Filter Application on Facial Features in a Smartphone App

Sofia Morar
Ovidius University of Constanta
124 Mamaia Bd., 900527
morarsofia@gmail.com

Elena Pelican
Ovidius University of Constanta
124 Mamaia Bd., 900527
epelican@univ-ovidius.ro

Dorin-Mircea Popovici
Ovidius University of Constanta
124 Mamaia Bd., 900527
dmpopovici@univ-ovidius.ro

ABSTRACT
Filter application on facial features is a rather new field and it has quickly become essential in all the social network applications. Fast and accurate filter application is still a field to be explored. In this paper, an automatic application of filters of different makeup products (lipstick and blush) is developed on the facial features of interest (lips and cheeks). Facial features are recognized and extracted with the help of a Machine Learning API (Application Programming Interface), Google Mobile Vision API. The application of the filters is developed using Bezier curves and the basic principles of graphics for the correct rendering of layers. The application of the filters is developed in an iOS App, with two functionalities, application on a picture, as well as in real-time through the phone's camera, where we also use a Convolutional Neural Network (CNN) in order to recognize the user's face, which introduces us into the field of Augmented Reality (AR) and Deep Learning.

Author Keywords
Filter Application; Facial Features; Real-Time; Augmented Reality; Bezier Curves; iOS; Convolutional Neural Network (CNN); Image Classification; Deep Learning; Network Architecture;

ACM Classification Keywords
H.5.m. Miscellaneous. H.5.1. Multimedia Information Systems. D.4.7 Organization and Design; F.2.1. Numerical Algorithm Problems; F.2.2. Nonnumerical and Algorithm Problems; H.1.2. User/Machine Systems; I.5. Pattern Recognition.

INTRODUCTION
Smartphone applications are becoming more popular everyday, but due to the limited hardware and software resources, they are harder to use for Machine Learning development. In the last 5 years, the mobile application industry has immensely developed creating different APIs, IDEs, programming languages in order to enhance the capabilities of smartphones.

Filter application on facial features is becoming mandatory in every social networking smartphone app, but makeup apps are still uncommon. In this work, we focus on creating an easy to use app for every smartphone user.

To this end, we have focused on an existing and accessible technology such as the iPhone 7. The proposed application is compatible starting with the iPhone software iOS 8.0 up until iOS 11.0. The devices that can be used on are both iPhones and iPads. We chose the Apple devices because of their performing cameras, and ultra-high video and photo resolution. The program in which the application has been developed is a specialized, OS X, iOS etc. application oriented IDE, Xcode [4].

The benefits of such an application is that the user can preview different products anytime, anywhere with a few touches on the smartphone's touchscreen.

The first contribution of this paper is the fast and cheap facial feature detection deployment using an open-source API: Google Mobile Vision [1]. Facial Feature detection is a well-known problem in the Machine Learning field, discussed in detail by Paul Viola and Michael J. Jones et al. [5]. Unfortunately, the process of training the algorithm with a dataset of thousands of pictures and then detecting the facial features is a long process. In order to speed up the process of detecting the human face, and the facial landmarks afterwards, we have customized the facial landmark detecting algorithm through the Google Mobile Vision API.

The second contribution of this paper is to create the filters and apply them on the features of interest. The filter rendering [3] is developed using CoreAnimation and CoreGraphics animation infrastructure (technologies available for iOS and OS X devices) [6]. Functions using Bezier curves are created in order to mimic the shape of real-life makeup looks.

The third contribution of this paper is the introduction into the augmented reality field. The Real-Time preview of the filters is a revolutionary way to deliver information to the user. The rendering of the filters using the frontal and/or rear-view camera is a step into Augmented Reality (AR) technology. Through this feature, the user can better experience the makeup application on their facial features.

The fourth contribution of this paper is the implementation of a Convolutional Neural Network (CNN) that recognizes the face of the user, therefore diving into the Deep Learning field.

