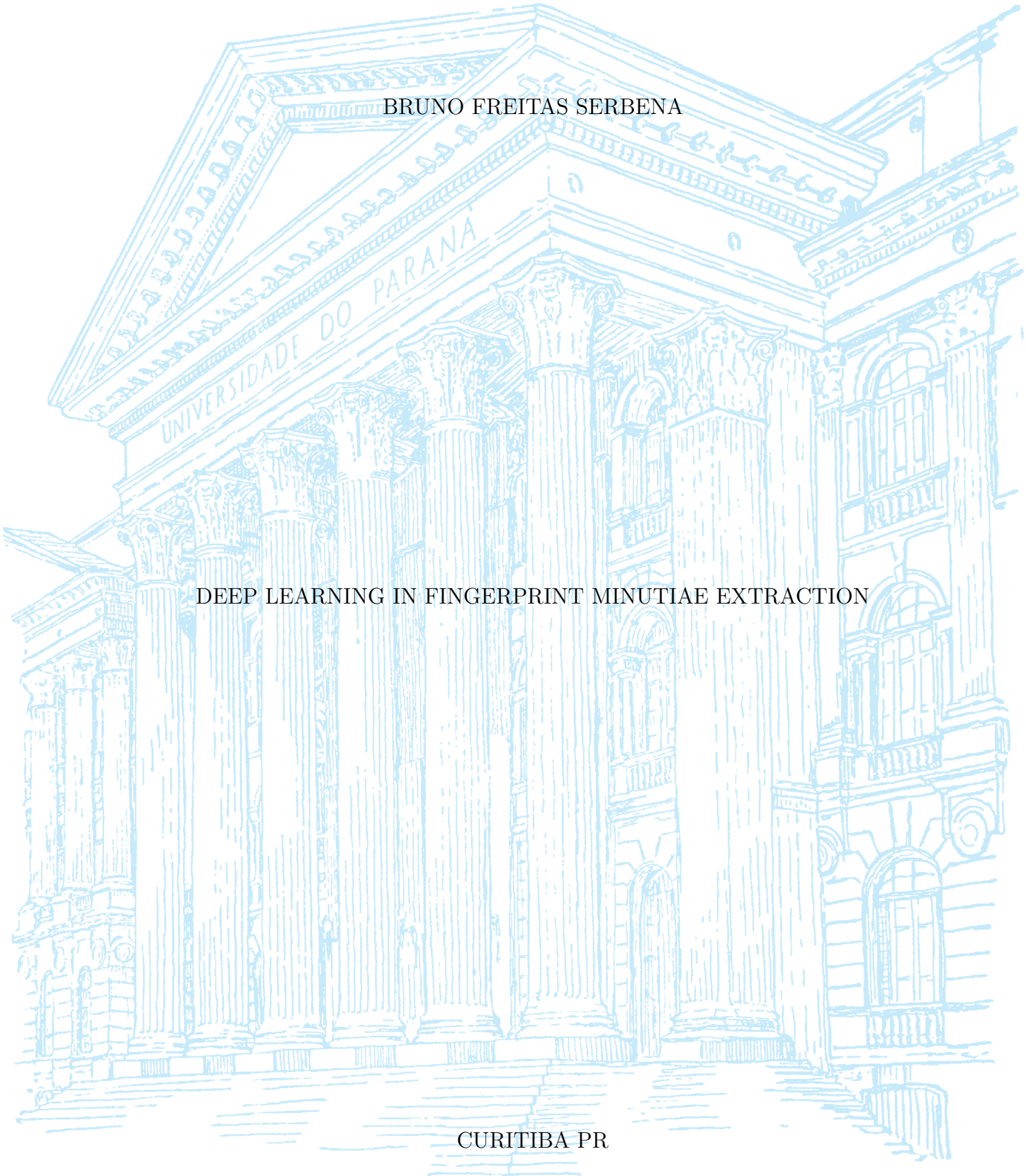
UNIVERSIDADE FEDERAL DO PARANÁ

BRUNO FREITAS SERBENA



DEEP LEARNING IN FINGERPRINT MINUTIAE EXTRACTION

CURITIBA PR

2019

BRUNO FREITAS SERBENA

DEEP LEARNING IN FINGERPRINT MINUTIAE EXTRACTION

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação.*

Orientador: David Menotti Gomes.

CURITIBA PR

2019

Ficha catalográfica

Substituir o arquivo `0-iniciais/catalografica.pdf` pela ficha catalográfica fornecida pela Biblioteca da UFPR (PDF em formato A4).

