UNIVERSITY OF CAGLIARI

FACULTY OF SCIENCE

Major Project Proposal

Face Recognition Using a Deep Convolutional Neural Network and a Graph-Based Distributed Computation Framework:

"ArganRecogn"

Submitted by:
CHAOUA Ilyas
IGUIDER Walid

Submitted to:
Mr. Silvio Barra

Department of Computer Science
University of Cagliari

February 14, 2017

Abstract

The goal of this report is the presentation of our biometry and security course's project: Face recognition for Labeled Faces in the Wild dataset using Convolutional Neural Network technology with Graphlab Framework. Deep learning is the new big trend in machine learning. It had many recent successes in computer vision, automatic speech recognition and natural language processing. We will go through general ideas and structures of face recognition using deep learning, important issues to load, summarize and visualize the dataset, we will explain the critical techniques and algorithms, and finally give some predictions and present our results.

Contents

List of Figures

1 Introduction

Face Recognition went mainstream in the early 1960's when Helen Chan and Charles Bisson, worked on using the computer to recognize human faces (Bledsoe 1966a, 1966b; Bledsoe and Chan 1965). The first semi-automated system for face recognition required the administrator to locate features (such as eyes, ears, nose, and mouth) on the photographs before it calculated distances and ratios to a common reference point, which were then compared to reference data. In the 1970's, Goldstein, Harmon, and Lesk used 21 specific subjective markers such as hair color and lip thickness to automate the recognition. The problem with both of these early solutions was that the measurements and locations were manually computed. In 1988, Kirby and Sirovich applied principle component analysis, a standard linear algebra technique, to the face recognition problem. In 1991, Turk and Pentland discovered that while using the eigenfaces techniques, the residual error could be used to detect faces in images a discovery that enabled reliable real-time automated face recognition systems.
Recent progress in this area has been due to two factors:

1. End to End learning for the task using a convolutional neural network.

2. The availability of very large scale training datasets.

In this report, we focus on image-based face recognition. Given a image from LFW dataset, we'd like to detect faces of any person inside and know who he/she is. Towards this goal, we generally followed this procedure :

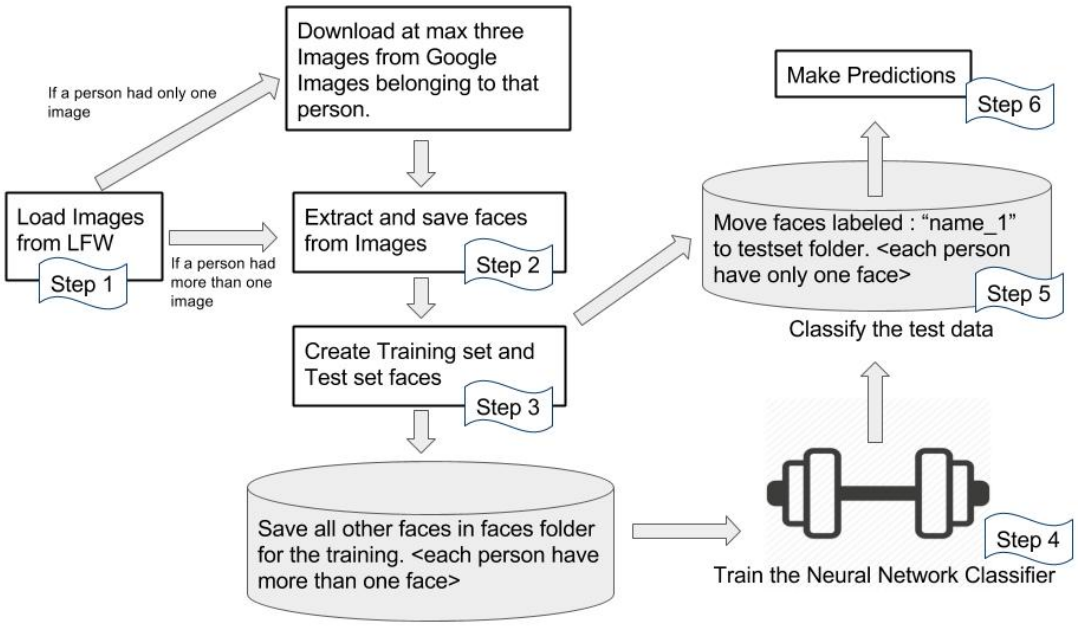


Figure 1: The process of the project.

2 Data Collection

In the world of face recognition, the dataset plays the role of raw material, for this reason, we choose to work with Labeled Faces in the Wild : LFW, a database of face photographs designed for studying the problem of unconstrained face recognition. The data set contains more than 13,000 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set.