The rest of this paper reviews the facial feature detection API usage and construction; describes the filter application process; outlines the implementation of a CNN that recognizes the face of the user; introduces us into the AR field; describes our experimental results; and concludes with discussions.

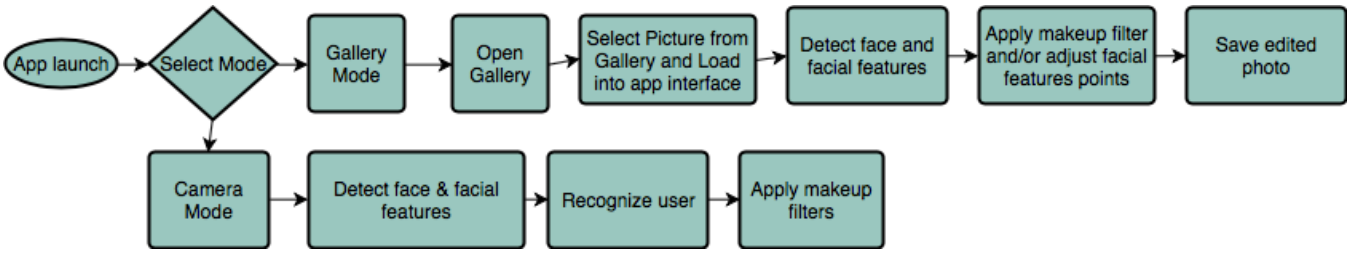

Figure 1: Structure of the application

RELATED WORKS

In this section we discuss related works from facial features detection, makeup filter application and face recognition. However the research topics mostly focus on PC applications rather than smartphone applications.

The problem of facial features detection has been discussed in detail in many works which found different approaches to extracting the features. In (Priyanka et al. 2015), Priyanka, et al. discuss different approaches for facial recognition and classification, from PCA (Principal component analysis) to the implementation of a neural network. In (Wang et al. 2017), Wang et al. review over 250 papers regarding face feature extraction. Many approaches are presented in their review, the use of CNNs is listed as one of them. Another approach is the use of a framework that includes components such as filtering, encoding, spatial pooling and holistic representation.

Makeup filters have recently begun to draw attention in pattern recognition and computer vision areas. In (Liu et al. 2013), Liu, et al. developed a Very-Deep CNN for makeup analysis and application where the application analysis the given face of the input picture and determines which makeup look is better suited, after that it transfers the chosen makeup look on the input face. In (Prasad, et al. 2014), Prasad et al. present an implementation of the Luxand Face SDK for facial feature extraction and cream makeup application in a web application. In (Wang, et al. 2014), Wang et al. propose a makeup detection and remover framework using a low-rank dictionary algorithm.

Face recognition has been a fast growing and challenging area in real time applications. A wide selection of methods has been developed in the last decades. In (Bhele et al. 2012), Bhele et al. discuss various implementation methods for face recognition. Algorithms such as PCA, LDA, ICA and SVM are presented. However, these algorithms have one faulty in common which is that they do not perform well enough in real time preview. Therefore, the idea of using a CNN is natural.

In comparison to the previously mentioned works, in this paper we implement a free makeup filter rendering and facial feature extraction method for a smartphone application. We continue the idea of implementing a convolutional neural network for the user recognition feature.

RELATED APPLICATIONS

Many smartphone applications have been developing filter application on facial features. The most popular application is called Snapchat, and it has face tracking and feature detection algorithms implemented. They use Delaunay triangulation to extract the facial features and to modify them (enlarge eyes, apply filters, modify the shape of the lips, etc). Two other phone applications, which can be installed on both Android and iOS operating systems are YouCam Makeup from Perfect Corp. and MakeupPlus from Xiamen Meitu Technology Co. Ltd. Both of these applications use the Mood Me SDK in order to detect the facial feature points and apply different makeup filters, on pictures and in real time preview. The reason we did not opt for the use of Mood Me SDK is because we have insisted on discovering a free, open-source method for applying makeup filters.