Instruções para obter a ficha catalográfica e fazer o depósito legal da tese/dissertação (contribuição de André Hochuli, abril 2019):

1. Verificar se está usando a versão mais recente do modelo do PPGInf e atualizar, se for necessário (`https://gitlab.c3sl.ufpr.br/maziero/tese`).
2. conferir o Checklist de formato do Sistema de Bibliotecas da UFPR, em `https://portal.ufpr.br/teses_servicos.html`.
3. Enviar email para "`referencia.bct@gmail.com`" com o arquivo PDF da dissertação/tese, solicitando a respectiva ficha catalográfica.
4. Ao receber a ficha, inseri-la em seu documento (substituir o arquivo `0-iniciais/catalografica.pdf` do diretório do modelo).
5. Emitir a Certidão Negativa (CND) de débito junto a biblioteca (`https://www.portal.ufpr.br/cnd.html`).
6. Avisar a secretaria do PPGInf que você está pronto para o depósito. Eles irão mudar sua titulação no SIGA, o que irá liberar uma opção no SIGA pra você fazer o depósito legal.
7. Acesse o SIGA (`http://www.prppg.ufpr.br/siga`) e preencha com cuidado os dados solicitados para o depósito da tese.
8. Aguarde a confirmação da Biblioteca.
9. Após a aprovação do pedido, informe a secretaria do PPGInf que a dissertação/tese foi depositada pela biblioteca. Será então liberado no SIGA um link para a confirmação dos dados para a emissão do diploma.

Ficha de aprovação

Substituir o arquivo 0-iniciais/aprovacao.pdf pela ficha de aprovação fornecida pela secretaria do programa, em formato PDF A4.

To Mary, for being the smartest person in this chaotic world.

AGRADECIMENTOS

Thank you Menotti for being the best teacher.
Thank you my friends for being weird.
Thank you Emilia for being there to support me and keep me going.

RESUMO

Detecção de minúcias é crucial para sistemas de reconhecimento automático de digitais. Minúcias são pequenas características formadas pelos cumes da impressão digital, como o fim de um cume ou uma bifurcação. O estado da arte de detecção de objetos em computação é todo realizado por redes neurais convolucionais (CNNs). Além disso, a pesquisa em *Deep Learning* melhorou o desempenho desses detectores por uma grande margem nos últimos anos; com técnicas de *design* novas, e mais rápidas, como os regressores de caixa delimitadora nas redes YOLO e SSD. Levando isso em conta, este documento propõe uma adaptação da rede YOLO para realizar detecção de minúcias com classificação de tipo. Alguns trabalhos existentes, MENet, FingerNet e MinutiaeNet, são bem precisos, mas nenhum classifica os tipos de minúcias e são mais lentos no tempo de processamento quando comparados a YOLO. Com as minucias detectadas, o desempenho conseguido em *matching* de impressões digitais usando o algoritmo do SourceAFIS é competitivo com a detecção interna do SourceAFIS; são iguais no ponto de Taxa Positiva Verdadeira (TPR) 84,5% e Taxa Falsa Positiva (FPR) 15%. Mas o SourceAFIS domina com maior TPR em valores de menor FPR na curva ROC. Mesmo assim, a abordagem proposta possui uma área maior que o SourceAFIS na curva. Com uma técnica de extração de ângulo aprimorada, o desempenho parece promissor, já que este resultado não utilizou nenhuma informação de ângulo. Como indicação, a detecção em si teve precisão alcançada 86,67% e *recall* de 86,09% utilizando um limiar de 0,2 no conjunto de dados FVC2004 é comparável aos resultados do estado da arte; Dado que o MinutiaeNet (Nguyen et al., 2018) atingiu 85,9 % de precisão e 84,8 % de *recall*. Apesar de a rede não extrair informação de angulos, ela extrai informações de tipo, que nenhum dos outros sistemas referenciados aqui tentam fazer.

Palavras-chave: Redes Neurais Convolucionais. Deep Learning. Extração de minúcias. Reconhecimento de impressões digitais.

ABSTRACT

Minutiae detection is crucial to automatic fingerprint recognition systems. These are small features formed by the ridges of the fingerprint, such as the end of a ridge or bifurcation. The state of the art of automatic object detection in computing are all performed by Convolutional Neural Networks. Additionally, research in Deep Learning improved the performance of these detectors by a large margin in the last few years; with new and faster, design patterns such as bounding-box regressors in YOLO and SSD. As such, this document proposes a YOLO adaptation to minutiae detection with type classification. Some existing work MENet, FingerNet and MinutiaeNet are very accurate, but none classifies minutiae types and are slower in processing time compared to YOLO. With the detected minutiae, the final fingerprint matching performance using SourceAFIS' algorithm competes with SourceAFIS' own internal detection performance; they are equal in the point True Positive Rate (TPR) 84.5% and False Positive Rate (FPR) 15%. But SourceAFIS' detection dominates with higher TPR in values of lower FPR in the ROC curve. Even so, the proposed approach has a higher area than SourceAFIS in the curve. With an angle extraction technique the performance looks promising, since this result did not use any angle information in the templates given to SourceAFIS. As indication, the detection itself achieved a precision of 86.67% and recall of 86.09% using a 0.2 threshold in the FVC2004 dataset is comparable to state of the art results; Given that MinutiaeNet (Nguyen et al., 2018) achieved 85.9% precision and 84.8% recall. Even if our precision doesn't take angle correctness into account (MinutiaeNet does), it extracts type information, which none of the other systems referenced here do.

