FINGERPRINT CLASSIFICATION

Capstone Report

Rafael Tibães

Udacity Machine Learning Engineer Nanodegree

January 2, 2018

Definition

In biometrics there are two main operations: verification (1:1) and identification (1:N). Verification is the process that confirms a person's identity based on their biometrics. Identification is the process to identify a person's identity based on its biometrics, i.e. searching the correspondent biometric model in the database. A naive identification algorithm would perform the verification procedure for each person's biometric model stored in the database. To reduce the computational costs, a clever approach would compute indexes for each biometric model in the database. With a properly indexed database, the identification process would require to perform the verification algorithm only in a reduced subset of the database.

There are a couple of biometrics modalities capable to accurately identify an individual, such as the face, fingerprint and iris. In this work we focus on fingerprints acquired by imaging sensors. Fingerprints are composed by ridges (darker pixels) and valleys (lighter pixels), where each fingerprint has a unique composition of ridges and valleys. Despite the unique composition, all fingerprints follow one of the five fundamental types, i.e. ridge patterns, described by *NIST* [2]: i) Arch, ii) Left Loop, iii) Right Loop, iv) Tented Arch, and v) Whorl. If we could compute the fundamental type of every fingerprint, this information could be used to perform the database indexation. Using just this indexation we would be able to perform the identification task (1:N) almost five times faster!

I am particularly interested in this problem because I need a solution for a complex newborn fingerprint identification platform that I'm working on. In a real world application, it would be easy to have a huge database, which would take a very long time to compute a naive identification algorithm. Proprietary technology is capable to achieve a high speed identification feedback; however, they are not suitable for our needs.

Inspired by the Udacity *dog breed* assignment and also by the work of Kai et Al. [1], which uses a Convolutional Neural Network (CNN) to identify the pattern of small fingerprint patches targeting forensics applications, in this work we trained a CNN to compute the most basic fingerprint indexation: the ridge pattern. The proposed network contains pairs of convolution and pooling layers, transforming the spatial information in a more abstract set of features capable to distinguish the ridge patterns.

This is a multi-class classification problem with well-established metrics for accuracy, as presented by Equation 1. Given a database of labeled fingerprint images, containing the fundamental type information, the algorithm should give the same answer as the database labels. Accuracy is a good measurement for our problem, because each correct prediction avoids a lookup over the entire database. High accuracy means less computing power needed to perform a biometric identification.

Equation 1: Accuracy definition for multi-class classification.

$$Accuracy = \frac{Correct\ Predictions}{Total\ Number\ of\ Examples}$$

Analysis

For this work, the ideal dataset should contain fingerprint images paired with notations about the fundamental type information. *NIST Special Database 4* [2][3] was developed for this purpose and is free for research. In this dataset all images are stored in grayscale *PNG* files having dimension of *512 x 512* pixels. Figure 1 presents an example of how a grayscale image can be represented digitally. The fingerprints images were acquired under varying conditions, so there are images with good and others with poor quality. The quality is affected mostly by the skin's humidity and the pressure applied on sensor. As result, a poor quality image has blurred borders, making it harder to distinguish ridges from valleys.

The *NIST* dataset contains **4000** fingerprint images, uniformly distributed along the five fundamental types, i.e. there are **800** fingerprint images of each class: i) (A) Arch, ii) (L) Left Loop, iii) (R) Right Loop, iv) (T) Tented Arch, and v) (W) Whorl. This is an important feature, because for classification tasks it is fundamental to balance the dataset before training. Figure 2 to Figure 6 are fingerprints samples from this dataset, to present the characteristics of each fundamental type, and Figure 7 presents the data distribution.