2. OUR IOS APPLICATION

The iOS application described in this paper presents the following structure (Figure 1):

- The user selects the gallery mode, and the photo gallery opens.
- The user selects a picture from the photo gallery and the picture selected is loaded into the application's interface.
- The application recognizes the faces and facial features.
- The facial feature points detected can be adjusted, if desired.
- Makeup filters of different colors are applied, and the intensity of the filters adjusted.
- The edited photograph is saved into the smartphone's photo gallery.
- If the real time preview is selected, the camera activates.
- The faces and the facial features in the camera preview are detected.
- The user's face is recognized.
- Makeup filters are rendered in real time.

3. FACIAL FEATURE DETECTION

3.1 Google Mobile Vision API

Google released in 2015 an open-source Machine Learning specialized API, part of the ML-Kit (Software Development Kit) [7]. The API is easy to integrate in both iOS and Android mobile applications. We have used the facial features detection sub-package of the framework.

The first step that the API takes is to detect a face in the image or in the live video feed. Xiangxin Zhu et al. [8] proposed an algorithm through which they can detect the face of a person in a real time environment. They base their work on the Viola-Jones algorithm, OpenCV frontal, Boosted frontal + profile face detector Z. Kalal et al. [9] and DPM (Deformable Part Model).

The second step is to detect the facial features of interest (lips and cheeks) as seen in Figure 2. Michel Valstar et al. [10] proposed an algorithm to detect more facial points in pictures than the Viola-Jones algorithm. They talk in their paper about SVR (Support Vector Regressor) and Markov Network implemented in order to create the BoRMaN Algorithm for a more precise facial points prediction. The Google Mobile Vision API has another specific feature: the smile detection. Yu-Hao Huang et al. [11] speak about a video-based solution for smile percentage detection. They consider that the distance between the left and right corner of the mouth plays an important role in the calculation of the smile percentage.

Figure 2: Facial Feature Detection after Implementing the Google Mobile Vision API and Top Lip Detection

3.2 Top Lip Detection

The Google Mobile Vision API detects only 4 points around the mouth area: left corner, right corner, bottom middle lip and the middle of the mouth. In order to properly apply the lipstick filter we have to detect the top lips cupid bow points.

The first step is to calculate the middle (Figure 3) of the cupid bow in order to create the proper shape of the filter. We use the bottom lip and the middle of the mouth points detected by the API as reference. The point is calculated with the symmetry formula. If the person in the picture is smiling, we adjust the formula accordingly.

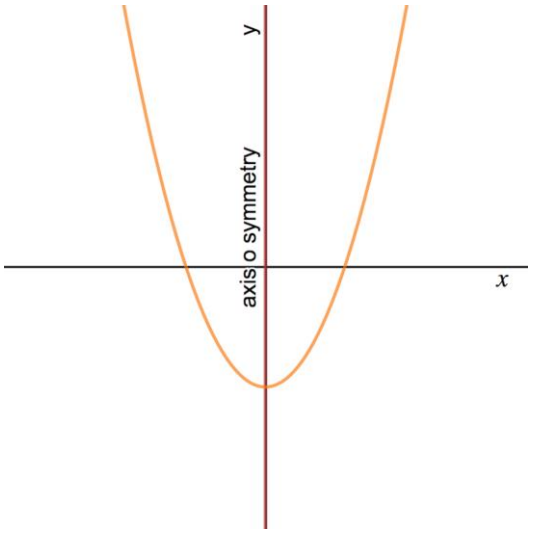

Figure 3: Graphic representation of the symmetry axis

Algorithm 3.1: Symmetry_point

Input: Points detected by the Google Mobile Vision API
Output: The three points of the cupid's bow
Set points[] = 0
v = variable used to adjust the values
if (center mouth point **and** bottom lip point == found) **then**

