-the-art methods involve deep networks, which have been extensively adopted and studied both by the academic community and industry. Current face detectors at commercial companies like Google and Facebook use massive datasets to train very deep and complex networks that work well on unconstrained datasets, but they require huge training datasets and powerful hardware to run.

Recent studies have shown that in the absence of massive datasets or hardware infrastructure, transfer learning can be effective as it allows one to introduce deep networks without having to train it from scratch. This is possible as lower layers of deep networks can be viewed as feature extractors, while higher layers can be tuned to the task at hand. Therefore, one can use the lower layers of common deep networks like AlexNet [23] to extract general features, that can then be used to train other classifiers. Works of Bengio *et al.* [24] have studied how transfer learning works for deep networks.

Specific to the mobile platform, Hadid *et al.* [6] have demonstrated a local binary pattern (LBP)-based method on a Nokia N90 phone. Though it is fast, it is not a robust method and was designed for an older phone. Current phones have more powerful CPUs and more importantly, even GPUs, which can implement DCNNs.

Finally, let us consider the datasets used for mobile face detection. While there are many face databases available, they are not suitable for evaluating mobile face detection algorithms. MOBIO is a publicly available mobile dataset [11] which consists of bi-modal (audio and video) data taken from 152 people, but it is a very constrained one as users are asked to keep their faces within a certain region, so that full faces are captured. A more suitable dataset for our purpose is the semi-constrained UMD-AA dataset [7], which shall be described in a later section.

2. Deep Features-based Face Detection on Mobile Devices

As mentioned briefly before, transfer learning is an effective way to incorporate the performance of deep networks. The first step of the Deep Features based Face De-

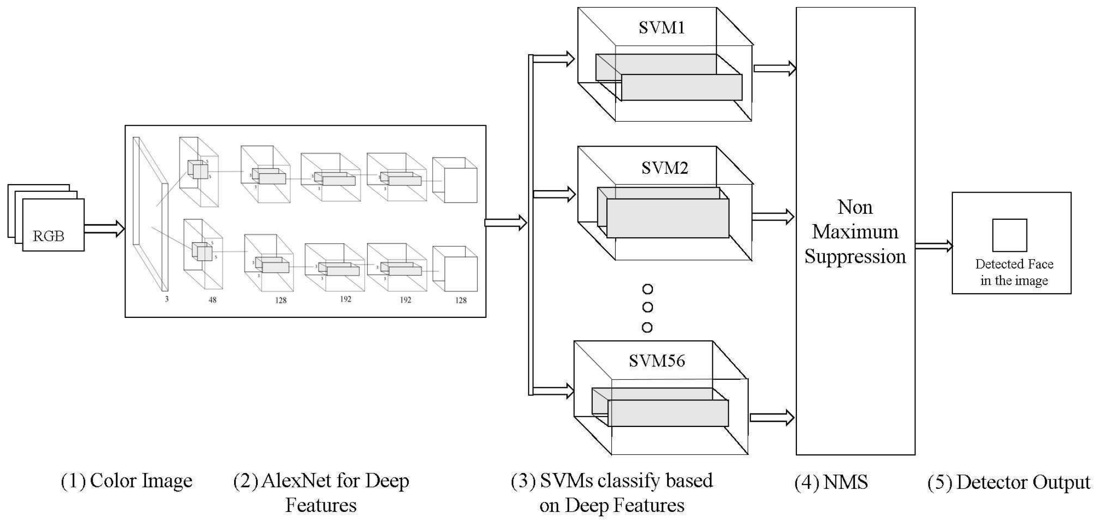
RGB
SVM1
SVM2
SVM56
Non Maximum Suppression
Detected Face in the image
(1) Color Image
(2) AlexNet for Deep Features
(3) SVMs classify based on Deep Features
(4) NMS
(5) Detector Output

Figure 1. Overview of the proposed deep feature-based face detection algorithm for mobile devices.

tection on Mobiles (DFFDM) algorithm is to extract deep features using the first 5 layers of Alexnet. Different sized sliding windows are considered, to account for faces of different sizes and an SVM is trained for each window size to detect faces of that particular size. Then, detections from all the SVMs are pooled together and some candidates are suppressed based on an overlap criteria. Finally, a single bounding box is output by the detector. In the following subsections, the details of the algorithm and model training are provided. Figure 1 provides an overview of the entire system.