Keywords: Convolutional Neural Networks. Deep learning. Minutiae extraction. Fingerprint recognition.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

DINF	Departamento de Informática
PPGINF	Programa de Pós-Graduação em Informática
UFPR	Universidade Federal do Paraná
mAP	mean Average Precision
APs	Average Precision on small scale objects (COCO)
CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit (an activation function)
SPP	Spatial Pyramid Pooling
FPN	Feature Pyramid Network (Lin et al., 2017a)
AUC	Area Under Curve
FAR	False Acceptance Rate
FPR	False Positive Rate
FRR	False Rejection Rate
GAR	Genuine Acceptance Rate
TPR	True Positive Rate
EER	Equal Error Rate
FM3	Fingerprint Manual Minutia Marker (Kayaoglu et al., 2013)
FPS	Frames Per Second

LISTA DE SÍMBOLOS

μ	mean
σ^2	variance

SUMÁRIO

1 INTRODUCTION

Fingerprint recognition is usually executed in two steps, minutiae extraction to create templates and template matching. This document will focus on the extraction of fingerprint minutiae using YOLO (Redmon e Farhadi, 2018), a state of the art object detection Neural Network. Additionally, the background of the most influential Neural Networks detectors will be studied to contextualize recent improvements in the field and the existing work in minutiae detection.

1.1 CHALLENGE

Automatic fingerprint recognition is one of the most studied fields in biometrics (Jain et al., 2016). The main methods of fingerprint recognition utilize minutiae, which are major features of a fingerprint formed by its ridges.

(a) Type: ending (b) Type: bifurcation

Figura 1.1: Minutiae examples from FVC2004DB1A image 1_1

Figures 1.1(a) and 1.1(b) are examples of minutiae of type: ending, and bifurcation respectively. The extraction of minutiae consists of given an input image of a fingerprint, outputting relevant information about each minutiae, for example: coordinates, type, and orientation. Some automatic extraction systems don't output type or orientation. This is because after the extraction, a matcher is employed (some matchers don't use some information). The matcher uses this extracted data to score a set of minutiae against each other. Which information that the matcher uses is implementation dependent. There is a lack of robust minutiae type classifiers (Prabhakar et al., 2003) so some fingerprint recognition algorithms found in the literature ignore the type of minutiae. A matcher like Minutiae Cylinder Code (Cappelli et al., 2010) uses coordinates, and orientation, but not the type. While SourceAFIS (Vazan, 2009) uses all three.

Fingerprint minutiae, because of imperfections originating from and lack of contact or pressure, lack of ink, and rolled impressions, are hard to detect with classical skeleton-based methods (Zhao e Tang, 2007) (Khan, 2011) with good precision and recall. These approaches utilize, for example, multiple minutiae detected in proximity to each other to detect and exclude spurious minutiae.

A bottleneck in automatic fingerprint recognition systems is the extraction of the minutiae. Particularly in latent fingerprints, for example in the context of criminal forensics, in which imperfections are accentuated.

This document studies the use of techniques of machine learning for automatic fingerprint minutiae extraction. Specifically the use of convolutional neural networks which recently obtained impressive results in image recognition, e.g. achieving superhuman performance in Microsoft's convolutional neural networks (He et al., 2015).

1.2 MOTIVATION

Recently a lot of progress has been made in object detection tasks utilizing deep learning. YOLOv3-608 (Redmon e Farhadi, 2018) achieved 33.0 mAP (mean Average Precision) on COCO (Lin et al., 2014). The COCO dataset has 1.5 million object instances within 330 thousand images, having 80 object categories. While RetinaNet-101-800 (Lin et al., 2017b) achieves 37.8 mAP with a tradeoff in speed, taking 3.8 times longer than YOLOv3-608. Additionally, on a easier dataset, PASCAL VOC2007 (Everingham et al., 2007) with 24.64 thousand object instances within 9,963 images, having 20 object categories. The network SSD (Liu et al., 2016) achieved 74.3 mAP and Faster R-CNN (Ren et al., 2015) 73.2 mAP.

The motivation is to utilize this recently developed advances in deep learning in other detection tasks. Rather than people and objects in videos and images in for example the COCO dataset (Lin et al., 2014), try to adapt them to see how well they perform in the detection of details and minutiae, such as fingerprint minutiae.

1.3 PROPOSAL

The idea is to analyze recent deep learning networks explored for object detection that have had successful evaluations on databases such as COCO (Lin et al., 2014) and PASCAL VOC (Everingham et al., 2012) and adapt one of them to test how well they perform in this specific task. Also to analyze how they compare with the classical approach of SourceAFIS, an open source project.