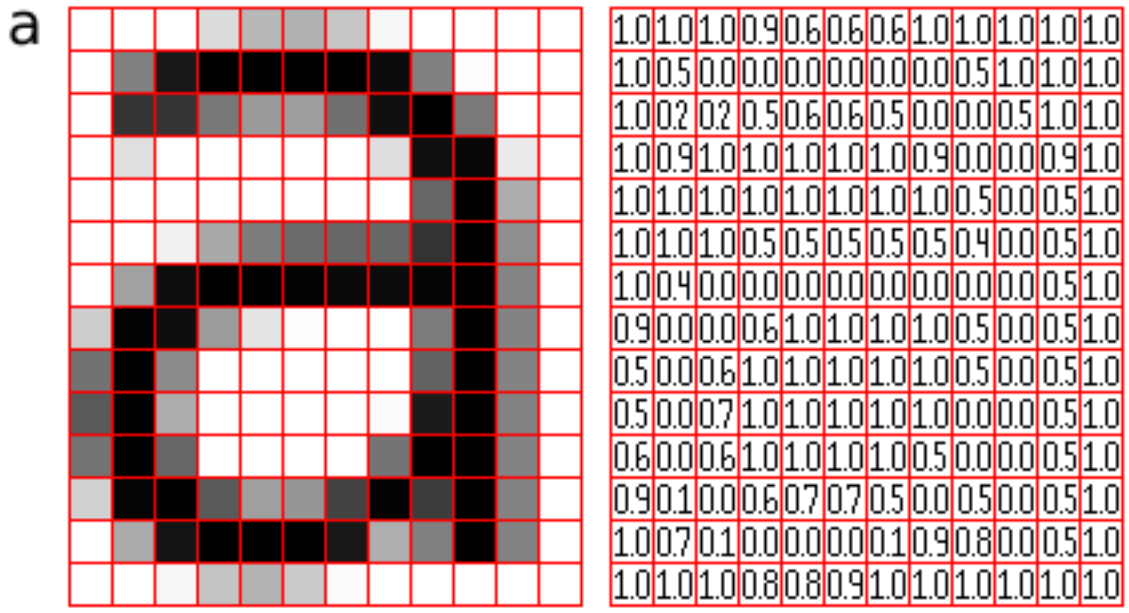

Figure 1: Digital image representation, example of the character 'a' representation [4].

Figure 2: Fingerprint of type Arch (A). Notice that the central part of the fingerprint is composed by smooth lines. This sample also presents a partially occluded fingerprint, because it was not acquired with focus on its center.

Figure 3: Fingerprint of type Tented Arch (T). Notice that the central part of the fingerprint is composed by a strong peak, in contrast to the Arch type. This sample also presents a poor quality image, as we can see that the ridges are blurred.

Figure 4: Fingerprint of type Whorl (W). Notice that the central part of the fingerprint is composed by a cyclic pattern.

Figure 5: Fingerprint of type Right Loop (R). Notice that the central part of the fingerprint is composed by a curve on the right.

Figure 6: Fingerprint of type Left Loop (L). Notice that the central part of the fingerprint is composed by a curve on the left.

