A Face Recognition Library using Convolutional Neural Networks

Leonardo Blanger[1], Alison R. Panisson[2]

[1]Department of Engineering and Computer Science, URI University, Erechim - Brazil
[2]School of Informatics, Pontifical University of Rio Grande do Sul, Porto Alegre - Brazil

***Abstract*—** *In this work, we propose a face recognition library, with the objective of lowering the implementation complexity of face recognition features on applications in general. The library is based on Convolutional Neural Networks; a special kind of Neural Network specialized for image data. We present the main motivations for the use of face recognition, as well as the main interface for using the library features. We describe the overall architecture structure of the library and evaluated it on a large scale scenario. The proposed library achieved an accuracy of 98.14% when using a required confidence of 90%, and an accuracy of 99.86% otherwise.*

***Keywords*—Artificial Intelligence, CNNs, Face Recognition, Image Recognition, Machine Learning, Neural Networks.**

I. INTRODUCTION

Nowadays, face detection and face recognition technologies are commonly present among online services provided by big Internet companies, such as Facebook and Google. Currently, some of the best image recognition techniques we have are based on variations of (Deep) Artificial Neural Networks (ANNs). For instance, the authors in [11] used such techniques to achieve record performance on the LSVRC[1]competition. However, the implementation of such systems is still complex, in the sense of knowledge required. This imposes a great difficulty for independent developers without the necessary AI expertise to apply face recognition in their own projects.

Among the practical situations where face recognition can be applied are:

- Surveillance systems, on places such as industries, offices, banks, etc., where it is necessary to have a control over the allowed staff;
- Employee control, in the form of biometric presence monitoring systems. Controlling the employee frequency provides useful metrics for human resources departments;
- Efficient login on applications, instead of a password, or in addition to it, in order to verify user authenticity on the recovering of forgotten passwords;
- Personalized user experience on systems such as smart houses, cars, or personal assistants;
- Social network applications on which users can share pictures;
- As an additional feature for mobile applications.

In order to provide an easy way for developers to apply face recognition on independent projects, we propose a software library for this task. The library created by us allows users to register new faces, monitoring the expected accuracy of the predictions, and predict probabilities over the registered faces when presented to images.

The proposed library works internally using a Convolutional Neural Network (CNN), a special kind of ANN with architecture specialized for taking advantage of some properties in image data. Beyond this, the library architecture makes heavy use of regularization techniques and artificial generation of data, in order to improve generalization capacity.

The main contributions of this work are: (i) we propose CNN architecture for image recognition; (ii) we developed a general purpose face recognition library for *Python* using the proposed architecture; and (iii) we evaluated the library architecture on a large scale scenario using a public dataset of face images.

II. ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANNs), or simply Neural Networks (NNs), are a kind of predictive model. It is considered a machine learning technique inspired by the way the biological brain works [15].

[1] www.image-net.org/challenge/LSVR

Standard NNs are usually interpreted as a set of layers of units, each unit being a function of the previous layer. There is always an input layer, which assumes the values of the input of the prediction, an output layer, which will present the prediction output, and a set of hidden layers, as in Fig.1. The term Deep Learning is usually associated with NNs with multiple hidden layers [15].

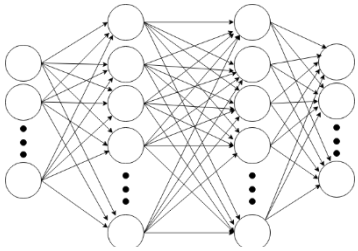

FIGURE 1 - AN ARTIFICIAL NEURAL NETWORK WITH TWO HIDDEN LAYERS

Each unit calculates a *logit* value, that is, a weighted sum of the outputs of the previous layer added to a bias value. The unit's output is then the application of an activation function on this logit value. The set of weights and the bias are the parameters that influence the mapping from the previous layer to the unit's output value. The goal of using ANNs is to adjust all the network parameters to make it predict some data correctly. For instance, predicting the person on a picture based on the pixels intensities. For this, the network uses a set of examples, called the training set. A good training set is a good approximation of the real distribution of data [6].