$$middle\quad mouth\quad line = \frac{(left\quad mouth\quad point + right\quad mouth\quad point)}{2}$$

$$symmetry\quad point(x,y) = \frac{center\quad mouth\quad point(x,y)^2}{bottom\quad lip\quad point(x,y)} : v$$

$$smile(x) = \frac{center\quad mouth\quad point(x) * middle\quad mouth\quad line(x)}{2}$$

$$smile(y) = \frac{v * (middle\quad mouth\quad line(y))^2}{bottom\quad lip\quad point(y)}$$

end if
if ((center mouth point(y) - middle mouth line(y)) < 2) **then**
points.append(smile)
else
points.append(symmetry point)
end if

For the left and right part of the cupid bow we approximate the values and add or subtract from the symmetry_point variable, depending on the axis we are on Ox or Oy. The results after implementing the algorithm in the application can be seen in the Figure 2.

3.3 Adjustment of the Feature Points

In order to make the iOS application user-friendlier we have implemented an option to adjust the facial feature points in case the algorithm did not perfectly detect them.

We have used the Euclidean Distance [12] to locate the current point (the point where the user touches on the touchscreen). If the distance is smaller than the set minimum, the Euclidean distance becomes the new minimum. The points can be moved by the user before the makeup filters are applied (Figure 4) or after the makeup filters are applied (Figure 5).

Algorithm 3.2: Facial Feature Points Adjustment

Input: Touch event
Output: The adjustment of the current point

current point = touch event()
pi = 0
min = v (value)
mini = 0
points[] = the values of the control points
for i in $0 : len(points)$ **do**

$$distance = \sqrt{((i(x) - current\ \ point(x)^2) + (i(y) - current\ \ point(y)^2))}$$

    if distance < min **then**
        min = distance
        mini = pi
    end if
    pi = pi + 1
end for

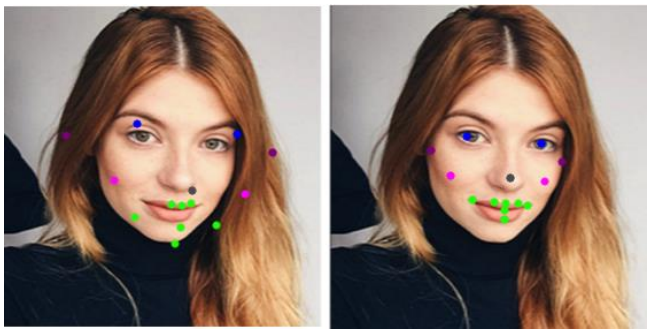

Figure 4: Adjustment of Facial Feature Points without filter

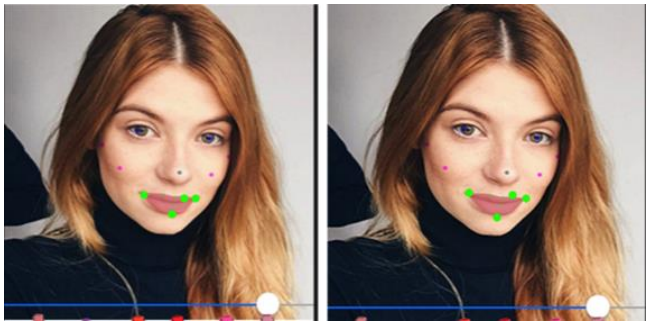

Figure 5: Adjustment of Facial Feature Points with filter

4. FILTER APPLICATION

4.1 Filters

Image editing has been in a continuous development in the past years. The newest trend is the filter application feature, recognizable in most social media and image editing applications. However, makeup filters are still progressing as they have not reached a true performance yet. In this paper, we have found a solution for creating easy to apply makeup filters to a picture and in real-time preview through the phone camera. We use Bezier curves [2] to create the filters and we apply them with the help of graphical layers (CoreAnimation Framework [16]).

