Hand Gesture Classification

Christian Braz, Junior Ovince, Rahul Seti
George Washington University
Department of Data Science

Abstract

Hand gesture recognition is the process of recognizing meaningful expressions of form and motion by a human involving only the hands. There are plenty of applications where hand gesture recognition can be applied for improving control, accessibility, communication and learning. In this work, we present our results in classifying twenty six hand signs for the letters of the alphabet. We conduct experiments using different convolutional neural networks architectures and report the performance of each of them.

Keywords: gesture recognition, machine learning, convolutional neural networks, deep learning

Introduction

Hand gestures provide a complementary modality for expressing ideas. It can be used in many different contexts to augment the human-machine interaction. For instance, it can help deaf people to communicate, send commands to a intelligent house system, and so on. There are mainly two ways to capture such interaction, via Data-Glove and vision approach Strezoski, Stojanovski, Dimitrovski, and Madjarov (2018). The Data-Glove based approach collects data from sensors attached to a glove mounted on the hand of the user. It is accurate and captures only the necessary information, which minimizes the amount of data. On the other hand, due to the necessary hardware, it is inconvenient and even infeasible in many scenarios. The vision approach has the advantage of the hardware simplicity, but requires a big amount of image data to be processed in the attempt to come up with a solution that is insensitive to adverse conditions like lightning, background invariant, subject, and camera independent.

The American Sign Language (ASL) is a visual language that incorporates gestures, facial expressions, and body movements. Hand signs are the foundation of the language. Many signs are iconic, meaning the sign uses a visual image that resembles the concept it represents. The alphabet is an important series of signs. Some hand signs for letters resemble the written form of the respective letter. The American Manual Alphabet (AMA) is a manual alphabet that augments the vocabulary of American Sign Language (Figure 1).

When you use the hand signs for letters to spell out a word, you are "fingerspelling", and in AMA this is one-handed. The American Manual Alphabet have often been used in deaf education and is composed by 26 hand positions to designate letters from A to Z, and 10 positions to represent the digits from 0 to 9.

In this work we address the vision solution to the problem of classifying the 26 hand gesture positions for fingerspelling the letters of the alphabet. We hope that with this work we can contribute somehow to the development of practical computational innovative solutions for human-computer interaction. The rest of this paper is organized as follows. Related Work presents a review of general hand gesture recognition approaches and also works related to recognize the sign language. Literature Review briefly explains the main machine learning techniques that has been used so far to tackle this problem. More specifically, we describe four Convolutional Neural Network (CNN) models: LeNet5, AlexNet, VGG Net and GoogleLeNet. In the Methods section, our experiment is described, detailing the training approach, the dataset, and the models. Finally, we conclude our work and discuss future development in the Conclusion section.

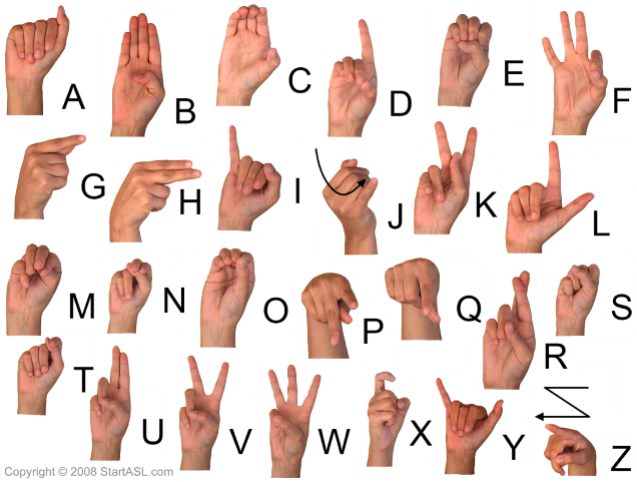

Figure 1. American Sign Language alphabet.

Related Work

Hand gestures recognition without having to wearing any extra device have been the target of many researches in the past years. Its applicability to enhancing human-computer interaction has use in many different domains such as video games, augmented reality, intelligent house system control, and in any case involving a 3D scenario. Trigueiros, Ribeiro, and Reis (2012) makes a comparative study of four algorithms (k-Nearest Neighbour (k-NN), Naïve Bayes (NB), Artificial Neural Network (ANN) and Support Vector Machines (SVM)) for static gesture recognition. They concluded that ANN had the best performance.