A cost function is a function, $C(\theta)$, where θ is the set of all the network parameters. This function evaluates how bad a specific configuration of the parameters is at predicting the training data. The training algorithm consists on iteratively adjust the parameters in order to decrease the cost. From calculus, we know that the gradient vector of a function points to the direction of steepest increment of this function. At the same time, the opposite direction points to the direction of steepest decrement. Based on this, at each iteration *t*, the network parameters are adjusted on the following way:

$$\theta_i^{t+1} = \theta_i^t - \epsilon \frac{\partial C}{\partial \theta_i} \tag{1}$$

Where ε is the **learning rate**, a (usually small) positive scalar that controls how much the parameters change at each iteration. This value is usually set to a large value at the beginning of training, to make the network quickly explore the parameter space and find a low cost region, and then we decrease it throughout the training, to make the network sensibly find the local minimum of this region.

This strategy of iteratively decreasing the cost using the partial derivatives is called **Gradient Descent**. We do not usually use all the training examples at each iteration. Instead, we select a different batch to estimate the cost at each iteration. This approach can take more iteration to find a good cost, due to statistical instabilities, but in advantage, it makes the individual iterations much faster. This strategy of estimating the cost using a random batch is called **Stochastic Gradient Descent** [15].

In order to make the network more capable of generalizing the prediction to new data (data not present on the training set), it is common to apply generalization techniques during training. These techniques aim to make the network learn more robust solutions, capable of identifying more frequent and realistic patterns from the training data, which are more likely to be present on the real data.

A commonly used regularization technique is *L2 Penalty* [15], which consists on adding another function to the cost, which grows proportionally to the square of the network parameters, scaled by some penalty. By decreasing the cost, the training algorithm will keep the parameters small in magnitude. This forces the network to find more simple solutions, capable of identifying more frequent patterns on the data.

Another common technique is *dropout* [11][7], which consists on, at each iteration, ignoring the output of some random units of a layer, using some predefined probability. This makes the units more individualistic during learning, which forces the network to find more robust solutions. When presented to unseen data, all the units are used, and the network is expected to perform better.

The easier way to make the network generalize better is by using more data for training, in order to have a sample that better approximates the true distribution of real data, minimizing the effect of individual irregularities. This can be done by either

collecting more data or by artificially generating more samples from the already collected data (by distortions, reflections or addition of noise)[17].

2.1 Classification Tasks

When performing classification tasks, it is useful to have a probability distribution over the possible categories as the model's output. For this, the last layer of an ANN is commonly a *softmax* layer. On such layer, the output of the unit, y_i, is defined as:

$$y_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2}$$

with z_i being the logit value of unit *i*.

III. CONVOLUTIONAL NEURAL NETWORKS

CNNs [12] are a kind of Deep Neural Network that uses Convolutional Layers as the first hidden layers of the network. These layers have an architecture designed to make better use of the knowledge obtained from the spatial structure present on image data.

Convolutional layers are based on the concept of local receptive fields[15]. On a convolutional layer, each unit is a mapping from a region (field) of neighboring outputs from the previous layer. In Figure 2, we have a receptive field of size 4×4, from a previous layer of 10×10. The connections to the first unit are shown in gray, on the left. For each of the next units, the receptive field is shifted right or down at a stride length. If this stride length is 1 on the horizontal direction, then the connections of the second unit are as in Figure 2 (right).

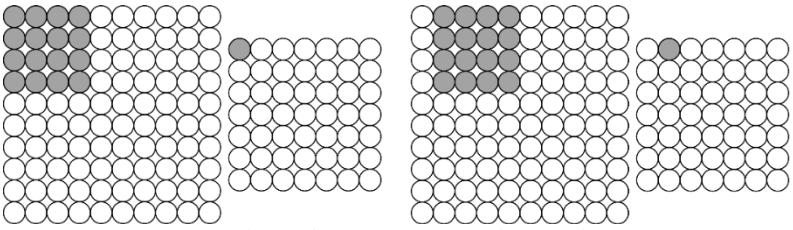

FIGURE 2 - CONNECTIONS OF THE FIRST (LEFT) AND SECOND (RIGHT) UNITS OF A CONVOLUTIONAL LAYER

Also, the set of parameters on all the units on the layer is shared, so that all the units perform the same mapping from different parts of the previous layer. The reason for sharing the mappings is to try to detect the same patterns that are present on different parts of the image. In order to detect multiple patterns, it is used not only one, but several mappings, each of which with a different set of parameters. Each mapping on a convolutional layer is called a *feature detector* [15].