A Bezier curve is a parametric curve that uses the Bernstein polynomials as a basis [13].

$$r(t) = \sum_{i=0}^{n} b_i \, B_{i,n}(t)$$

The b_i coefficients are the control points, that among with the function basis $B_{i,n}(t)$ determine the shape of the curve.

Michael S. Floater talks in his paper [14] about different types of Bezier curves and surfaces and how to apply them with computer technology.

The makeup filters are based on different Bezier curves and shapes created by the curves. We have created two different filters: one for lips and another one for the cheeks.

4.2 Filter for Lips

For the application of the lipstick filter we use the cubic Bezier curve. The cubic Bezier curve [15] has 4 points that guide the curve it creates: start point, control point 1, control point 2 and the endpoint. The distance between the start point and the control point 1 shows "how long" the curve moves into direction of the control point 1 before turning toward the endpoint. We have created a Bezier curve for the lower lip, using the left corner of the mouth as the start point, the right corner of the mouth as the endpoint and the control points formulas were adjusted with the combinations between the left and right corner of the mouth and the bottom of the mouth. The top lip can be created with two methods: if the lip presents a more rounded shape we use another cubic Bezier curve, or if the lip appears to be more of a sharp shape we simply unite the feature points of the lip (Figure 6).

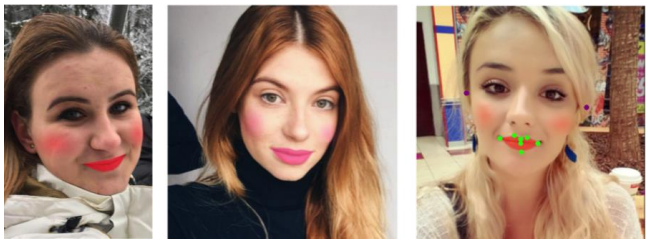

Figure 6: Makeup Filter Application with/without Control Points

4.3 Filter for Cheeks

The filter for cheeks was created with Bezier curves based on a primitive geometric shape (rectangle). The function creates a new Bezier Path object initialized with an oval shape inscribes in the specific rectangle. As control point for the filter we use the left and right corner of the mouth and the left and right ear feature points (the purple points as seen in Figure 4). The "gradient" fade effect is created with a loop which we apply to the opacity parameter (Figure 6).

5. USER RECOGNITION

Convolutional Neural Networks (CNNs) are becoming a popular method to classify images [17]. In this paper, we use CNNs in order to recognize the user's face in the real time filter rendering. After conducting different implementation experiments of projection-like algorithms, such as eigenfaces and tensor-like algorithms [30], we have concluded that previous mentioned algorithms did not obtain the desired results in the real time preview. We chose the CNN implementation because this method responds quickly and accurately in the real time preview.

5.1 Implementation Details

We have built and trained the CNN on a Macbook Pro 2014, 8GB 1600 MHz DDR3 Memory, 2.6 GHz Intel Core i5 CPU, Intel Iris 1536 MB GPU. The CNN was created using Tensorflow software [18] and Keras API [19]. The deployment for iOS was made by converting the model [20] and pickle [21] files with the Coremltools [22] Python package that creates a trained machine learning model file [23]. The implementation of the trained model into the iOS application is made with the Core ML [24] Framework (Machine Learning Apple Framework) and Vision [25] (Image Classification Apple Framework).

5.2 Architecture

The first step was to build a dataset with pictures containing the user's face, and pictures that do not contain the user's face. We have created a dataset that contains five classes with a total of 540 pictures, 102 if which contain the user's face, each with a representative subdirectory in order to parse the labels easily. The pictures resolution varies from 303x280 to 1762x1868 pixels.