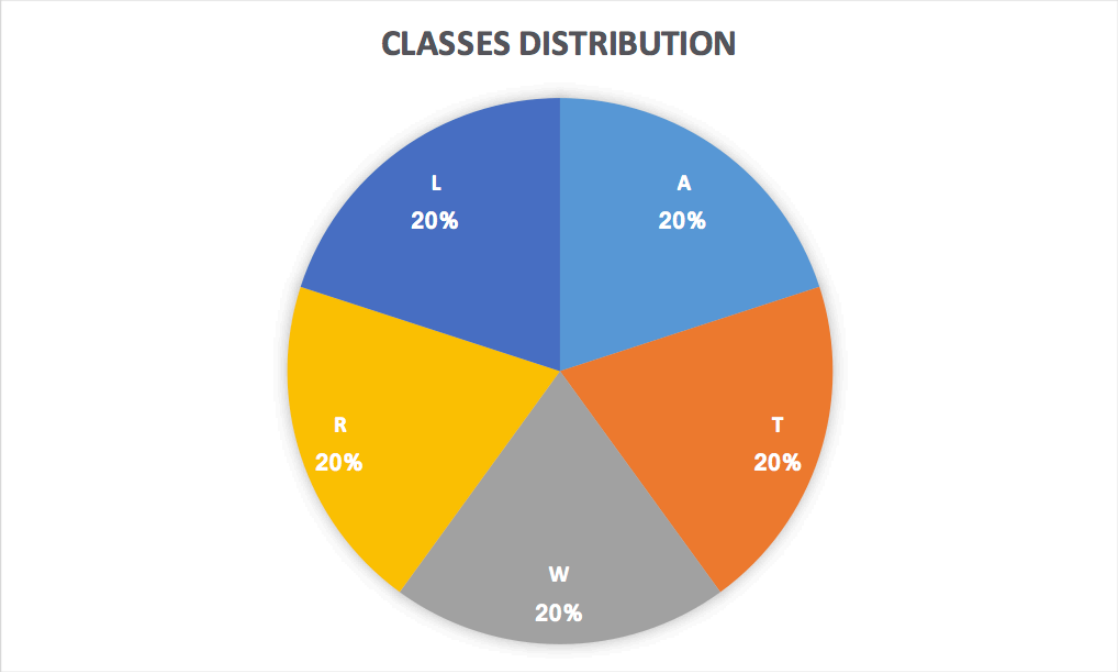

Figure 7: Graphic of classes distribution on NIST Dataset: all samples are evenly distributed among the five classes.

Image classification can be achieved by different algorithms, such as Perceptron, Support Vector Machines (SVM) and Convolutional Neural Networks (CNN). In recent competitions of image classification, it is common for CNN to be ranked as the most accurate algorithm, using complex networks such as *AlexNet*, *Inception* and *ResNet*. In this work we adopted CNN for its robustness and accuracy, building a simple network inspired by the work of Kai et Al. [1].

A CNN can be composed by different kind of layers. The proposed network relies on these: convolutional, pooling, dropout and dense layers. Convolutional layers act like image processing filters, transforming the input to explore specific features, such as edges and corners. Pooling layers are another type of filters, but these focus on merging or reducing the data. A sequence of multiple convolutional & pooling layers is capable to learn more abstracts features as longer the sequence gets. For instance, initial layers learn useful edge and corners filters, but the last layers learn discriminant ridge patterns. Dropout layers are used to drop some features during training as a

measure to avoid overfitting. Finally, dense layers are fully-connected nodes, where all features are combined to find the most discriminant combination. The output layer usually is a dense layer where each node represents one class.

The algorithm will be evaluated accordingly to its accuracy to correctly classify the fundamental type of each fingerprint image in the *NIST Dataset* [2]. To avoid overfitting, the dataset will be split in three subsets: training set, validation set and testing set. For the accuracy measurement it will be used the testing set. The accuracy represents the percentage of fingerprint fundamental type that were corrected labeled by the algorithm according the dataset labels, as described by Equation 1.

As I haven't found in literature an algorithm for this exact purpose, I will compare the proposed approach with a random guess and a simple neural network. In this manner, giving that we have five classes, our approach must achieve an accuracy above 20%, otherwise it would perform worse than picking a random value. Also, it must beat the simple neural network to justify its complexity. The simple neural network, denoted here as the vanilla neural network, is composed by the input layer, two dense layers and the output layer. The input layer uses the Flatten method to convert a 2D matrix (image) to a 1D vector, losing a lot about the pixels' spatial relationship.

Methodology

Usually, an image classification algorithm requires pre-processing of the image data. In this work, this pre-processing is composed by three modules: i. conversion of images to tensors, ii. conversion of database text to classification labels, and iii. database split into training, validation and testing subsets.

The first module, conversion of images to tensors, is a simple procedure that read the dataset images and reorganize the information to be compatible with Keras/Tensorflow *Tensor* data structure, in the format *(512, 512, 1)*.

The second module, conversion of database text to classification labels, is based on a **script** that parses the dataset text to one of these five characters: 'A', 'W', 'T', 'R' and 'L'. For example, the sample text: "**`Gender: M Class: W History: f0001_01.pct W a0591.pct`**" results to **`W`**. However, a character is not proper for classification, so we perform one hot encoding over these labels.

The last pre-processing module, database split, uses the Keras function **`train_test_split`** to split the samples into 70% for training and 30% for testing, using a random factor for picking samples. The training set is split again into 80% for training and 20% for validation, also using a random factor. So, looking over the entire dataset we have 56% for training, 14% for validation and 30% for testing.

Before training, it is applied data augmentation to increase the amount of samples and also avoid image memorization. Only rotation was used, simulating fingerprints captured with an inclined sensor. Flips must be avoided for data augmentation because a fingerprint fundamental type depends on features disposition in the image, e.g. the type 'R' is a flipped version of the type 'L'.