Strezoski et al. (2018) point out how the advent of specialized hardware like the GPU's boosted some machine learning algorithms, giving rise to the Deep Learning architectures and paved the design of models with deep architectures. One such architecture is Convolutional Neural Networks (CNN), which have proven to be superior for many problems

involving image processing. In their paper, they evaluate different efficient CNN architectures designed by industry leaders such as Google, IBM and Microsoft, in the problem of identifying hand signs from the Marcel dataset.

Computer recognition of sign language is an important research problem for enabling communication with hearing impaired people. Garcia and Viesca (2016) propose an ASL fingerspelling translator based on CNN. They build their solution upon the pre-trained GoogleLeNet using transfer learning and develop an ASL recognition system in real time to translate a video of a user's ASL signs into text. Bheda and Radpour (2017) tackle the problem of automatic sign language recognition trying to make use of images with depth-sensing technology. These images, besides the regular pixels, have contour and depth information.

Literature Review

In this section we briefly describe the technologies related to this work.

Convolutional Neural Network

A Convolutional Neural Network (CNN) consists of a number of convolutional and pooling layers, optionally followed by fully connected layers. Convolution layers take inner product of the linear filter and the underlying image portion, usually followed by a nonlinear activation function at every local portion of the input. The resulting outputs are called feature maps. Its architecture is designed to take advantage of the 2D structure of an input image. The input to a convolutional layer is a $m\ x\ m\ x\ r$ image where m is the height and width of the image, and r is the number of channels. A channel is a conventional term used to refer to a certain component of an image. A RGB image will have three channels – Red, Green and Blue – and we can imagine them as three 2D-matrices stacked over each other, one for each color, each having pixel values in the range 0 to 255. On the other hand, a grayscale image, has just one channel. The convolutional layer will have k filters (or kernels) of size $n\ x\ n\ x\ q$ which convolve the image to produce k feature maps of size m−n+1. Thus, the purpose of convolutions is to extract features from the image. Each feature map can then be processed again by the pooling operator which reduces the dimensionality of each feature map while retains the most important information. The output from the convolutional and pooling layers represent high-level features of the input image. The higher levels layers are typically fully connected layers which are traditional Multi Layer Perceptron (MLP) whose purpose is to use these features for classifying the input image into various classes using a softmax activation function. These components are the basic building blocks of every Convolutional Neural Network and we can see a scheme of them in Figure 2[1].

The LeNet architecture

In the seminal paper of 1994, Yann LeCun defined the first CNN named LeNet5. Among other things it was important for the insight that image features are distributed

[1] https://cdn-images-1.medium.com/max/1255/1*XbuW8WuRrAY5pC4t-9DZAQ.jpeg

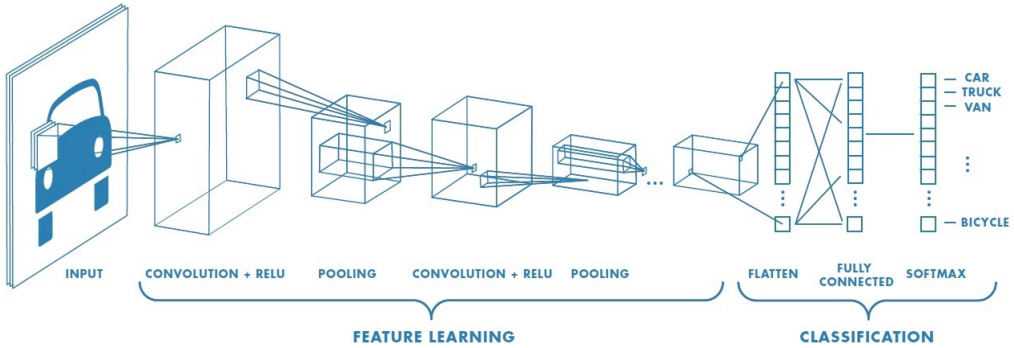

Figure 2. Convolutional Neural Netowrk scheme.