Karen Simonyan et al. propose a CNN architecture solution for large-scale image recognition [26] that we have adapted for our dataset. During training, the input to our Network is a fixed-size 96x96 pixels with 3 channels (i.e. RGB). We subtract the mean RBG value from each pixel from the training set. We use 3x3 convolutional layers stacked on top of each other in increasing depth. The image is passed through the stack of layers, where we use filters to extract the features from the image. The first CNN layer has 32 filters with a 3x3 kernel. Every new added layer doubles the amount of filters it uses keeping the 3x3 kernel. All layers use a ReLU, $f(x) = x^{+} = \max(0, x)$, activation function (Kriehevsky et al., 2012). After that, we normalize the batches in order to speed up the learning process. Batch normalization process normalizes the output of a previous activation layer, subtracts the batch mean and divides it by a batch standard deviation. We use three spatial pooling layers, one for each layer of the CNN, in order to downsample the size of the image. By reducing the size of the picture we allow the CNN to make assumptions about features contained in the sub-regions binned. In the first layer, Max-pooling is performed on a 3x3 pixel window with a stride 3, whereas in the next two layers we decrease the size, using 2x2 pixel windows with stride 2. We use the dropout [27] $\min f \sum_{i=0}^{n} V((f(x_i), y_i) + \lambda R(f)$ function, to randomly disconnect nodes from the current layer to the next layer in order to naturally introduce redundancy into the model. The fully-connected layer is specified with a rectified linear unit activation and batch normalization. Finally we use the softmax [28] function, $\frac{\delta}{\partial_{q_k}} \sigma(q, i) = \cdots = \sigma(q, i)(\partial_{ik} - \sigma(q, k))$, to output the probability distribution.

5.3 Training

The training of the CNN was made on the Macbook Pro's CPU. For resulting dataset we have decided to train 20 epochs (i.e. how many times our network "sees" each training example and learns patterns from it). The batch size we chose to use is 32 (the CNN is trained in image batches). We chose the Adam optimizer to update the network weights iterative based in training data, because of its computational efficiency and little memory requirements. The performance of the model was measured with the cross-entropy classification loss function. We split the dataset into 80% for training and 20% for testing. The training took 20 minutes, approximately 60 seconds per epoch. The accuracy value on the first epoch started with 0.6973 and by the last epoch grew to 0.9373, whereas the loss value started with 1.1771 and decreased to 1.735 as seen in Figure 7.

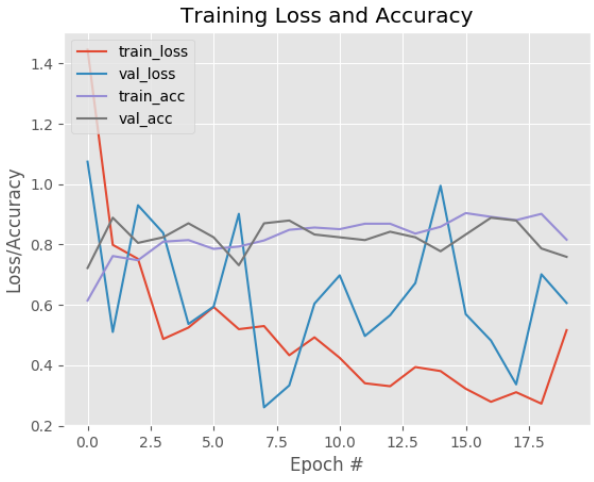

Figure 7: Training and Loss Accuracy

5.4 Testing

We have decided to test the CNN on the Macbook before converting it into an iOS app appropriate format. The testing was made on different pictures of the user. Given a trained CNN model and an input image, it is classified in the following way. The first step is to pre-process the image for classification. We resize the image to 96x96 pixels to fit out network, we extract the RBG pixel values from the image, and convert it into an array.

After the model and labels are loaded, the classification is made. If the input image filename contains the predicted label text, the result will be predicted as correct. Depending on the background of the picture, the results varied: if the picture was a close-up of the user's face the results came close to 100% as seen in the first picture in Figure 8, otherwise the results were around the 90 percentile (Figure 9 and Figure 10).