This analysis will consider two factors. precision and recall of extracted minutiae when comparing to a annotated test dataset, using a distance tolerance for true positives (so it's not only pixel perfect matches). And matching performance, using the matcher from SourceAFIS (SourceAFIS can extract minutiae and match separately). We convert our extraction results to the template of the matcher and test it against the match performance of the internal extraction implementation.

1.4 CONTRIBUTION

In this document we present the configuration and steps to successfully adapt YOLOv3 to extract fingerprint minutiae for automatic fingerprint recognition, the extractor configuration, trained weights and scripts to analyze the performance can be found here `https://github.com/bfs15/fingerYolo`. It is also shown performance tests in the form of precision-recall curves of the trained networks, together with speculations into the possible causes of lack or gains in performance of different configurations.

Although the adapted network did not achieve the high performances of specialized networks, such as MinutiaeNet (Nguyen et al., 2018) that achieved 85.9% precision and 84.8% recall in FVC2004, the adapted YOLOv3 approach here described achieved 86.67% precision and 86.09% recall using a 0.2 threshold. Which is a very good outcome given that YOLO is the fastest available detection convolutional neural network[1].

[1]Our precision does not take angle correctness into account, while MinutiaeNet does, which is why we don't claim to have achieved better performance.

1.5 DOCUMENT LAYOUT

Next chapter, Background, basic information about convolutional neural networks will be explained together with some specific improvements in network design. Following that, on the same chapter, there is a recapitulation of Deep Learning designs for object detection in recent years, in which successful systems will be described.

In Chapter 3, Bibliography Review, existing work on minutiae detection will be discussed, with a focus on Deep Learning approaches. Firstly the availability of databases seen in the existing work will be listed, and their properties described. On the Existing Work section, three recently researched systems for minutiae extraction will have their networks and techniques reviewed together with the performance assessment each research made available.

In Methodology the process of adapting the network and building the extraction system will be described in detail. This includes the network configuration, database information, the background needed to use and evaluate SourceAFIS, and the method for angle extraction.

In Experiments the scenarios for each test will be described. The resulting precision-recall curves, identification accuracies, and FAR FRR curves are displayed there, together with speculations about these results.

2 BACKGROUND

2.1 DEEP LEARNING

Deep learning is a class of machine learning methods that learn data representations and can have either supervised, semi-supervised or unsupervised training. In this study we analyze the use of convolutional neural networks (CNNs) for the purposes of classification, and localization (together called detection).

Supervised training utilizes a database of examples, previously labeled (manually or by a curated automatic approach), each example being defined by the network input and its desired output. For example an image and its object's bounding boxes with the name of the class of the object (the label). A network is trained with such dataset to adjust its parameters guided by an optimization algorithm with the objective of minimizing a defined loss function, for example classification error in a classification network. And for the calculation of the loss function the labels, in other words, the desired output, is used.

While the training stage minimizes the loss function in the dataset, the objective of a classifier is achieving great performance in examples which are not in the training dataset. What is desired is that the network generalizes from the training dataset to classify real world examples well.

Evaluating the error rate of the final model based on the training dataset is unfair due to the fact it does not properly demonstrate the generalization power of the network and its performance on real world examples. With that in mind, the efficacy of CNNs are measured with a test database, a database which was never seen during training, using statistical analysis. The test database can't be used for purposes of hyper-parameter tuning and final model choice, to avoid overfitting the hyper-parameters to the test database and making an unfair performance assessment. For those purposes a set disjoint from the test set, the validation set is used. To make such performance assessments, the validation and test sets need to be labeled as well, vast quantities of labeled data are required for the development of convolutional neural networks.

There are other types of training such as semi-supervised or unsupervised but this study focuses on supervised which is widely used in classification and detection tasks.

Convolutional Neural Networks are multilayered structures, each layer transforming its input and outputting the values to the next layer(s) (which will in turn be their input). Though each layer can have multiple layers as inputs and outputs (Szegedy et al., 2016), there are very successful CNNs in image recognition that are sequential (Simonyan e Zisserman, 2014) (each layer having only one previous and one next layer, pipeline-like).

Many successful image classification networks are assembled as multiple convolutional and max pooling layers that are eventually flattened and input into dense (fully connected) layers (in dense layers each unit/neuron is connected to each neuron in the previous layer) ending in a softmax output (Dense) layer the size of the number of classes in the classification task.

A convolutional layer, basis for image classification networks, applies multiple kernels (filters) to the input tensor (analogous to image processing kernels/masks). A tensor being for example could be an image with shape (lines, columns, channels) or an output from a convolutional layer, called a feature map, which instead of channels is as deep as the number of filters in the layer.

These layers together with max pooling are responsible for learning the representation (Garcia-Gasulla et al., 2017) of the data. After flattening[1], the dense layers then uses the extracted feature maps to classify the data.

This information can be used in transfer learning, in which a already trained network is repurposed for other tasks/classes (Donahue et al., 2014), with this the powerful feature extraction capabilities of big (and expensive to train) networks such as Inception can be reutilized.