The proposed classification algorithm is a simple convolutional neural network, composed by three pairs of convolution and max-pooling layers, that transforms the spatial information to a more abstract feature space. These convolution layers are followed by a polling and a fully-connected layers to connect and select these features. A dropout layer is followed to avoid over-fitting. Finally, another fully-connected layer with only five nodes and a softmax activation is used to compute the final classification label. This architecture is illustrated by Figure 8.

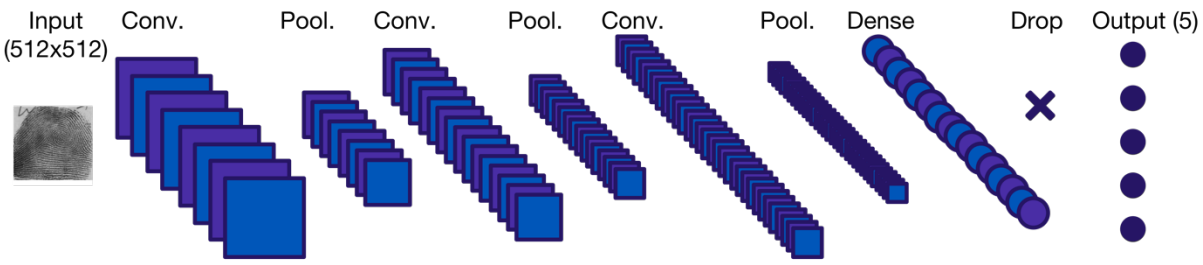

Figure 8: Proposed CNN architecture, composed by three pairs of convolution and pooling layers, followed by a dense layer, a dropout layer and the output layer with only five nodes.

To refine the approach, it was performed small changes in some architecture's parameters. The first one was the dimensionality of the output space of each convolution layer, picking the best of these combinations: a. (8, 16, 32), b. (16, 32, 64), and c. (16, 16, 16). After choosing the best dimensionality combination, it was evaluated the batch size, followed by the learning rate. All these experiments were performed with 20 epochs using the checkpoint technique to store the best model, i.e. skipping the persistence of an epoch results when not achieving accuracy improvement. The best model was evaluated again in a longer run of 50 epochs. The criteria to select each parameter was the accuracy on the testing set.

During the training I found some complications dealing with GPUs. The first limitation was the over-heating of my laptop that has a NVIDIA GTX 1070 and it was not capable of handling more than 20 epochs. To make it run longer, I switched the experiments to my workstation using a NVIDIA GTX 1060 and a far better cooling system. The 1060 has only half the memory of a 1070 GPU, which was not enough to compute using the default batch size. Reducing the batch size to half I was able to perform the experiments using 50 epochs. To better handle this situation I rent a cloud service with a NVIDIA K80, which is capable to deal with any of my situations, but definitively this is not cheap. Even with the computing power of expensive GPUs I got long training times, making it harder to iterate over parameters.

Another complication was dealing with data augmentation. Despite being a powerful tool to increase the amount of samples and minimize the risk of data memorization, for this problem the data augmentation tool from Keras must be parametrized carefully. In this work I kept only the rotation parameter, because some other parameters as `featurewise_center` and `featurewise_std_normalization` make the classifier performs like a random guess. Also, image flipping can result in misclassification, because a fingerprint image of the class 'R' is a flipped image of the class 'L'. Finally, the last complication that I found was a failure in the Keras documentation, where they do not inform that the parameter steps_per_epoch is mandatory for the function fit_generator when using image augmentation. I will contact the Keras maintainers to confirm if this is really an issue or if I misunderstood something.

Results

Assuming that a random guess achieves 20% accuracy, the vanilla neural network does not improve the classification task. In fact, analyzing its resulting confusion matrix, presented by Figure 9, this network performs the same as random guess.

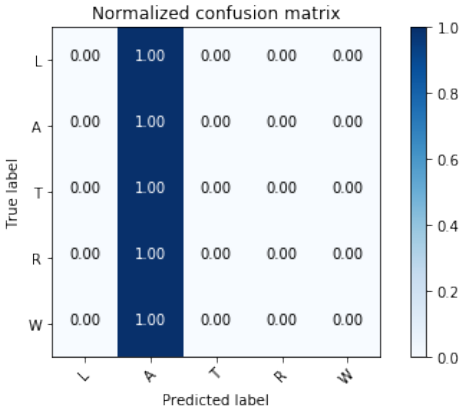

Figure 9: Confusion matrix of the vanilla neural network. It can be noticed that this network always picks the same class, being this as bad as random guess.