In order to try to identify the presence of the same features on groups of nearby output units, the *polling* technique is frequently applied to the output of convolutional layers [15]. A common kind of pooling is the *max-pooling*, which consists on outputting the maximum value present on a group of units from the convolutional layer. The reason behind this is to try to detect the presence of features, losing some information about its exact position on the image. For instance, knowing that an image has a nose and an eye strongly indicates the presence of a face, not mattering where on the image these features are. The use of pooling on image classification can allow a certain degree of translation invariance [15], and thus, overcome the effects of small variations of pose, angle and distance. Another advantage is that they do not add any extra parameter to be trained.

The main motivation for the use of CNNs on image recognition is the fact that these kinds of models are capable of extracting spatial features from the data. Image data have a nice and useful hierarchical decomposition: groups of nearby pixels can form edges, groups of nearby edges can form polygons, and so on, until complex visual structures are formed. Because of this property, CNNs usually use a chain of convolutional and pooling layers, with the output of one being the input to the next. [14][4][11][10].

IV. FACE RECOGNITION LIBRARY

In this section, we describe the developed face recognition library, focusing on the methods that are used to register new faces, make the recognizer fit the model to the provided data, and use the recognizer to predict the probability of the registered faces when presented to new images. The main class of the library is the Recognizer class, which is instantiated as follows:

```
recognizer = Recognizer(width, height, channels,
                        image_dir, dataset_dir, models_dir,
                        qt_images, valid_proportion)
```

where the parameters `width` and `height` are the dimensions of the images to be fed into the network, `channels` is the depth of the pixels: 1 for grayscale, 3 for RGB, etc., `image_dir` is the path to save the training images, `dataset_dir` is the path to save the preprocessed training data, `models_dir` is the path to save the network's checkpoints, `qt_images` is the minimum number of images of each person to use, and `valid_proportion` is the proportion of these images to be used for validation, i.e., to estimate the prediction accuracy.

The register of a new person on the system is done as follows:

```
recognizer.add_person(pictures, name)
```

where `pictures` is a list containing the paths to the person's pictures that the user is providing for the library. If an image is not of size *width* × *height*, it is automatically reshaped. If there are less than `qt_images`, the library artificially generates more images from these ones, using different degrees of translations and rotations, in a similar way as is done in [17].

The generation of new images is performed as follows. The translations are made by shifting the pixels some number of units to the left, right, top, and bottom of the image, ignoring the parts that end outside the frame, and filling the empty pixels with black. The rotations are made both clockwise and counter-clockwise, also ignoring the outside parts and filling the empty pixels with black. Initially, the images are shifted one unit on each of the four directions, and one degree in each orientation, and then two, and so on, creating six new images for iteration, until the number of original images plus the generated ones reaches the required number of images for training.

The received images have their pixel values rescaled to avoid big input numbers. As grayscale pixels and single components of RGB pixels both assumes single integers from 0 to 255, the recognizer scales the pixel values to (`pixel_value` − 127.5)/255. After this rescaling, the pixel values of all images are compressed on a file inside `dataset_dir`, using a friendlier format for the training algorithm, and with a proper separation between training and validation sets.

To request the library to adjust the model to the currently registered people, we call:

```
recognizer.train_model(num_steps, valid_interval, start_over)
```

where `num_steps` is the number of iterations to train, `valid_interval` is the number of iterations on which the network accuracy is estimated on the validation set, and `start_over` is a boolean variable indicating that the model should either initialize its parameters and start training from the beginning, or use the last checkpoint saved on `models_dir`.

For using the library to predict the person on a picture, we need to call:

```
predictions = recognizer.predict(picture)
```

where this method receives `picture`, the path of the image to be analyzed, and returns a vector with dimension equals to the number of people registered, representing a probability distribution over these people. The higher the probability of a person, the more the network believes the image is from this person.