Deep learning recently has been consistently given good results thanks to some factors:

- The development of new models and algorithms to train deep neural networks.

  Activation functions such as ReLU that diminish the problem of the vanishing gradient that was pronounced in deep neural networks (Glorot et al., 2011). Batch Normalization. Residual learning. Wider sparsely connected architectures popularized by Inception (Szegedy et al., 2015) and VGGNet (Simonyan e Zisserman, 2014). Optimizers such as Adadelta (Duchi et al., 2011) and later Adam (Kingma e Ba, 2014).

- Vast quantities of labeled data, necessary to train robust DNNs.

  Though a lot of unlabeled data is available on the web, the labeling costs sometimes are an issue in building a sufficiently big database. The biggest contributors are publicly available databases such as ImageNet (Deng et al., 2009).

- Faster and cheaper processing power fit for the purposes of running CNNs have become available through GPUs (Graphics Processing Unit).

- Availability of open source high-level neural networks frameworks that have efficient C++/CUDA backend implementations.

  Tensorflow (Abadi et al., 2016) + Keras (Chollet, 2015); Torch7(Collobert et al., 2016)

2.1.1 Improvements

As mentioned in the previous section, new architectures and and algorithms helped to make training deeper networks more feasible.

2.1.1.1 Batch Normalization

Batch Normalization, was tested in Inception in a study by Sergey Ioffe and Christian Szegedy (Ioffe e Szegedy, 2015). By introducing Batch Normalization in the Inception network they were able to match the accuracy of Inception in less than half the number of training steps, but by taking advantage of the technique and increasing the learning rate by 5 times they were able to achieve the same 14 times fewer steps than Inception.

Batch Normalization is simply a standardization of the batch; to do it the mean (μ) and variance (σ^2) of the batch are calculated, with x_i being all the N input features of the batch i=0..N-1:

[1] converting the feature map into a single column (a 1D vector).

$$\mu = \frac{1}{N}\sum_{i=0}^{N-1} x_i$$

$$\sigma^2 = \frac{1}{N}\sum_{i=0}^{N-1} (x_i - \mu)^2$$

With these variables you can now calculate the normalized features x_i:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

The new normalized features have zero mean and unit variance. And the result is that Batch Normalization reduces the covariate shift introduced along the network by the transformations performed in the layers, by normalizing the output of a layer before the activation. Covariate shift meaning when the input distribution to a learning system changes (specifically the mean and variance), and this can happen by the virtue of the network kernel and activation function configurations.

After the normalization a linear transformation is applied in each feature:

$$l_i = W\hat{x}_i + b$$

Where W and b are learnable parameters, possible since batch normalization is differentiable (the basis of backpropagation). l_i is then input into the activation function and that is the layer's output. Batch Normalization yields a lot of improvement in the network's training speed and some in the accuracy, especially when paired with ReLU activation functions (Ioffe e Szegedy, 2015).

2.1.1.2 ReLU

Comparing ReLU $max(0, Wx+b)$ with the formerly widely used Sigmoid activation function $1/(1+e^{-x})$, it brings some solutions to the problems that Deep Learning introduces when stacking a bigger quantity of layers (Krizhevsky et al., 2012). First, the function is computationally inexpensive, a single fused multiply add and max function as opposed to a division and exp function.

Secondly it handles the vanishing gradient problem that is emphasized in Deep Learning as the gradient from, for example, a Sigmoid is reduced each layer. Since the derivative of a Sigmoid is at most 0.25 after multiplying it for the however many layers it quickly converges to zero (vanishes). This in contrast to ReLU which solves this because it has a constant gradient when $x > 0$.

Additionally ReLU enforces sparse representations, since the function returns real zeroes when $x < 0$. Sparse representations are more likely to be linearly separable, which is desirable of feature extraction. In practice ReLU shows a faster convergence of deep models, in particular, ImageNet achieved six times faster training time than the equivalent network with tanh neurons (Krizhevsky et al., 2012).

2.1.1.3 Data augmentation

Vasts amounts of data are required for training a deep network, but there's not always much available and it may be costly to gather and label more. Data augmentation is a technique to artificially increase the quantity of data the network receives as input

during training. Simple ways of achieving this are to crop, rotate, saturate, flip, color jitter, the input training images randomly, thereby diversifying the input.

This technique reduces the chances of the network overfitting, by it's random nature and artificial increase in size making it harder for the network to perform well on the training data without more generalization than the non-augmented counterpart. Data augmentation also improves the performance of networks as reported in (Taylor e Nitschke, 2017), as increases in the training set generally expand the networks potential. Some other more advanced techniques such as using other Neural Networks (e.g. GANs) to alter the input data have gathered some good results (Perez e Wang, 2017).

As for object detection tasks, the network designs and their improvement, adaptations, and shortcomings are described in the following section.