2.1. Dataset

The UMD-AA dataset is a database of 720p videos and touch gestures of users that are captured when the user performs some given tasks on a mobile device (iPhone) [7]. There are 50 users (43 males and 7 females) in the database, who perform 5 given tasks (eg, typical tasks like scrolling, reading, viewing images etc.) in three illumination conditions (a room with natural light, a well-lit room and a poorly lit room). A total of 8036 images, spread over all users and all sessions, were extracted from these video recordings and manually annotated with bounding boxes for faces. Of these 6429 images had user's faces in the frame and 1607 were without faces, or with faces at extreme poses, with eyes and nose not visible or a very small partial face visible in the frame, which are all the cases when we can safely say there is no face present in the frame.

2.2. Training SVMs

For training, 5202 images from the UMD-AA database is used. Analysing the distribution of face sizes, we find that the height of faces vary from around 350 to 700 and the width varies from 300 to 600. A 2D histogram of the height and widths of the faces in the dataset are shown in Figure 2.

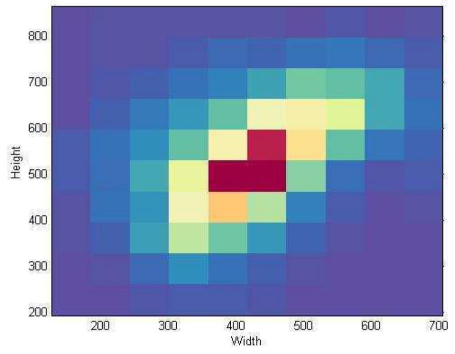

Figure 2. A histogram showing distribution of bounding box widths and heights.

Now the images are captured at 720p resolution (1280 rows x 720 columns). But since that resolution is too high for our purpose, we resize it to 640 x 360. Therefore typical faces range from 175 to 350 rows and 150 to 300 columns in this reduced resolution.

First we extract deep features from these resized images by forwarding them through AlexNet [23]. We tap the network at the 5th convolutional layer (after max-pooling). The standard AlexNet reduces an image by a factor of 16 in both dimensions. Thus, if the k^{th} input image is of size $p_k \times q_k$, the output is of dimensions $w_k \times h_k \times 256$, where the feature space width w_k and height h_k are given by (1)

$$w_k = \lceil p_k/16 \rceil, h_k = \lceil q_k/16 \rceil. \tag{1}$$

The 3rd dimension is 256 because the 5th layer of AlexNet uses 256 filters. Given the typical face dimensions in the last paragraph, they are reduced by a factor of 16 in the feature space to heights ranging from 10 to 22 and widths ranging from 9 to 19 approximately. Obviously, a single sized sliding window cannot account for these varying sizes, therefore we consider windows of width starting from 8 and increasing to 20 in steps of 2, and height starting from 9 and increasing in steps of 2 to 23. In total we get 56 different window sizes for which we need to train 56 different SVMs. We denote a window by W_{ij}, where i denotes its window height and j denotes its window width.

Let w_k and h_k, as defined in (1), denote the width and height of the deep feature for the face in the k^{th} training image. The face from the k^{th} training image is used as a positive sample for the SVM W_{ij}, if Eq. (2) is satisfied.

$$|i - h_k| \leq t_p \; \& \; |j - w_k| \leq t_p, \tag{2}$$

for some threshold for selecting positive samples, t_p. That is, we select those faces for W_{ij} whose sizes are comparable and close to the window's dimensions.