Other hyper-parameters (batch size, dropout probability, weight penalty, number of units per layer) can be easily modified through this Recognizer object. Although, for any change to reflect on the predictions, it is necessary to retrain the model. The same applies for the addition of new people.

V. NETWORK ARCHITECTURE

The network architecture was developed using *TensorFlow*[2][1], [2], an open source software library for numerical computation that was originally developed by researchers and engineers working on the Google Brain Team, for machine learning purposes.

[2]More information available on `www.tensorflow.org`

The network is a chain of convolutional and pooling layers, followed by fully connected layers. The number of layers of each type, as well as the number of units in each layer can be easily set through the `Recognizer` object. The input layer is a 3D tensor of size *width*×*height*×*channels*, which just assumes the pixel intensities of the images. In the default configuration, the first six hidden layers are an intercalation of three convolutional layers and three pooling layers, and the next hidden layer is a fully connected layer which starts with 512 units by default. The output layer is an *n-way* softmax, where *n* is the number of registered people. The softmax function is used to make the output of the last layer a probability distribution over the possible categories [6]. With the exception of the units on the output layer, all the other units apply the *Rectified Linear Function* as an activation function, which is simply equal to *max(0, z)*, with *z* being the unit's logit value.

Each of the convolutions has 16 feature detectors, each of them with a receptive field of size 10 × 10 over the output of the previous pooling layer (or the image pixels for the first convolution). Each of the pooling layers applies a max-pooling function to the convolution's output with a field of size 2 × 2 and a stride length of 2, both horizontally and vertically. With this pooling stride, every pooling layer halves the dimensions of the previous output.

Prior to the training, the network parameters have to be initialized. The unit weights are set to random values, drawn from a normal distribution with mean zero, and standard deviation equals to 0.1. The unit biases are set to zero on the convolution's units and 1 on the last two fully connected layers.

The training phase is performed using an initial learning rate of 0.01, which is decreased by 10% at each 500 training iterations. At each iteration, the cost gradient is estimated based on a batch of 32 samples from the training set, and the cost function used is the average cross entropy error [6]:

$$C = -\frac{1}{|B|}\sum_{x\in B}\sum_{i=1}^{n} t_i^x \ln y_i(x) \tag{3}$$

where *B* is the batch of samples of the current iteration, *x* is a training sample, *n* is the number of people registered, t^x is the one-hot label vector of the samplex, that is, a vector with 1 on the position corresponding to the correct person and zeros on all other positions, and *y(x)* is the output vector of the network for the simplex, that is, the output of the softmax layer. The overall network architecture can be seen on Figure 3.

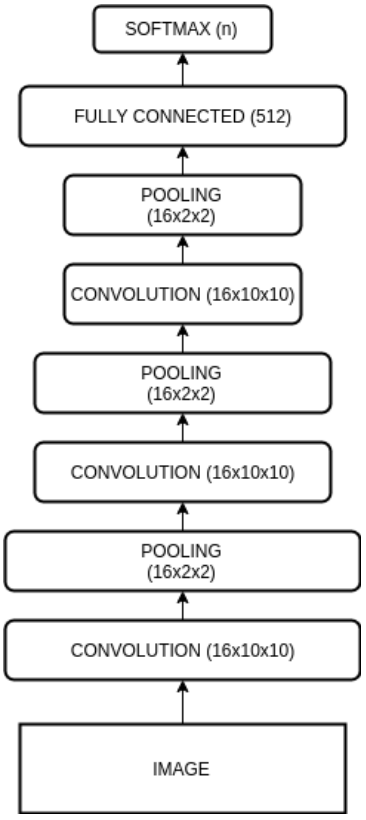

FIGURE 3– OVERALL VIEW OF THE DEFAULT NETWORK ARCHITECTURE

In order to improve the generalization capacity, it is applied an *L2 Penalty* [15] to all the network weights, with a penalty of 0.001, and it is applied dropout [11][7] on the fully connected hidden layers, with the units having a probability of being used of 0.3. The accuracy is estimated on the validation data periodically according to the parameter valid interval, and informed to the user.

For using the model to recognize new faces, the final configuration of the parameters is chosen to be the configuration on which the network had the best validation accuracy, not mattering how many iterations were performed after.