2.1.2 Convolutional Neural Networks for Object Detection

A CNN classifiers, such as Inception, output categorical data in a one-hot encoded vector. You can interpret the output vector as the probability of the input being in each class represented in the vector. This is possible due to having a softmax layer as the output and using cross-entropy as the loss function.

But for purposes of location that is not enough, what we need is either a single point coordinate or a bounding box (which can be described with 2 coordinates).

2.1.2.1 Sliding Window

An expensive way of adapting a CNN classifier into a location network is using a sliding window over the desired image. Taking windows of fixed size and stride from image and using them as input to the network we can then use the output probabilities to discern in which area the object is.

Aside from having a large amount of windows for the network to process, turning the computation expensive, there is also a problem of the scale of the object in the image. Depending on the kind of images being analyzed an object could occupy a wide range of the image's area, e.g. from 20% of the image to 75%.

A solution would be to resize the input image to multiple scales and choose a window size such that it will completely contain the object in one of the resized images, independent from the original image's scale.

2.1.2.2 Region-based Convolutional Neural Networks (R-CNN)

The sliding window approach being computationally very expensive, R-CNNs (Girshick et al., 2014) try to lessen the cost of using a CNN for object detection by using a region proposal algorithm. The one used in the original paper by Ross Girshick, et al. (Girshick et al., 2014) uses Selective Search (Uijlings et al., 2013) but is agnostic to the method used. The idea is that the region proposal algorithm reduces the amount of windows that will be processed by the CNN by finding Regions of Interest (ROI).

In the original paper the proposed regions are fed into a CNN and a feature vector is extracted, which is then classified with linear SVMs (Girshick et al., 2014).

2.1.2.3 Spatial Pyramid Pooling (SPP-net)

While R-CNNs reduced the amount of windows evaluated by the CNN, its was still way more computationally expensive than HOG based classifiers, which spurred the creation of SPP-net (He et al., 2014).

SPP-net calculates the feature maps from the for whole image only once and use that to calculate window-wise features for each patch generated by Selective Search. In other words SPP-net extracts window-wise features from regions of the feature maps, while R-CNN extracts from image regions.

To achieve that Spatial pyramid pooling (SPP) is used, which generates fixed-length representations from pooling features in arbitrary regions (sub-images). SPP layer divides a region of any arbitrary size into a fixed number of bins that are then max pooled on each of the bins. By consequence SPP-net allows for a variable input image size, since fixed-length representation are generated regardless of image size. The SPP layer was inspired from spatial pyramid matching (SPM) (Lazebnik et al., 2006).

One disadvantage SPP-net has is that due to the SPP layer it cannot update the convolutional layers that precede it, which limits the accuracy.

With SPP-net, Kaiming He et al. achieved rank 2 in ILSVRC 2014 object detection and a speedup of more than one hundred times over R-CNN.

2.1.2.4 Fast R-CNN

Fast R-CNN (Girshick, 2015) uses ideas from both R-CNN and SPP-net while rectifying their drawbacks. As opposed to SPP-net, Fast R-CNN's training can update all network layers (end-to-end learning).

Fast R-CNN takes as input an entire image and a set of object proposals; the network generates a feature map from the entire image and then each object proposal goes through a region of interest (RoI) pooling layer.

RoI pooling layer converts the features inside valid regions of interest into a small fixed spatial extent (H x W) feature map. The RoI layer is a simple spatial pyramid pooling layer used in SPPnets (He et al., 2014), with only one level.

The feature vectors from the RoI pooling layer are then fed into dense layers that branch into two output layers. One layer classifies and the other outputs four real numbers for each of the classes, which encode bounding boxes. Having 2 output layers, Fast R-CNN no longer needs training independently for classification and localization.

The part that generates a single feature map from the entire image before the RoI pooling layer is called the shared convolutional subnetwork, while the part that receives separate object proposals and get computed individually from the shared output is called the RoI-wise subnetwork.

To find object proposals it still uses Selective Search.

2.1.2.5 Faster R-CNN

In the interest of getting even faster results, Faster R-CNN (Ren et al., 2015) introduces Region Proposal Networks (RPN). Since Selective Search was the speed bottleneck of Fast R-CNN, RPNs were designed to substitute it as region proposal method.

The input of RPN is an image of variable size (like previous methods) and outputs rectangular object proposals, each with a measure of membership to classes vs. background.

This is done by using a small fully-convolutional network over the feature map output by the last shared convolutional layer. Then the output from RPN goes into two sibling fully-connected layers: a box-regression layer, and a classification layer.

2.1.2.6 Region-based Fully Convolutional Networks (R-FCN)

R-FCN (Dai et al., 2016) takes inspiration from Residual Nets (ResNets) (He et al., 2016) and GoogLeNet (Szegedy et al., 2015) of being designed as fully convolutional, RPNs from Faster R-CNN (Ren et al., 2015), and capitalizes on the idea of shared feature maps from SPP-net (He et al., 2014) by getting rid of RoI-wise subnetworks.