across the entire image, and convolutions, whose parameters could be learned, are an effective way to extract similar features at multiple location. At that time this was an revolutionary idea, and contrasted to using each pixel as a separate input of a large multi-layer neural network. LeNet5 explained that images are highly spatially correlated, and using individual pixel of the image as separate input features would not take advantage of these correlations. It was specially designed to recognize handwritten digit. It starts with an image of 32x32x1, then in the first step, six 5x5 filter with stride 1 resulting in a 28x28x 6 feature map. Then, an average pooling with a filter width of 2 and stride of 2 reduce the dimension by factor of 2 and end up with 14x14x6. Then another convolutional layer with sixteen 5x5 filter connected to another pooling layer yields a 5x5x16 map. The next layer is a fully connected layer with 120 nodes. The previous layer is 400 (5*5*16) dimensional is connected with 120 neurons. Finally, another layer with 84 nodes connect to the output layer with 10 neurons to predict the 10 digits.

AlexNet

AlexNet, winner of the ILSVRC-2010[2], used the insights provided by LeNet and build a much larger neural network that could be used to learn much more complex features. It consisted in a 11x11, 5x5, 3x3, convolutions, max pooling, dropout, and ReLU activations after every convolutional and fully-connected layer. AlexNet was trained on two Nvidia Geforce GTX 580 GPUs which in turn allowed the use of larger datasets and also bigger images.

VGG Net

In Simonyan and Zisserman (2014) the authors describe their work where they propose employing much more layers with much smaller filter sizes. “We fix other parameters of the architecture, and steadily increase the depth of the network by adding more convolutional layers, which is feasible due to the use of very small (3×3) convolution filters in all layers. Max-pooling is performed over a 2×2 pixel window, with stride 2 .”. Stacking three 3x3 convolutional layers without pooling between them, has an effective receptive field of using

[2]ImageNet Large Scale Visual Recognition Challenge

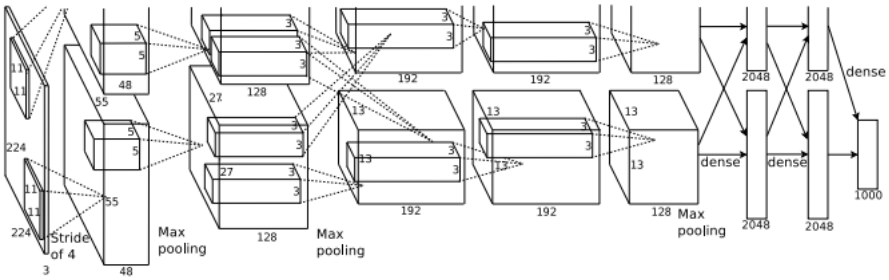

Figure 3. AlexNet.

just a single 7x7 one, with the advantage of incorporating three non-linear rectification layers instead of a single one, which makes the decision function more discriminative while reducing the number of parameters. They train over 224x224 RGB images using networks weights from 11 to 19 layers architectures, varying from 133 to 144 million parameters. They claim that they have the best general approach for image feature extraction and that its model "out performs GoogleLeNet in terms of the single-network classification accuracy".

GoogleLeNet

In the aim of creating even deeper architectures while keeping the computational cost constant, the Google team (Szegedy et al. (2014)) devised a 22-layer deep architecture inspired by the Network In Network idea.

Network In Network (NIN) (Figure 4) is an approach proposed by Lin, Chen, and Yan (2013) in order to increase the representational power of neural networks. The authors argue that conventional CNN "implicitly makes the assumption that the latent concepts are linearly separable, and then use the generalized linear model (GLM) as convolutional filters". They claim that this does not allow a good abstraction level, and that replacing GLM with a more potent nonlinear function approximator can improve the abstraction capability of the model. Thus, in the NIN, the GLM is replaced with a general nonlinear function approximator, which in this case is the Multilayer Perceptron (MLP), known as a universal function approximator. The feature maps are obtained by sliding the MLP over the input in a similar manner as CNN and then the resulting output is fed into the next layer. The name is due by the fact that there are "micro networks", each consisting of multiple fully connected layers with nonlinear activation functions, which are composing elements of the overall deep network. The output layer of these micro MLP's define the number of channels of the next layer. They called this new type of layer *mlpconv*.