VI. EVALUATION

6.1 Dataset

The data used for the library test is the *Extended Yale Faces Dataset B*(**B+**)[3][5]. This dataset consists of 16380 pictures of 28 different people, with 585 pictures of each person, using different variations of pose and illumination. All the images were grayscale of size 640 × 480 pixels. Some samples of this dataset are shown in Figure 4.

[3]Dataset publicly available at `http://vision.ucsd.edu/content/extended-yale-face-database-b-b`

FIGURE 4 - SOME SAMPLES OF THE B+ DATASET

From this datasets, we selected 50 random images from each person to compose the test set. These 1400 images were only used for the final test of the library and were not used for any training or estimation purposes.

The recognizer was instantiated to use grayscale images of size 100 × 100, and expect a minimum of 700 images for each person. The remaining 535 images of each person were registered on the recognizer, which rescaled them to the appropriate size, and artificially generated more images to compose the 700 required images. The recognizer was instantiated with a validation proportion of 0.1, thus, 630 random images of each person were used for training and the remaining 70 images were used as the validation set to estimate the generalization capacity.

6.2 Training Phase

We then run the Gradient Descent algorithm for 3000 iterations, using the default library configuration: initial learning rate of 0.01, L2 penalty of 0.001, the probability of being used when using dropout of 0.3, and the default configuration for the overall network structure. The validation set was analyzed at each 100 iterations.

We recorded the training cost at each iteration, which is plotted on Figure 5. The graph is plotted logarithmically because the main variations occur on the beginning of training. It is possible to see that the network quickly found a low cost region on the parameter space, and then specialized to approximate the local minimum of this region.

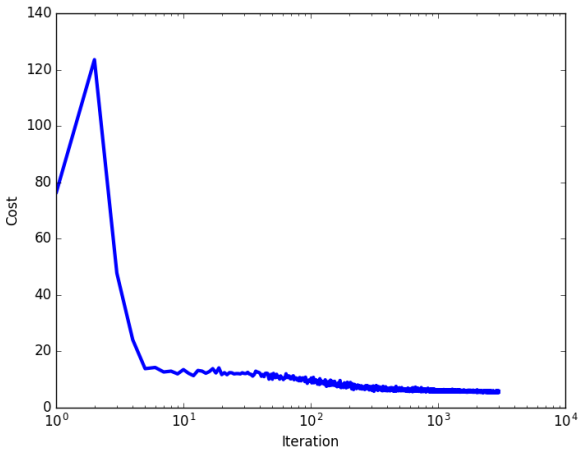

FIGURE 5 - COST RELATED TO THE RANDOM BATCHES OF 32 IMAGES DURING TRAINING ITERATIONS.

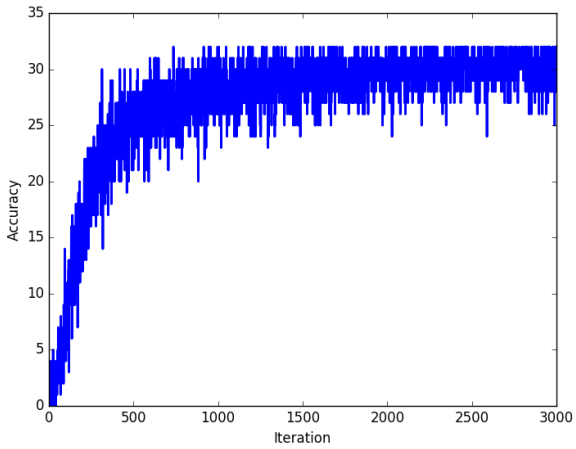

FIGURE 6–NUMBER OF CORRECTLY PREDICTED IMAGES RELATED TO THE RANDOM BATCHES OF 32 IMAGES DURING TRAINING ITERATIONS

On Figure 6, we can see the number of correctly predicted images. This shows that the network learned to recognize the images correctly as the cost lowered, meaning that the cost function truly captures the prediction accuracy. We considered a prediction to be correct if the network attributed the highest probability to the correct person.

On the cost (Figure 7) and accuracy (Figure 8) on the validation set, we can see that the network has extracted real patterns from the faces, being able to generalize the prediction to unseen data.