Figure 8: Testing of the CNN on PC and iOS

5.5 Results

After finding an appropriate architecture for the given dataset, and testing thoroughly the trained CNN model we converted the model in order to integrate it into the iOS application. The converter saves the network training results and the labels accordingly. The iOS application extracts the data from the trained model through the Core ML and Vision frameworks. The user recognition in the iOS application is made in a real time environment through the smartphone's front and back camera as seen in Figure 9 and in Figure 10. The results from the trained CNN varied on computer and phone testing. The difference of results on the phone are due to different external disturbing factors, such as: hand trembling, light change, the movement of the phone camera to and from the user's face. Evenso, the results were very good, even if the user was wearing glasses as seen in the second picture of Figure 9.

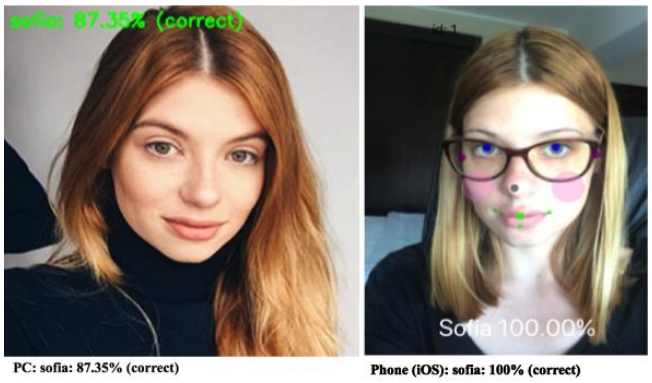

Figure 9: Testing of the CNN on PC and iOS

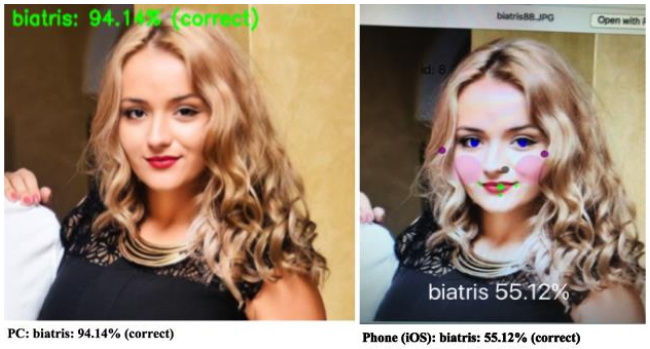

Figure 10: Testing of the CNN on PC and iOS

CONCLUSIONS

We have proposed a novel method for finding facial feature points calculated with the help of those created by the API and for applying filters on an image and in real-time preview. The filter application is based on graphical rendering with algorithms which include Bezier curves functions. The proposed method is sensitive to loaded backgrounds in pictures and moderate variations in head. The real-time filter application introduces us into the AR filed and we propose on rendering more real-time accurate filters. The user's face recognition through a Convolutional Neural Network based solution is a new way to personalize the user's preferences in the application.

Moreover, these preliminary results motivate us to extend our efforts at other facial features as user's eyes and eye gaze, as natural interaction techniques with ubiquitous systems.

ACKNOWLEDGMENTS

This work was supported by grant of the Romanian Ministry of Research and Innovation, CCCDI – UEFISCDI, project number PN-III-P1-1.2-PCCDI-2017-0917 / contract no. 21PCCDI/2018, within PNCDI III.