Dataset	Identities	Images
LFW	5749	13233
Beginning Set	5751	13229
Ours:ArganRecogn	5751	25651

We proposed a procedure to create a reasonably large face dataset : ArganRecogn with the characteristic that each identity has more than three images whilst the Beginning Set contains 4072 identities with only one image (1679 identities have more than one image). To this end we downloaded more than 13987 new images from Google Images. The method for collecting images is writing with python using open source scripts available on the web. We employ this procedure to build a dataset with over 25651 images.

3 Face Detection

In this section we propose an open source method to efficiently detect and extract faces from an image given using OpenCV, the most popular library for computer vision. Originally written in C/C++, it now provides bindings for Python.
For face detection, the algorithm starts at the top left of a picture and moves down across small blocks of data, looking at each block, constantly asking, "Is this a face? ... Is this a face? ... Is this a face?" Since there are 6,000 or more tests per block, you might have millions of calculations to do, which will grind your computer to a halt. To get around this, OpenCV uses cascades.
Like a series of waterfalls, the OpenCV cascade breaks the problem of detecting faces into multiple stages. For each block, it does a very rough and quick test. If that passes, it does a slightly more detailed test, and so on. The algorithm may have 30-50 of these stages or cascades, and it will only detect a face if all stages pass. The advantage is that the majority of the pictures will return negative during the first few stages, which means the algorithm won't waste time testing all 6,000 features on it. Instead of taking hours, face detection can now be done in real time.
Though the theory may sound complicated, in practice it is quite easy. The cascades themselves are just a bunch of XML files that contain OpenCV data used to detect objects. You initialize your code with the cascade you want, and then it does the work for you.

Figure 2: Image of the identity : Johnny Deep

Figure 3: Face of the identity : Johnny Deep.

4 Splitting the Data

After applying our face detection algorithm based on OpenCV library, we loaded all the resulting data and started the data splitting. First we split the data into the following :

- Test data, which contains the images of all the identities such that we have one image by identity (5751 image).
- The other data contains at least one image by identity(19900 image).

Then we took 80% of the other data for the training the network and 20% for the validation.
The following figures represents information about the experimental test sets.

item	value	is exact
Length	1679	Yes
# Missing Values	0	Yes
# unique values	1684	No

Figure 4: Beginning Test set

item	value	is exact
Length	5733	Yes
# Missing Values	0	Yes
# unique values	5740	No

Figure 5: ArganRecogn Test set

The following figures represents information about the experimental training sets.

item	value	is exact
Length	12031	Yes
# Missing Values	0	Yes
# unique values	5247	No

Figure 6: Beginning Training set

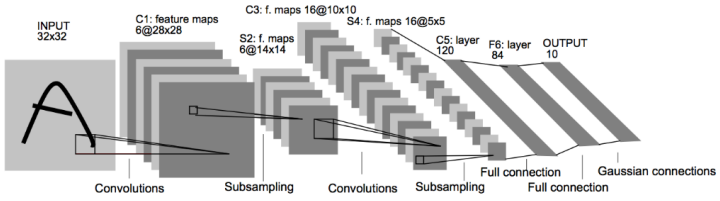
item	value	is exact
Length	19900	Yes
# Missing Values	0	Yes
# unique values	5743	No

Figure 7: ArganRecogn Training set

5 Network architecture and training

5.1 Convolutional Neural Network

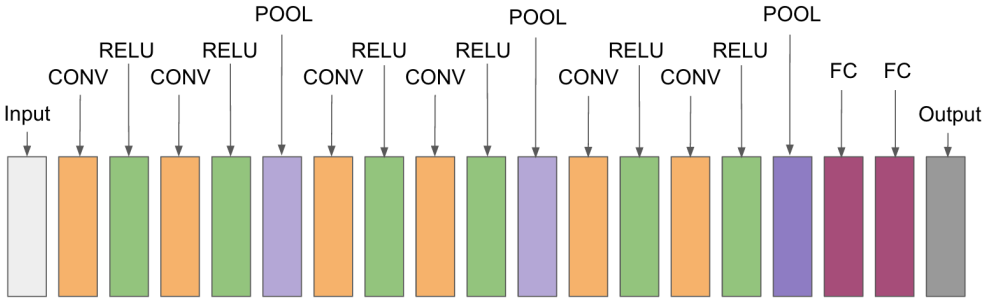
In this section, we explain the algorithms and tools for building and training the convolutional neural network using GraphLab, a Python library, backed by a C++ engine, for quickly building large-scale, high-performance data products. GraphLab provides machine learning algorithms including deep learning, boosted trees, and factorization machines and can be run the code on your laptop or in a distributed system, using a Hadoop Yarn or EC2 cluster.
Convolutional neural networks are a special type of feed-forward networks. These models are designed to emulate the behaviour of a visual cortex. CNNs perform very well on visual recognition tasks. CNNs have special layers called convolutional layers and pooling layers that allow the network to encode certain images properties.