The proposed solution achieved up to 66% accuracy to classify the samples of the testing subset. Interestingly, the best accuracy was achieved by the model trained by 20 epochs instead of the one trained by 50 epochs, both under the same architecture and parameters. The later achieved 56% accuracy. I believe that this behavior was caused by the random nature of the online data augmentation and the training itself. This result is three times better than the random guess and the vanilla neural network. However, there is room for improvement.

Analyzing the loss/accuracy plot of the Figure 10 we can see that the model loss and accuracy didn't achieved a convergence even after 50 epochs. Maybe running even longer we could achieve better accuracy, however this is a costly task that I will pursue in a future work. There is no evidence of overfitting, because in both plots the validation and testing behaves almost linearly.

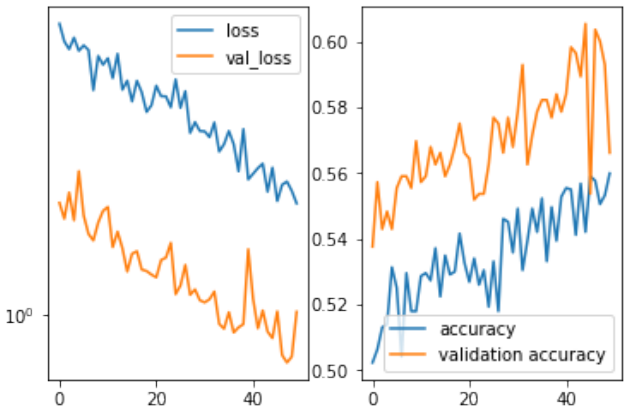

Figure 10: Training and validation loss/accuracy plots of the final CNN model trained with 50 epochs.

The resulting confusion matrix, presented by Figure 11, shows that the class 'W' has the best performance. On the other hand, type 'T' has the worst, being predicted wrongly as 'A', 'L' or 'R'. These classes also got misclassified as 'T'. This makes sense, as the class 'T' share similarities with all the other classes except by the class 'W'. I was expecting that the class 'R' would be classified as 'L', and vice-versa, however this did not happen.

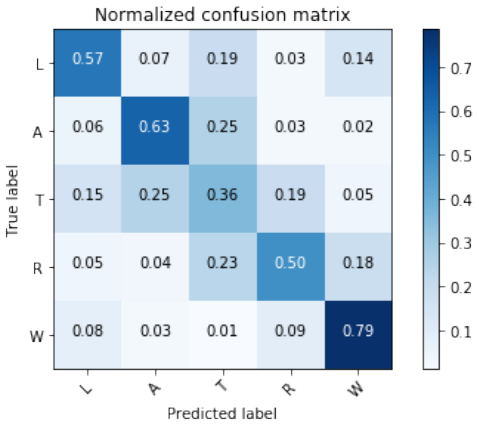

Figure 11: Confusion matrix of the proposed CNN. Notice the misclassification of the class 'T', distributed along 'L', 'A' and 'R'.

With just 66% accuracy it is already a great improvement for the identification task. Given that the naive approach represents 100% of time, our algorithm takes **`(66% * 20%) + ((100% - 66%) * 100%) = 47.2%`** of time, which represents a speed up of 2.12 times, i.e. two times faster. In contrast, a random guess achieves speed up of just 1.19 times. The graphic of Figure 12 compares the identification using naive, random guess, vanilla NN, CNN and CNN refined.