For negative samples, we extract random patches of size $i \times j$ from those training samples which have no faces. If the k^{th} training sample has a face of size $w_k \times h_k$, and for a particular window W_{ij}, if (3) holds,

$$|i - h_k| > t_n \; \& \; |j - w_k| > t_n, \tag{3}$$

for some threshold for selecting negative samples, t_n, then we extract a few random patches from the k^{th} training sample that act as negative samples for W_{ij}. That is, if the face in an image is of a very different size from the current window W_{ij} under consideration, we extract negative samples from it, so that W_{ij} gives a negative response of faces of different size. Finally, since the UMD-AA database does not have many images with no faces, we extract some random negative patches from images of the UPenn Natural Image Database [25].

Once we have extracted the positive and negative samples for each window size, we discard those window sizes which do not have enough positive examples. Then we convert the three dimensional deep feature patches into a single dimensional feature vector. Thus for W_{ij}, we get a feature vector of length $i \times j \times 256$. We estimate the mean and standard deviation of features from each window, which are used to normalize the features.

Next we train linear SVMs for each window. Since we get a very long feature vector, it is difficult to train an SVM with all positive and negative samples together. To make the training tractable, we divide the samples into batches and train over many cycles. Specifically, let p_{ij} be the number of positive samples for W_{ij}. Then we choose a small number of negative samples say n_{ij} and train the SVM. Then we find the scores of the n_{ij} negative training samples using the weights we get after training and retain only those that are close to the separating hyperplane and discard the rest. We refill the negative samples batch with new negative samples and continue this cycle multiple times. This procedure is performed for each SVM.

2.3. Full Face Detection Pipeline

After the SVMs are trained, we can scan the deep feature extracted from the given image k in a sliding window fashion for each SVM. Specifically for an image of size $w_k \times h_k$, the deep feature is of h_k rows and w_k columns as given by (1) and 256 depth. Therefore, for W_{ij}, we can slide the window from position $(1, 1)$, which is the top left, to $(h_k - i, w_k - j)$. Let (r_{ij}, c_{ij}) denote the position where the SVM yields highest score. Then we say that a bounding box, whose top left is at $16 \times (r_{ij}, c_{ij})$ and has width $16 \times j$ and height $16 \times i$ is the prediction from W_{ij}. Note that we multiply by 16, because the feature space's height and width is approximately 16 times smaller than that of the original image.

Now that we have 1 prediction from each of the 56 SVMs, we need to combine them to get a single prediction. A modified version of the non maximal suppression scheme used by Girshick *et al.* [26] is used for this purpose. First we sort the 56 proposals by their scores and then pick the candidate with the highest score. Boxes that overlap significantly with it and have a considerably lower score than it are ignored. This is continued for the next highest scoring candidate in the list, till all boxes are checked. After this we process the remaining candidates by size. If a larger box significantly overlaps a smaller box, but the larger box has a slightly lower score than the smaller box, we suppress the smaller box. This is useful in the following scenario: A smaller SVM may give a strong response for part of a full face, while the larger SVM responsible for detecting faces of that size may give a slightly lower response. But clearly the larger SVM is making the correct prediction, so we need to suppress the overlapping smaller SVM's candidate. After performing these suppressions, we pick the SVM's candidate that has the highest score. We then choose a suitable threshold, and if final candidate's score is larger than that, we declare a face is present at that location, else declare that there is no face present.

2.4. Salient Features

Sliding window approaches usually work on the principle of extracting appropriate features and then sliding a window and deciding if an object is present in that window or not. The proposed algorithm, DFFDM, can be thought of as using DCNNs to extract the features for the sliding window approach. However, to make the sliding window approach work for detecting faces of varying scales, we need to extract features across scaled versions of the input image. The approach followed by Ranjan *et al.* in [17] is based on extracting deep features at multiple resolutions of the image and then training a single SVM to detect faces.