The NiN method can be viewed as additional 1×1 convolutional layers: "The cross channel parametric pooling layer is also equivalent to a convolution layer with 1x1 convolution kernel.". Given a 3D input volume, some transformation will be applied for each element of the grid over its depth component (cross channel). The output size of each transformation defines the depth component of the next volume. We can easily note that, besides the gain of having a more flexible way of combining cross channel elements, we can also achieve a significant reduction of the final volume overall dimension.

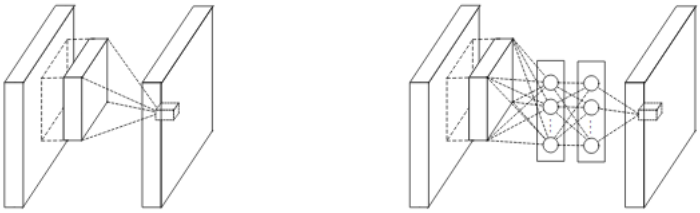

Figure 4. Comparison of linear convolution layer (left) and NiN layer (right).

GoogleLeNet makes use of this idea to create the Inception modules. Roughly speaking, each inception module is a stack of different feature maps, all with the same width and height, and using 1×1 convolutions to compute reductions before the expensive 3×3 and 5×5 convolutions. "This allows for not just increasing the depth, but also the width of networks without significant performance penalty." claim the authors. Thus, in GoogleLeNet, NiN is used mainly as dimension reduction to remove computational bottlenecks that would otherwise limit the size of networks.

They were the winner of the ILSVRC 2014 competition. GoogleLeNet, which is the name of the team chosen for the competition and is a homage to Yann LeCuns pioneering LeNet 5 network, achieved a top-5 error rate of 6.67%. This is very close to human level performance.

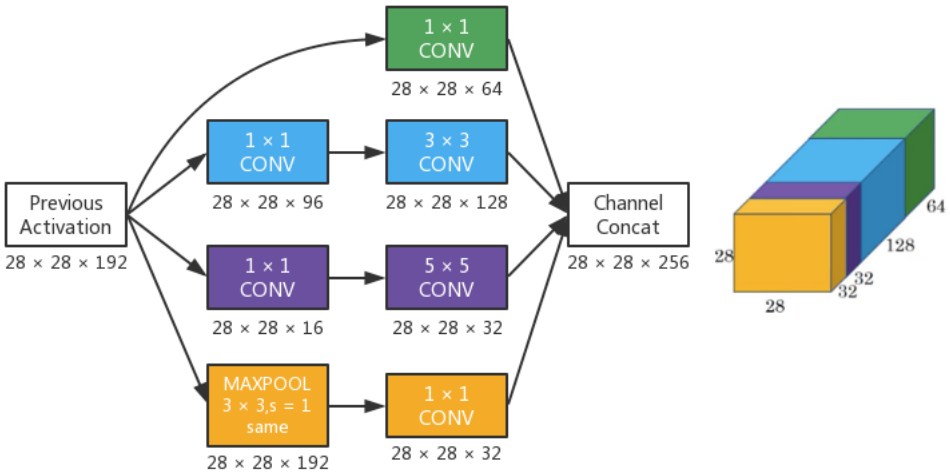

Figure 5. Inception module.

Methods

The method used to accomplish this project is divided into three main phases: pre-training, training, and post training. We implement various specific tasks in each phase.

A. Pretraining

During the pretraining phase, we proceed to the data preprocessing and the choice of the neural networks. The data preprocessing includes 2 main components:

- Normalization: As the dataset was uploaded using ImageFolder package from pytorch, we applied a normalization of 0.5 in order to have all the values between minus and positive one.
- Split of the dataset: The dataset is organized in one folder including all the images folder, we had to split it into three sets: train, validation, and test set. A rate of 70/15/15 was applied to obtain respectively the different sets. We used sklearn train/test split module in order to keep the 29 classes balanced among the sets.

As the dataset is on image pattern recognition, convolutional neural network is the best neural network classification algorithm to use. As presented in the context, we use some of the most applied CNN architecture for image classification. We design a baseline model and adapt two existing CNN models: LeNet and AlexNet as this dataset does not seem to be a complex one. Table 1 summarizes the model architecture used:

	Baseline	**LeNet**	**AlexNet**
Number of conv layers	2 conv (5,5)	2 conv (5,5)	5 conv (11,5,3,3,3)
Number of fully connected layers	1 (250 neurons)	2 (120,84 neurons)	2 (4096,4096)
Layer output transfer function	LogSoftmax	Linear	Linear
Estimated number of parameters	11M	4.2 M	103 M
Estimated number of feature maps	230	114	250 k

Table 1
Summary of the Network architecture used.