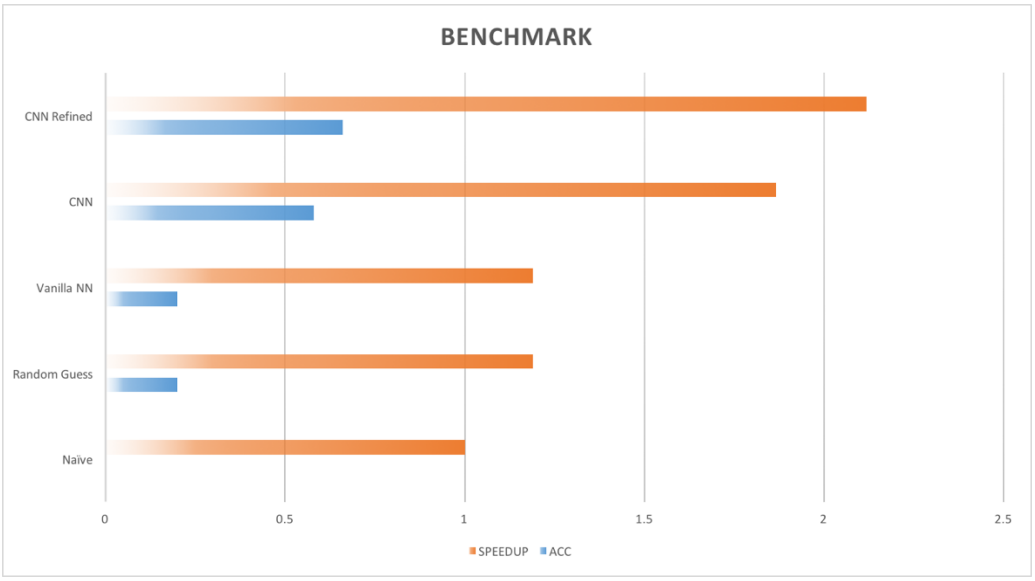

Figure 12: Identification performance using naive, random guess, CNN with 20 epochs and CNN with 50 epochs.

Conclusion

In this work it was used a small convolutional neural network composed by few convolutional and max-pooling layers followed by fully-connected layers for the training of an image classifier, targeting the fingerprint image classification into one of the five fundamental types described by *NIST*. The proposed network achieved up to 66% accuracy, which is above the random guess of 20% and the vanilla neural network used as baseline, which performed as bad as the random guess. I found interesting the fact that a very simple convolutional network is capable to achieve a decent classification accuracy. However, a more robust network requires a lot of 'trial and error', demanding a great amount of computing resources.

As a free-form visualization of the results achieved by the proposed model, Figure 13 presents examples of correctly classified images, one image for each class. The class 'T' is the most problematic one, having very similar features with other classes. The Figure 14 presents examples of misclassifications got for the class 'T'. This class demands most of the attention for a future work.

This project intent to be part of a proprietary fingerprint solution. For this reason, I plan to keep working on improvements using different techniques such as increasing the network complexity or even by transfer-learning based on a robust network, i.e. AlexNet, Inception, ResNet, etc. For faster identification more complex classifiers needs to be used, as bag-of-features based on the most discriminant features, learned by unsupervised clustering.