Clearly extracting deep features is a very costly operation because of the sheer number of convolutions involved. Passing the image at multiple resolutions through the network increases the workload even more. Therefore, the proposed algorithm passes the image through the DCNN only once, but trains SVMs of different sizes to achieve scale invariance. Also, the different SVM sizes help in detecting partial faces. For example, tall and thin windowed SVMs are usually trained with left half or right half faces, while short and fat windowed SVMs are trained for top half of faces. SVMs whose aspect ratio match a normal full face's aspect ratio are trained on full faces. Thus, different sized windows help in scale invariance as well as in detecting partial faces.

3. Implementation

Current popular deep learning platforms include Caffe, Theano and Torch. Although, these platforms have a CPU only version, they are significantly slower than the GPU enabled versions. These platforms have a CUDA based backend that offloads the heavy, but parallelizable, computations involved in a convolutional deep network to an Nvidia GPU. Nvidia has been actively developing and supporting deep learning research and has released optimized libraries such as cuDNN. Thus, although there are multiple frameworks in the deep learning system, the computational backend is dominated by CUDA based-code and Nvidia GPUs.

Unfortunately, CUDA is proprietary and works only for Nvidia's CUDA enabled GPUs. Therefore, existing deep learning frameworks are difficult to port on to GPUs made by other vendors. Current mobile devices have GPUs that are predominantly provided by Adreno, Mali and PowerVR. Nvidia's mobile processor Tegra does power some phones and tablets, and these devices support CUDA, but the overwhelming majority of devices do not have CUDA enabled GPUs.

OpenCL [27] is an open standard, developed by Khronos Group, to support multiple vendors and facilitate cross platform heterogeneous and parallel computing. All major vendors like Qualcomm, Samsung, Apple, Nvidia, Intel and ARM conform to the OpenCL standard. Thus OpenCL is a portable option for implementing convolutional networks in GPUs other than those made by Nvidia. Recently though, Google has developed RenderScript to facilitate heterogeneous computing on the Android platform.

Mobile devices are obviously not an ideal platform to perform training on massive datasets. But once the model has been trained, we can hope to run the forward pass on mobile platforms. Thus to harness GPUs of mobile devices to perform the convolution heavy forward pass, we have implemented OpenCL and RenderScript-based libraries. The OpenCL library is general and should work on any GPU, while the RenderScript library is specifically tailored for Android. An Android specific example is the use of Schraudolp's fast exponentiation [28] to approximately but quickly compute the normalization layer in AlexNet. Full exponentiation takes a significant amount of time and can become bottlenecks in weaker mobile GPUs.

The OpenCL and RenderScript libraries implement the primary ingredients for a basic convolutional deep network: convolution and activation layers, max pooling layers and normalization layers, each of which can be parallelized on GPUs. By appropriately stacking up these layers in the correct combination and initializing the network with pretrained weights we can build a CNN easily. For our purpose we have implemented the AlexNet network as described earlier, but we can easily build other networks given its weights and parameters. For an image of size 360x640, a single forward pass, running on a machine with 4th generation Intel Core i7 and Nvidia GeForce GTX 850M GPU, takes about 1 second for the OpenCL implementation. For an image of the same size, on the Renderscript implementation running on different phones, we summarize the run time results in Table 1. Only about 10% or less of this run time is due to max-pooling layer, normalization layer, SVMs and non maximum suppression. The rest of the time is due to the heavy computations of the convolutional layers. Continuously running the algorithm on a Nexus 5 drains the battery at 0.45% per minute, while leaving the phone undisturbed drains the battery at around 0.16% per minute.