CNN called LeNet by Yann LeCun (1998)

5.2 Convolutional Neural Networks Architecture

The simplest architecture of a convolutional neural networks starts with an input layer (images) followed by a sequence of convolutional layers and pooling layers, and ends with fully-connected layers. The convolutional layers are usually followed by one layer of ReLU activation functions.
The convolutional, pooling and ReLU layers act as learnable features extractors, while the fully connected layers acts as a machine learning classifier. Furthermore, the early layers of the network encode generic patterns of the images, while later layers encode the details patterns of the images.Note that only the convolutional layers and fully-connected layers have weights. These weights are learned in the training phase.[7]



Example of a CNN Architecture

5.2.1 Convolution Layer

This layer consists of a set of learnable filters that we slide over the image spatially, computing dot products between the entries of the filter and the input image. The filters should extend to the full depth of the input image. For example, if we want to apply a filter of size 5x5 to a colored image of size 32x32, then the filter should have depth 3 (5x5x3) to cover all 3 color channels (Red, Green, Blue) of the image. These filters will activate when they see same specific structure in the images.[7]

5.2.2 Pooling Layer

Pooling is a form of non-linear down-sampling. The goal of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. There are several functions to implement pooling among which max pooling is the most common one. Pooling is often applied with filters of size 2x2 applied with a stride of 2 at every depth slice. A pooling layer of size 2x2 with stride of 2 shrinks the input image to a 1/4 of its original size.[7]

5.2.3 Training the CNN

The general workflow of training a neural network using GraphLab involves three steps:

1. Create a convolutional neural network : CNN.

2. Tweak the CNN parameters (number of iterations, target class, metrics...) so that they are appropriate for the desired task.

3. Load the classifier and give it the CNN as a parameter to begin the training using the given network architecture and parameters.

During the data splitting, we dedicated 78% of the whole dataset for the training and validation. Thus we did a random split of the resulting dataset such that 80% of the data is aimed to be the training set whereas the remaining 20% is for the validation. As a first experience of network training we tried to give the classifier 300 iteration, which gave us, after the validation of the training, an accuracy of about 65%. Then we started increasing the number of iterations until we reached a maximum accuracy at the 600th iteration which was about 87%.
In a first time we trained the network CNN using the Beginning training set with a learning rate of 0.001, a momentum of 0.9 and the metric: "accuracy,recall@2" for 600 iterations. After that we experimented an other training for the CNN with the same parameters using ArganRecogn training set. In the next section, we will present our results for the both attempts.

6 Test Set

Now that we have a trained model, we can use it to make predictions. To do this, we can use the ArganRecogn test set (previously described) which consists of 5751 faces and present 22% of global data. This set contains faces that have never been subjected to the network.
In the following we first test several variations of the proposed models and training data using the beginning test set based on LFW database and ArgagnRecogn test set to compare the performance. Briefly, these are the results I have achieved using MacBook 12"-Intel HD Graphics 5300 1536MB for beginning test set after 600 iterations:

Training time (sec) : 23697.8544

Training accuracy	Validation accuracy	Training recall@2	Validation recall@2
0.8764	0.1527	0.9405	0.1799

Test results for beginning test set

These are the results I have achieved using MacBook 12"-Intel HD Graphics 5300 1536MB for ArganRecogn test set after 600 iterations:

Training time (sec) : 63363.5553

Training accuracy	Validation accuracy	Training recall@2	Validation recall@2
0.7312	0.0865	0.8342	0.1023

Test results for ArganRecogn test set

7 Conclusion

In this work we have used a large scale dataset, with small label noise, while keeping down the number of manual annotation. The main idea was to train a classifier to rank the data presented to the annotators. This procedure has been developed for faces, but is evidently suitable for other object classes as well as fine grained tasks. Then as another contribution we have showed that a well trained deep convolutional neural network can achieve very good results comparing to the state of the art.

Certificate of approval

We found the students Ilyas Chaoua and Walid Iguider to be hardworking, skilled, bonafide and ready to undertake any commercial and industrial work related to their field of study.

Professor. Silvio Barra
(Project Supervisor)
.....

Copyright

Department of Computer Science
University of Cagliari
Cagliari, Italy

References

[1] http://cs231n.github.io/convolutional-networks/
karpathy@cs.stanford.edu.

[2] https://turi.com/products/create/docs/graphlab.toolkits.deeplearning.html?highlight=deep

[3] https://www.robots.ox.ac.uk/vgg/publications/2015/parkhi15/parkhi15.pdf.

[4] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures,neural networks: Tricks of the trade. *Springer Berlin Heidelberg*, 437-478, 2012.

[5] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[6] Yann LeCun. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86.11*, 2278-2324, 1998.

[7] Adil Moujahid. http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/. 2016.