Jifeng Dai et al. (Dai et al., 2016) brings up the issue that while image-level classification perform better with translation invariance, object detection actually needs representations that are translation-variant so that it has information on how well the bounding-box overlaps with the object. Their proposed theory is that deeper networks provide this invariance, and that using RoI pooling layer in the middle of the network breaks this, but as a consequence part of the network becomes independent (the RoI-wise subnetwork) and this compromised speed.

So they propose a network such that there are no RoI-wise subnetwork. To accomplish that they generate a set of position-sensitive score maps by using specialized convolutional layers in the FCN. The score maps are position-sensitive because each has position information as a relative spatial position (e.g., "to the left of an object") encoded within.

The score maps from the FCN are then used in a RPN to extract region proposals (RoIs). Then the same score maps, filtered by the RPN, go through a position-sensitive RoI pooling layer, with no RoI-wise subnetwork afterwards.

With these changes they achieved speeds 2.5x to 20x faster than Faster R-CNN.

2.1.3 CNN regressors

The methods above tackle the detection problem as a classification problem with a pipeline: Object proposals are generated and then these proposals are used in the input to the CNN which will classify (or in Fast/er R-CNN it will also perform bounding-box regression but individually so).

The following methods tackle detection as a regression problem, they discard the idea of region proposals and through a unified network simultaneously predict multiple bounding boxes and class probabilities for each.

RetinaNet-101 (Lin et al., 2017b) won't be detailed here, but deserves mentioning. It has the best performance on the COCO dataset, a bit above YOLOv3, while taking 3.8 times longer than YOLOv3-608. Note worthy is the use of the same technique used in YOLOv3, multi-scale predictions, explained in the following section.

2.1.3.1 YOLO (You only Look Once)

YOLO's (Redmon et al., 2016) output is divided in a grid that represent regions of the input image. Each cell of the output grid is responsible for predicting B bounding-boxes and their confidence as well as the C class probabilities (a single set of probabilities for all bounding-boxes). YOLO's architecture is inspired by the GoogLeNet model, it uses smaller modules in total having 24 convolutional layers followed by 2 dense layers the last of which generates the output from the convolutional layer's feature maps.

In contrast to region proposal-based methods that use a subset of the whole image's feature maps to classify, each cell from YOLO's output has information from the entire image, as such they also encode contextual information about classes. As evidence of this, compared to Fast R-CNN YOLO makes less than half the number of background errors.

One shortcoming YOLO has is that the size if the output grid heavily limits the network's ability to detect small objects cluttered together, such as birds in a flock.

While the original YOLO's (Redmon et al., 2016) architecture failed to achieve state-of-the-art detection accuracy a subsequent improved version, YOLO9000, was able to perform state-of-the-art, real-time object detection (Redmon e Farhadi, 2017), and has since improved the design further with YOLOv3 .

A change introduced in YOLOv3 (Redmon e Farhadi, 2018) improves detection for small objects, called multi-scale predictions. This design change was inspired by Feature Pyramid Network (FPN) (Lin et al., 2017a). It is a method of feature extraction in multiple levels of the network. Similar to Scale-space Image Pyramids in image processing (Adelson et al., 1984). Except it doesn't scale the image in multiple pyramid levels, and processes each of them on a network. Instead this design pattern uses the inherent architecture of CNNs, which are arranged in multiple layers themselves. The convolutional layers get smaller in spatial size (resolution) as they go deeper in the network. Therefore, it uses features of multiple levels, instead of just the last one, to do multiple predictions. However, an important modification to that idea is the reuse of higher-resolution maps of the feature hierarchy, especially for smaller objects.

The way YOLO implements this idea is to upsample the current layer feature maps and concatenates them with previous higher-resolution feature maps. Afterwards more convolutional layers are added to refine this merged output. This makes both semantic information from the current (upsampled) features and finer-grained information available to prediction layer. As a result, this improves small object detection by a large margin. Tests in COCO resulted in going from a APs of 5.0 in YOLOv2 to 18.3 in YOLOv3 (Average Precision on small scale objects).

In addition to that there is YOLOv3 SPP which adds a set of layers just before the three prediction layers of YOLO v3 to perform Spatial Pyramid Pooling. In the implementation there are three different maxpooling layers with different sized filters parallel to each other. The output of these layers are then concatenated and continue down to the prediction layers. These changes are discussed in a 2019 paper (Huang e Wang, 2019).

2.1.3.2 Single Shot Detector (SSD)

Developed in the same year as YOLO (Redmon et al., 2016), SSD (Liu et al., 2016) surpassing the accuracy from Faster R-CNN while also performing in real-time (59 FPS).

To allow detection at multiple scales, progressively smaller convolutional feature layers are added following the base convolutional network. From each of these feature maps 3×3 kernels produce bounding-boxes (encoded as 4 offsets) and C class probabilities. Therefore, to predict B bounding boxes a total of $(C + 4)B$ kernels are needed. Applying such filters in a $m \times n$ feature map yields $(C + 4)Bmn$ values, Bmn bounding boxes.