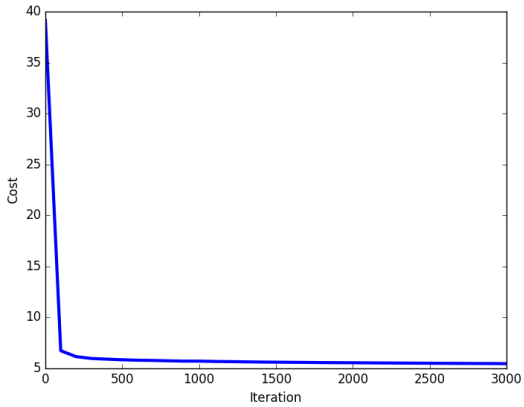

FIGURE 7 - COST FROM THE VALIDATION SET DURING TRAINING ITERATIONS

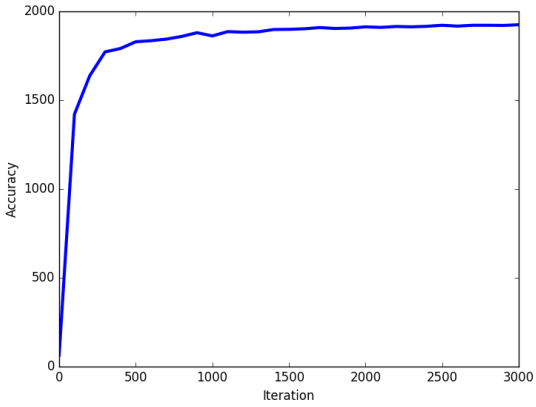

FIGURE 8 - NUMBER OF CORRECTLY PREDICTED IMAGES FROM THE VALIDATION SET DURING TRAINING ITERATIONS

The best validation accuracy occurred on the last validation, at the iteration number 3000 of the training. Then the parameter's configuration on this iteration was chosen to be the official model for future predictions.

As the final evaluation, we presented the trained library with the 1400 images reserved as the test set. If we considered a prediction to be correct if the network assigned the higher probability to the correct person, then the network correctly predicted 1398 of the 1400 images, an accuracy of 99.86%. If we considered a prediction to be correct if the network assigned a probability greater than 0.9 to the correct person, than the network correctly predicted 1374 of the images, an accuracy of 98.14%. However, if we consider this second metric, as the network has never assigned more than 0.9 to a wrong person, than the network did not classify any image erroneously. In these situations, the recognizer considers this face as unknown, which is an important feature on security applications.

VII. RELATED WORK

The authors in [11] used a CNN with dropout trained on a GPU for the *ImageNet* dataset from the LSVRC-2010 contest. The images were of high-resolution and belonging to 1000 different object classes. Their network achieved a top-1 and top-5 error rates of 37.5% and 17.0% respectively, a considerably improvement from the previous state-of-the-art models.

In [17], the authors presented a set of good practices for the development of CNNs for document recognition. It is explained the importance of the dataset size. It is also proposed a flexible architecture for document retrieval that is evaluated on the MNIST dataset [13]. As one of the contributions of our work, we have abstracted the process of artificial generation of data in order to transparently handle the possibility of few data samples.

In [4], it is proposed a model based on a committee of 7 deep CNNs trained on GPUs applied to handwritten digits and letters recognition. They reported an error rate of 0.27% for MNIST dataset [13]. In our work, we extended the application of CNNs to the faces domain, allowing the use images with much higher resolution than the 28 × 28 MNIST images.

In [8], the authors provided an evaluation of CNNs on large-scale video classification using a dataset of 1 million YouTube videos and 487 classes. They analyzed a number of approaches of extending the connectivity of CNNs to the time domain, in order to take advantage of the spatial-temporal information of video data.

In [14], the authors proposed CNN architecture for handling the task of face detection on images. In real world scenarios, this task is considered hard because of the many variations on distance, angle, pose, light, background color and objects, presence of more than one face, etc. The proposed architecture quickly discards large portions of background at low resolution and then carefully analyzes high resolution smaller portions. Our work is complementary with [14], in the sense that face detection is a commonly prior stage of face recognition.