Phone	Runtime	GPU	CPU
Moto G	36.7 s	Adreno 305	Qualcomm Snapdragon 400
HTC One (M7)	31.2 s	Adreno 320	Qualcomm Snapdragon 600
Samsung Galaxy S4	28.0 s	Adreno 320	Qualcomm Snapdragon 600
Nexus 5	11.9 s	Adreno 330	Qualcomm Snapdragon 800
LG G3	10.3 s	Adreno 330	Qualcomm Snapdragon 801
Nexus 6	5.7 s	Adreno 420	Qualcomm Snapdragon 805

Table 1. Run times of DFFDM on different mobile platforms

4. Evaluation and Results

For evaluation, we consider common metrics like Precision-Recall plots, F1 scores and Accuracy. We compare the performance of our algorithm on the UMD-AA [7] and MOBIO [12] [11] datasets with Deep Pyramid De-

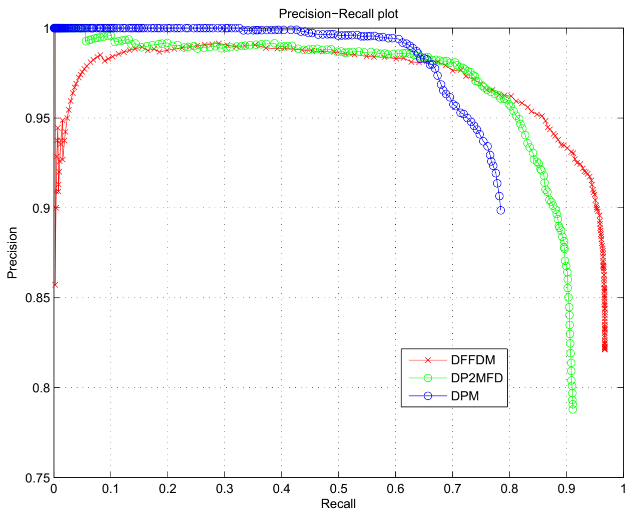

Figure 3. Precision Recall plot corresponding to the UMD-AA dataset.

formable Part Model (DP2MFD) [17], which is among the state-of-the-art algorithms for some challenging datasets like AFW and FDDB, deformable part model for face detection (DPM) [20] and Viola Jones detector (VJ) [21].

We compute detections based on 50% intersection over union criteria. Let d be the detected bounding box, g be the ground truth box and s be the associated score of the detected box d. Then for declaring a detection to be valid, we need Eq. (4) to be satisfied for some threshold t

$$\frac{area(d \cap g)}{area(d \cup g)} > 0.5 \;\&\; s \geq t. \tag{4}$$

4.1. UMD-AA Dataset

Results on UMD-AA dataset are summarized in Table 2.

Metric	DFFDM	DP2MFD	DPM	VJ
Max F1	92.8%	89.0%	84.1%	67.7%
Max Accuracy	88.0%	82.3%	76.4%	58.0%
Recall at 95% precision	85.7%	81.7%	72.6%	-

Table 2. Comparision of different metrics for various detectors on UMD-AA database

To check the robustness of the detector, we vary the intersection-over-union threshold as defined in Eq. (4) from 0.1 to 0.9 and plot the resulting F1 score in Figure 4 and accuracy in Figure 5. We see that the DFFDM algorithm gives better performance at higher overlap thresholds too.

A few example positive and negative detections are shown in Figure 7. The detections are marked in red, while the ground truth is in yellow. The first row shows a few examples of positive detections with partial faces and the second row shows positive detections with pose variations. The third row shows some false detections, or detections with score lesser than 1. The detector is quite robust to illumination change and is able to detect partial or extremely posed faces.