B. Training

In the training phase we use two main categories of tools. The first category is the fundamental one and includes the definition of the training algorithm and the performance index function. The second one is to improve the overall performance of the models in order to reduce possible cases of overfitting and underfitting.

For the performance index, we use cross entropy as we want to find a probability distribution for this multiclass image classification problem. For the training algorithm, we use Adam optimizer, which is a robust variant of the general stochastic gradient algorithm.

The second category includes practices to improve the convergence of the algorithm and the general performance of the model. So, we use Xavier Normal initialization for the convolutional layers and a scheduler to update the rate when the losses are constant. We also use dropout nodes and early stopping to prevent overfitting. In general, the different models were trained using a total of 20 epochs.

C. Post training

For the post training, we use the classic metrics such as confusion matrix, precision, recall, f-score, AUC and ROC. We also calculate the error parameters and the processing time in order to assess the quality of the different model performances.

We use pytorch and tensorboard to run all the models.

Results

This section presents the result of each model.

1. Baseline

After running 20 epochs on the baseline model, we obtained the following trends for the losses and accuracies (Figure 6):

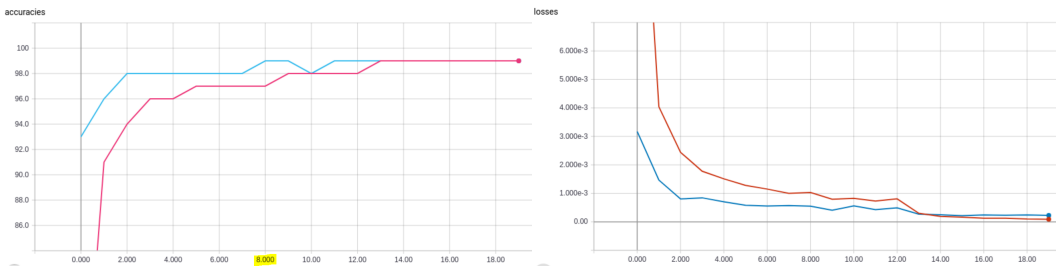

Figure 6. Left, accuracy trends for the baseline model over 20 epochs. Right, Loss trends for the baseline model over 20 epochs.

The trends indicate that the model performs well at the 10th epoch and does not learn anything from that point one and even tend to overfit. We run the model with epoch and compare the result as presented in Table 2:

2. LeNet

The same trend analysis on the loss and accuracy on the LeNet model shows that the model tends to overfit at the 4th epoch. So, we compare the result of models with 20 and 4 epochs and conclude that four is the number of epochs to adopt (Figure 7).

The summary of our results is presented in Table 3.

Metrics	Initial	Final
Precision	0.99595	0.9916846
Recall/Accuracy	0.99593	0.9916475
F-score	0.99594	0.9916470
AUC	All greater than 0.9999	All greater than 0.9998
Processing Time	36 minutes	19 minutes
Error	std = 0.000649517 mean = 0.000641096	std= 0.0007859 mean= 0.00096642
Updated at epoch	12 and 18	None
Stopped at epoch	20	10

Table 2
Baseline Model Summary Results

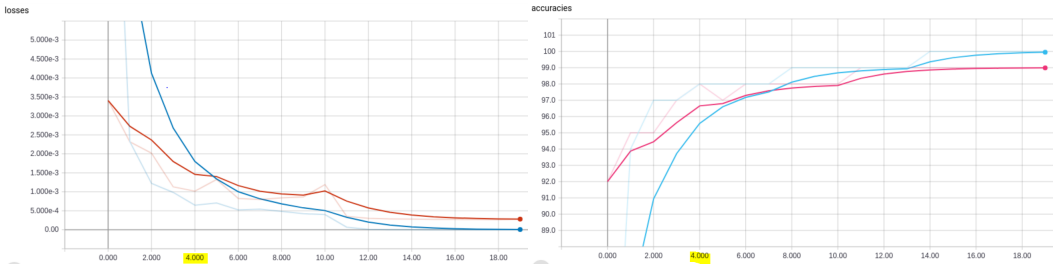

Figure 7. Left, loss trends for the LeNet model over 20 epochs. Right, accuracy trends for the LeNet model over 20 epochs.