REFERENCES

1. Google Mobile Vision website: https://developers.google.com/vision/ last visited May 29 2018.
2. Ha Jong Wong, Choe Chun Hwa, Li Kum Song. On the mathematic modeling of non-parametric curves based on Bézier curves. *Eprint arXiv 1411.6365 (11/2014).*
3. Peter Shirley, Michael Ashikhmin, Michael Gleicher, Stephen R. Marschner, Erik Reinhard, Kelvin Sung, William b. Thompson, Peter Willemsen. Fundamentals of Computer Graphics. *A K Peters Wllesey, Massachusetts.*
4. Xcode website: https://developer.apple.com/xcode/ last visited May 22 2018.
5. Michael Jones, Paul Viola. Fast Multi-View Face Detection. *Mitsubishi Electric Research Laboratories Inc., 2003.*
6. Core Graphics Framework Apple website: https://developer.apple.com/documentation/coregraphics?changes=_7 Last visited at May 24 2018.
7. ML-Kit website: https://developers.google.com/ml-kit/ last visited May 18 2018.
8. Xiangxin Zhu, Deva Ramanan. Face Detection, Pose Estimation, and Landmark Localization in the Wild. Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference.
9. Z. Kalal, J. Matas, and K. Mikolajczyk. Weighted sampling for large-scale boosting. In *BMVC 2008*.
10. Michel Valstar, Brais Martinez, Xavier Binefa. Facial Point Detection using Boosted Regression and Graph Models. *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference. 1063-6919.*
11. Huang, Yu-Hao & Fuh, Chiou-Shann. (2018). Face Detection and Smile Detection.
12. Dokmanic Ivan, Parhizkar Reza, Ranieri Juri, Vetterli Martin. Euclidean Distance Matrices: Essential theory, algorithms, and applications. *IEEE Signal Processing Magazine, vol. 32, issue 6, pp. 12-30.*
13. S. Barnett, Polynomials and Linear Control Systems, Marcel Dekker, New York, USA, 1983.
14. Michael S. Floater. Bézier Curves and Surfaces. http://www.mn.uio.no/math/english/people/aca/michaelf/papers/bezier.pdf.
15. Khan, Khalid & Lobiyal, Daya & Kilicman, Adem. (2015). Bezier curves and surfaces based on modified Bernstein polynomials. *CoRR, 2015, vol.1511.06594*
16. Nick Lockwood .iOS Core Animatio: Advanced Techniques. *Addison-Wesley 2014.*
17. M Hakeem Selamat, Helmi Md Rais, "Enhancement on image face recognition using Hybrid Multiclass SVM (HM-SVM)", *Computer and Information Sciences (ICCOINS) 2016 3rd International Conference on*, pp. 424-429, 2016.
18. Tensorflow Software website: https://www.tensorflow.org last visited May 30 2018.
19. Keras API website: https://keras.io last visited May 30 2018.
20. Python Model Documentation: http://scikit-learn.org/stable/modules/model_persistence.html last visited May 30 2018.
21. Python Pickle Documentation: https://docs.python.org/2/library/pickle.html last visited May 30 2018.
22. Coremltools Documentation: https://pypi.org/project/coremltools/ last visited May 29 2018.
23. M. Zhang and Z. Zhou, "A Review on Multi-Label Learning Algorithms," in *IEEE Transactions on Knowledge & Data Engineering*, vol. 26, no. 8, pp. 1819-1837, 2014.
24. Core ML Framework website: https://developer.apple.com/documentation/coreml last visited May 28 2018.
25. Vision Framework website: https://developer.apple.com/documentation/vision last visited May 28 2018.
26. Simonyan, K. & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*, abs/1409.1556.
27. Pierre Baldi, Peter Sadowski. Artificial Intelligence Volume 210, May 2014, pages 78-122. *ISSN: 0004-3702. Elsevier B.V.*
28. Duan K., Keerthi S.S., Chu W., Shevade S.K., Poo A.N. (2003) Multi-category Classification by Soft-Max Combination of Binary Classifiers. In: Windeatt T., Roli F. (eds) Multiple Classifier Systems. MCS 2003. Lecture Notes in Computer Science, vol 2709. Springer, Berlin, Heidelberg.
29. R.L. Burden, J.D. Faires - *Numerical Analysis*, 9th edition, Brooks/ Cole, 2011.
30. Elena Pelican and Lăcrămioară Liță. Algoritmi pentru recunoasterea fețelor, MatrixRom, Bucuresti, 2015.