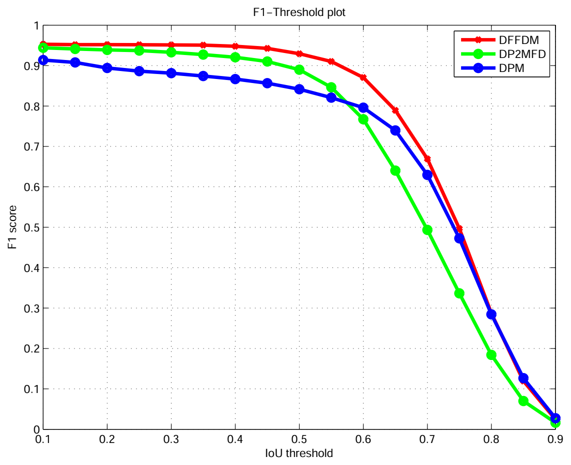

Figure 4. Plot showing variation of F1 score with respect to overlap threshold corresponding to the UMD-AA dataset.

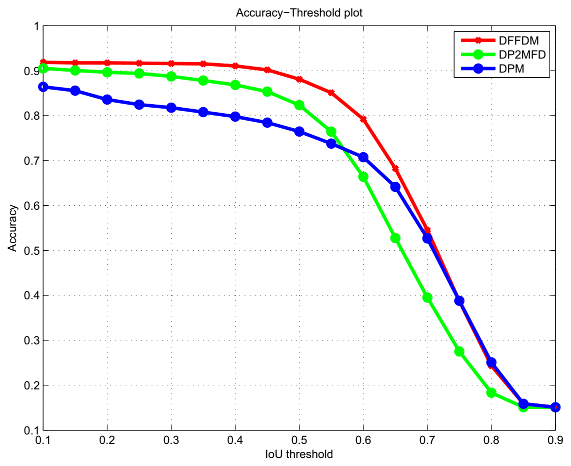

Figure 5. Plot showing variation of accuracy with respect to overlap threshold corresponding to the UMD-AA dataset.

4.2. MOBIO Dataset

Results on MOBIO dataset are summarized in Table 3. The MOBIO dataset has full frontal faces only, therefore we get very high performance. DP2MFD beats our algorithm for this dataset, which can be attributed to the fact that DP2MFD is one of the best algorithms, trained on a large, varied dataset, and for full frontal faces it has near perfect performance over multiple scales. For DFFDM,

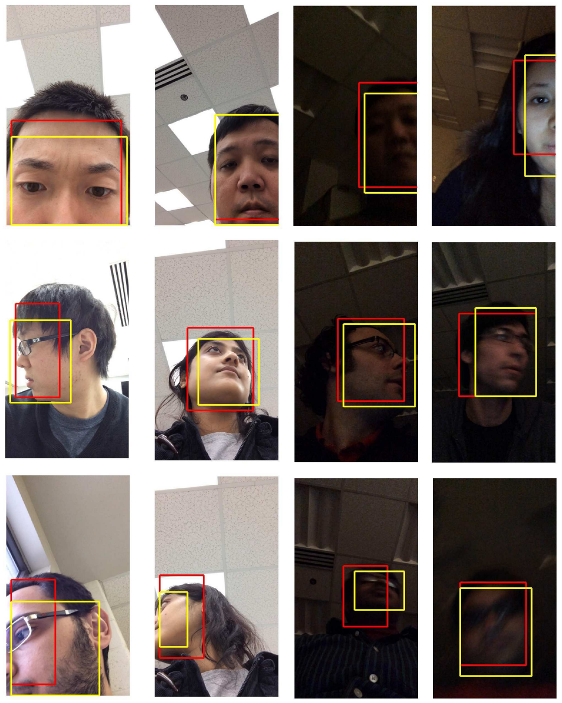

Figure 6. Examples of positive detections (with pose variations and occlusion, in first 2 rows) and examples of negative detections (due to insufficient overlap or low score in 3rd row) on UMD-AA. The detector's output is in red, while ground truth is in yellow.

SVMs of different sizes were trained, based on the typical size of faces captured by the front camera. But sometimes for very large or small faces, the training dataset of UMD-AA may not have enough samples, therefore for extremely scaled faces, DFFMD may fail. This can be remedied by training on a larger database, and also by training SVMs on more scales. A few example positive and negative detections are shown in Figure 7. The first row shows positive detections while the second row shows failures. As the examples show, there are some false detectiosn for really large faces, of which we did not have many examples in the UMD-AA training dataset on which DFFDM was trained.