3. AlexNet

The AlexNet is designed to be implemented for complex model with a dataset of hundreds even thousands of classes. However, the LeNet results show that our problem is simpler and a two convolutional layer network with thousands of parameters should be able to fit this classification problem. Furthermore, the AlexNet is time consuming. For the two main reasons, we decided to run the AlexNet architecture on only five epochs. Even then, the model tends to overfit and takes 45minutes to complete the learning. Table 4 brings more details.

4. Model Testing

We use the baseline and LeNet models to test their performance on the test set. The trend stays the same as the baseline model gives a higher accuracy with 10 epochs and the LeNet model performs better in time. The trade-off time/accuracy is a classic one in data science. One might prefer the LeNet model for a translation application where time is more important. Other might prefer the baseline model for a tagging application such as Facebook where the time is less important. Our sense indicates that the model is a combination of the two architectures (Table 5).

Metrics	Initial	Final
Precision	0.996112	0.976701
Recall / Accuracy	0.996091	0.975785
F-score	0.996091	0.975852
AUC	All greater than 0. 9801	All greater than 0.9603
Processing Time	30 minutes	7 minutes
Error	std = 0.00081509 mean = 0.0009158	std= 0.0008129 mean= 0.002219
Updated at epoch	None	None
Stopped at epoch	20	4

Table 3
LeNet Model Summary Results

Metrics	Initial
Precision	0.989630
Recall / Accuracy	0.989272
F-score	0.989259
AUC	All greater than 0.9949
Processing Time	44 minutes
Error	std = 0.0004457 mean = 0.0007320
Updated at epoch	None
Stopped at epoch	5

Table 4
AlexNet Model Summary Results

Metrics	Baseline- 10 epochs	LeNet – 4epochs
Precision	0.9902369	0.9794684
Recall / Accuracy	0.9901915	0.9786973
F-score	0.9901984	0.9787579
AUC	All greater than 0.9998	All greater than 0.9596
Processing Time	18 minutes	7 minutes
Error	std = 0.0007808 mean = 0.000998	std = 0.000858553 mean = 0.00224975
Classes with lowest accuracy	S, A, B, del	G, P, Space, B

Table 5
Model testing summary.

Conclusion

In this project, we focus on comparing different CNN architectures as it is the first assignment to do in any machine learning application. The different model results show that the dataset requires a not complex architecture such as AlexNet to classify the images. A two convolutional layer network seems to perform well. The baseline model outperforms

the LeNet model in performance whereas the latter only needs seven minutes to classify the images at almost 98% accuracy. A combination of the two architectures can help find the ground truth of this dataset. One might also consider an ensemble model for this pattern recognition. We did very few hyperparameters tuning. A small change in the number of neurons and the batch size can improve the result. Other hyperparameters such as weight regularization can also be fine tuned.

References

Lin, M., Chen, Q., & Yan, S. (2013). Network in network. *CoRR*, *abs/1312.4400*. Retrieved from http://arxiv.org/abs/1312.4400

Bheda, V., & Radpour, D. (2017). Using deep convolutional networks for gesture recognition in american sign language. *CoRR*, *abs/1710.06836*. Retrieved from http://arxiv.org/abs/1710.06836

Garcia, B., & Viesca, S. (2016). Real-time american sign language recognition with convolutional neural networks. In *In convolutional neural networks for visualrecognition.* Stanford.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., ... Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, *abs/1409.4842*. Retrieved from http://arxiv.org/abs/1409.4842

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, *abs/1409.1556*. Retrieved from http://arxiv.org/abs/1409.1556

Strezoski, G., Stojanovski, D., Dimitrovski, I., & Madjarov, G. (2018). Hand gesture recognition using deep convolutional neural networks. In G. Stojanov & A. Kulakov (Eds.), *Ict innovations 2016* (pp. 49–58). Cham: Springer International Publishing.

Trigueiros, P., Ribeiro, F., & Reis, L. P. (2012, June). A comparison of machine learning algorithms applied to hand gesture recognition. In *7th iberian conference on information systems and technologies (cisti 2012)* (p. 1-6).