In [9], it is done a research on different face recognition techniques for biometric systems. They showed the advantages of different NN approaches compared to other methods, as well as the current limitations. The contribution of our work is in the sense of providing a library abstracting the complexity of such techniques, as well as demonstrating, through a large scale experiment, the potential of CNNs on achieving high accuracy and not making mistakes on the prediction.

The authors in [16] used CNNs to build a mapping from face images to vector points, and then use the distance between these points as a direct measure of similarity between faces. They achieved a representation efficiency of 128 bytes per face.

In [3], it is proposed an open-source face recognition implementation called *Open Face* based on the work of[16]. They provided a web page with comparisons of accuracies of models[4] and concluded that face recognition is still an unsolved problem, with many issues yet to be handled, and accuracies from research papers have just begun to surpass human accuracies. One important contribution of our work over existing ones is the automatic generation of data, with the objective of compensates small number of pictures available, and thus, improving generalization capacity through better representative datasets.

VIII. CONCLUSION

In this work, we proposed a face recognition library in *Python*, with the objective of lowering the implementation complexity of face recognition features in applications in general. We provided the description of the main library features, as well as a description of the general architectural model of the internal CNN.

Despite the current existence of a similar software library [3], face recognition is still not a fully solved problem, with human level accuracies being surpassed only on the last few years. In this perspective, the main contributions of our work are: (i) we proposed CNN architecture for image recognition; (ii) we developed a general purpose face recognition library for python using the proposed architecture; (iii) we evaluated the library architecture on a large scale scenario using a public dataset of face images;

The proposed library achieved high accuracy on a large scale task of simple recognition. Also, when using a required confidence of 90%, the network did not classified any image erroneously, which is an important requirement for security systems. In summary, the results showed an excellent generalization capacity.

In future works, we intend to apply the library on real mobile apps and evaluate its performance. We also intend to further improve the library, investigating the relation between number of categories (people), amount of available data, and network capacity, with the objective of implementing a partially automatic generation of the model's architecture based on specific problems.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv preprint arXiv:1604467*.

[2] M. Abadi, P. Barham, J. Chen, et al., "Tensorflow: A system for large-scale machine learning," In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI), Savannah, Georgia, USA, 2016.

[3] B. Amos, B. Ludwiczuk, M. Satyanarayanan, "Openface: A general-purpose face recognition library with mobile applications," Techical report, CMU-CS-16-118, CMU School of Computer Science, 2016

[4] D. C. Ciresan, U. Meier, L. M. Gambardella, J. Schmidhuber, "Convolutional neural networks committees for handwritten character classification," In 2011 International Conference on Document Analysis and Recognition (ICDAR), IEEE, pages 1135-1139.

[5] A. Georghiades, P. Belhumeur, D. Kriegman, "From few to many: Illumination cone models for face recognition under variable lighting and pose," IEEE Trans. Pattern Anal. Mach. Intelligence, 2001, 23(6):643-660.

[6] I. Goodfellow, Y. Bengio, A. Courville, "Deep Learning," MIT Press, 2016, www.deeplearningbook.com

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

[8] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, "Large-scale video classification with convolutional neural networks," In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2014, pages 1725-1732.

[9] M. M. Kasar, D. Bhattacharyya, T.-h Kim, "Face recognition using neural network: A review," In International Journal of Security and its Applications, 2016, 10(3):81-100.

[10] A. Krizhevsky, G. Hinton, "Learning multiple layers of feature detectors from tiny images," 2009.

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

[12] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, 1998, 86(11):2278-2324

[13] Y. LeCun, C. Cortes, C. J. Burges, "The mnist database of handwritten digits," 1998.

[4] http://cmusatyalab.github.io/openface/models-and-accuracies/

[14] H. Li, Z. Lin, X. Shen, J. Brandt, G. Hua, "A convolutional neural network cascade for face detection," In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pages 5325-5334.

[15] M. A. Nielsen, "Neural Networks and Deep Learning," 2015, Determination Press.

[16] F. Schroff, D. Kalenichenko, J. Philbin, "Facenet: A unified embedding for face recognition and clustering," In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pages 815-823.

[17] P. Y. Simard, D. Steinkraus, J. C. Platt, et al., "Best practices for convolutional neural networks applied to visual document analysis," In ICDSR, volume 3, 2003, pages 958-962. Citeseer.