Metric	DFFDM	DP2MFD	DPM	VJ
Max F1	97.9%	99.7%	97.8%	92.6%
Max Accuracy	96.0%	99.3%	95.8%	86.3%

Table 3. Comparision of different metrics for various detectors on MOBIO database

5. Conclusion and Future Directions

This paper presents a deep feature based face detector for locating faces in images taken by a mobile device's front camera. Keeping the constrained nature of the problem in

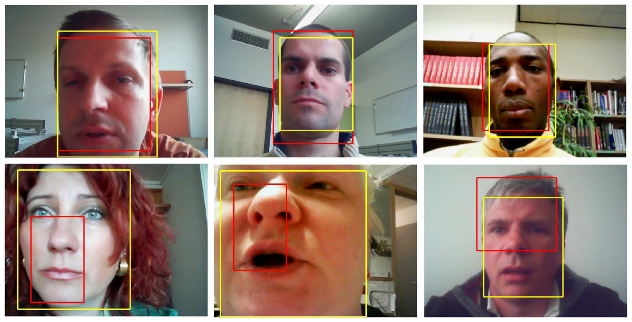

Figure 7. Examples of positive (1st row) and negative (2nd row) detections on MOBIO. The detector's output is in red, while ground truth is in yellow.

mind, the algorithm performs only one forward pass per image and shifts the burden of achieving scale invariance to the multiple SVMs of different sizes. As is expected from DCNN-based algorithms, it outperforms traditional feature-based schemes at the cost of a longer run time. Thus although DCNN based methods do not seem suitable for real time monitoring due to their run times on mobile devices, they can still be used as a backup in case a simpler detector fails. However there is much scope of optimizations and also mobile hardware has been getting more and more powerful, which looks promising.

This study also produced OpenCL and RenderScript based libraries for implementing DCNNs, that are more portable and suitable for mobile devices than CUDA based frameworks currently in popular use.

Future directions of inquiry includes code optimizations to make the GPU utilization faster thus speeding up the whole process. Also, we wish to explore simpler DCNNs that may be more suited to the mobile environment than a full blown AlexNet. Finally, the libraries used for this algorithm are more portable than CUDA based libraries and we hope to expand on them to facilitate research on deep networks on mobile GPUs.

Acknowledgement

This work was supported by cooperative agreement FA8750-13-2-0279 from DARPA.

References

[1] A. Vance. (2010, Jan) If your password is 123456, just make it hackme. [Online; posted JAN. 20, 2010]. [Online]. Available: http://www.nytimes.com

[2] A. J. Aviv, K. Gibson, E. Mossop, M. Blaze, and J. M. Smith, “Smudge attacks on smartphone touch screens,” in *Proceedings of the 4th USENIX Conference on Offensive Technologies*, 2010, pp. 1–7.

[3] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song, "Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 136–148, Jan 2013.

[4] H. Zhang, V. M. Patel, M. E. Fathy, and R. Chellappa, "Touch gesture-based active user authentication using dictionaries," in *IEEE Winter conference on Applications of Computer Vision*. IEEE, 2015.

[5] T. Feng, Z. Liu, K.-A. Kwon, W. Shi, B. Carbunar, Y. Jiang, and N. Nguyen, "Continuous mobile authentication using touchscreen gestures," in *IEEE Conference on Technologies for Homeland Security*, Nov 2012, pp. 451–456.

[6] A. Hadid, J. Heikkila, O. Silven, and M. Pietikainen, "Face and eye detection for person authentication in mobile phones," in *ACM/IEEE International Conference on Distributed Smart Cameras*, Sept 2007, pp. 101–108.

[7] M. E. Fathy, V. M. Patel, and R. Chellappa, "Face-based active authentication on mobile devices," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2015.