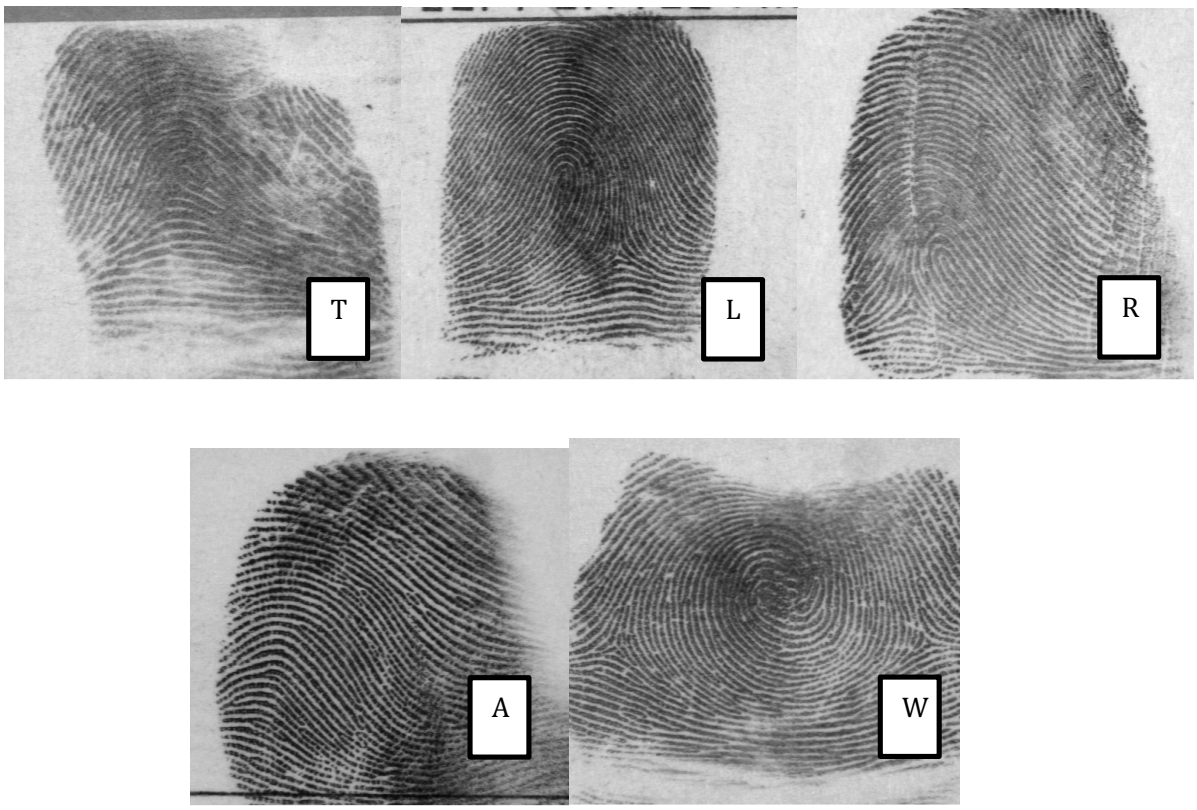

Figure 13: Samples of fingerprint types 'T', 'L', 'R', 'A' and 'W' correctly classified.

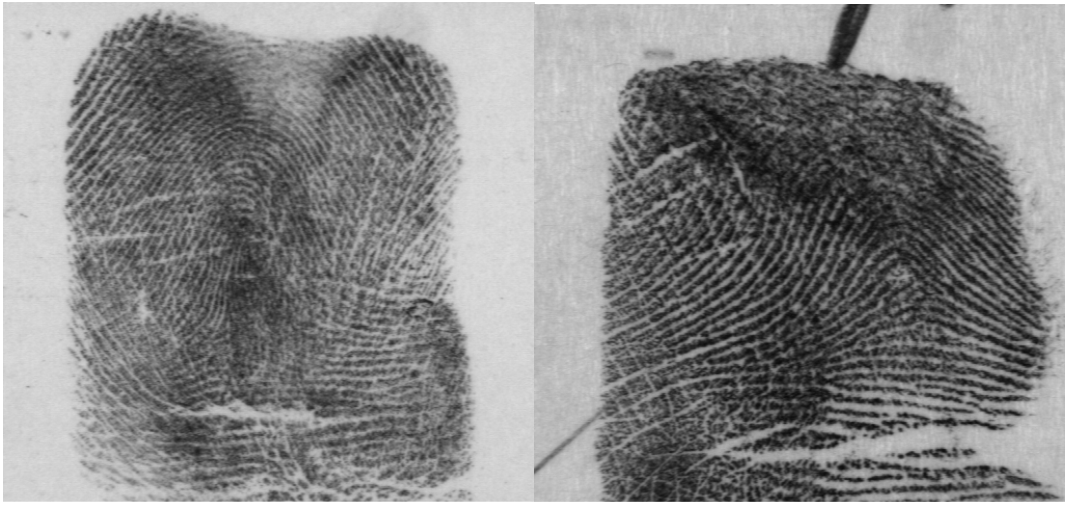

Figure 14: Type 'T' misclassified as 'L' (left) and 'A' (right). Notice that the left image is blurred in a very important region of the image for the classifier (in the middle of the image).

References

[1] Cao, Kai, and Anil K. Jain. "Latent orientation field estimation via convolutional neural network." Biometrics (ICB), 2015 International Conference on. IEEE, 2015.

[2] **https://www.nist.gov/srd/nist-special-database-4** (31/12/17)

[3] **https://www.nist.gov/itl/iad/image-group/resources/biometric-special-databases-and-software** (31/12/17)

[4] **http://pippin.gimp.org/image_processing/chap_dir.html** (31/12/17)