[8] P. Samangouei, V. M. Patel, and R. Chellappa, "Attribute-based continuous user authentication on mobile devices," in *International Conference on Biometrics Theory, Applications and Systems*, 2015.

[9] M. Derawi, C. Nickel, P. Bours, and C. Busch, "Unobtrusive user-authentication on mobile phones using biometric gait recognition," in *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Oct 2010, pp. 306–311.

[10] L. Fridman, S. Weber, R. Greenstadt, and M. Kam, "Active authentication on mobile devices via stylometry, gps location, web browsing behavior, and application usage patterns," *IEEE Systems Journal*, 2015.

[11] C. McCool, S. Marcel, A. Hadid, M. Pietikainen, P. Matejka, J. Cernocky, N. Poh, J. Kittler, A. Larcher, C. Levy, D. Matrouf, J.-F. Bonastre, P. Tresadern, and T. Cootes, "Bi-modal person recognition on a mobile phone: using mobile phone data," in *IEEE ICME Workshop on Hot Topics in Mobile Multimedia*, Jul. 2012.

[12] C. McCool and S. Marcel, "Mobio database for the icpr 2010 face and speech competition," Idiap, Idiap-Com Idiap-Com-02-2009, 11 2009.

[13] D. Crouse, H. Han, D. Chandra, B. Barbello, and A. K. Jain, "Continuous authentication of mobile user: Fusion of face image and inertial measurement unit data," in *International Conference on Biometrics*, 2015.

[14] A. Primo, V. Phoha, R. Kumar, and A. Serwadda, "Context-aware active authentication using smartphone accelerometer measurements," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, June 2014, pp. 98–105.

[15] H. Zhang, V. M. Patel, S. Shekhar, and R. Chellappa, "Domain adaptive sparse representation-based classification," in *IEEE International Conference on Automatic Face and Gesture Recognition*. IEEE, 2015.

[16] C. Zhang and Z. Zhang, "A survey of recent advances in face detection," Microsoft Research, Tech. Rep. MSR-TR-2010-66, 2010.

[17] R. Ranjan, V. M. Patel, and R. Chellappa, "A deep pyramid deformable part model for face detection," in *International Conference on Biometrics Theory, Applications and Systems*, 2015.

[18] S. S. Farfade, M. Saberian, and L.-J. Li, "Multi-view face detection using deep convolutional neural networks," in *International Conference on Multimedia Retrieval*, 2015.

[19] V. Jain and E. Learned-Miller, "Fddb: A benchmark for face detection in unconstrained settings," University of Massachusetts, Amherst, Tech. Rep. UM-CS-2010-009, 2010.

[20] X. Zhu and D. Ramanan, "Face detection, pose estimation, and landmark localization in the wild," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2012, pp. 2879–2886.

[21] P. A. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.

[22] C. Huang, H. Ai, Y. Li, and S. Lao, "High-performance rotation invariant multiview face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 4, pp. 671–686, 2007.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, p. 2012.

[24] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *CoRR*, vol. abs/1411.1792, 2014. [Online]. Available: http://arxiv.org/abs/1411.1792

[25] G. Tkaik, P. Garrigan, C. Ratliff, G. Milinski, J. M. Klein, L. H. Seyfarth, P. Sterling, D. H. Brainard, and V. Balasubramanian, "Natural images from the birthplace of the human eye," *PLoS ONE*, vol. 6, no. 6, p. e20409, 06 2011. [Online]. Available: http://dx.doi.org/10.1371%2Fjournal.pone.0020409

[26] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, 2014, pp. 580–587.

[27] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *IEEE Des. Test*, vol. 12, no. 3, pp. 66–73, May 2010. [Online]. Available: http://dx.doi.org/10.1109/MCSE.2010.69

[28] N. N. Schraudolph, "A fast, compact approximation of the exponential function," 1999.