Since there are multiple feature maps at different scales and each pixel of them gets B bounding-boxes predicted, the size of the output is quite larger than YOLO's. In the example of the original SSD paper (Liu et al., 2016) the feature maps used to extract

detection boxes are of size 38, 19, 10, 5, 3, 1 and with 4, 6, 6, 6, 4, 4 bounding boxes for each feature map respectively, this yields $38^2 \times 4 + 19^2 \times 6 + 10^2 \times 6 + 5^2 \times 6 + 3^2 \times 4 + 1^2 \times 4 = 8732$ detection boxes per class (in contrast to YOLO's 98). This higher granularity results in a greater ability to detect smaller objects closer together when compared to YOLOv2. However, they don't implement the modification FPN endorses, the reuse of higher-resolution maps of the feature hierarchy. Consequently, it underperforms on small objects when compared to YOLOv3; On the COCO dataset it has a APs of 10.2, compared to YOLOv2 APs of 5.0, and YOLOv3 APs of 18.3.

As a consequence of these adaptations they achieve high-accuracy using relatively low resolution inputs.

2.2 MINUTIAE

Biometric authentication has an advantage over traditional authentication in that the characteristics are not usually easily stolen or reproduced. Additionally, fingerprint recognition has been extensively used by forensic experts in criminal investigations (FBI, 1984). Minutiae are believed to be the most discriminating and reliable features (FBI, 1984), and a good accuracy in minutiae extraction of latent fingerprints is desirable in scenarios such as criminal investigations.

Minutiae are major features of a fingerprint, also called level 2 features, they are formed by the ridges of the print and are classified as shown in Fig. 2.1.

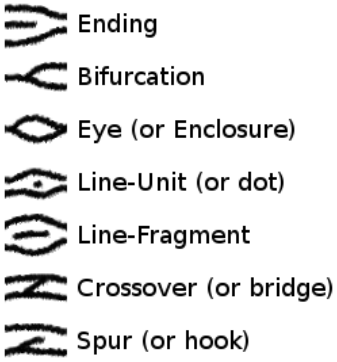


Figura 2.1: Minutiae examples

In the literature the most used ones are ridge ending and ridge bifurcation, which are the most common minutiae (Maltoni et al., 2009). Other minutiae types are rare enough that most minutiae matchers (such as the one in SourceAFIS) only support two distinct types, endings and bifurcations. Often none of the other types appear in any given fingerprint.

The extraction of minutiae, for use in set pairing algorithms (for fingerprint recognition), means finding the following attributes of each minutia: position, orientation angle, minutiae type. Some fingerprint recognition algorithms found in the literature ignore the type of minutiae because of the lack of robust minutiae type classifiers (Prabhakar et al., 2003). As an example, Minutiae Cylinder Code (Cappelli et al., 2010) does not use type information.

Having the location of each minutiae, the orientation angles can be estimated through the orientation field obtained by a dictionary based method (Yang et al., 2014). Therefore the focus will be on the detection (position extraction) of the minutiae in a given fingerprint.

2.3 CONCLUSION

There were quite a lot of techniques in Deep Learning researched over the years. Some like data augmentation, batch normalization and ReLU are extensively used in recent publications. These consistently increase performance and are quite easily integrated into initial network designs, as seen in the evolution of YOLO and Inception versions. Especially ReLU-like activation functions, since sigmoid activations make deep models restrictive because of vanishing gradients. Specifically for detection, a lot of research effort has gone into making faster networks, since the naive approaches are very costly (e.g. sliding windows). While other design patterns like region proposals, be it by using Selective Search or a Region Proposal Network, aren't used in some detection networks state of the art designs like YOLO and RetinaNet. A new promising design pattern is Feature Pyramid Network-like extraction of features, adopted by YOLO and RetinaNet, while being used in a primitive form in SSD. It improves small object detection greatly, as evidenced by a comparison made to baseline networks in the source paper (Lin et al., 2017a), and in the integration of the pattern in YOLOv3 (Redmon e Farhadi, 2018). This is especially relevant for detecting objects small as minutia in fingerprints.

3 BIBLIOGRAPHY REVIEW

In this chapter we review the available databases and their layout as well as some existing work on minutiae detection using Deep Learning. The databases seen in the referenced work will be described together with the existing corresponding minutiae labels. Three existing works in minutiae extraction using CNNs will be linked with discussed detector designs. Additionally, the performance assessment each research made available will be summarized.

3.1 DATABASE AVAILABILITY

As discussed, a high number of labeled data is desirable in the development of a neural networks, especially deep neutral network which have many weighs to be trained. The desired would be a publicly available database with fingerprint images and all their corresponding minutia, of varying people and possibly multiple capture devices for robustness.

Since data collection in biometric systems are time-consuming, there are few publicly available fingerprint image databases, even fewer with minutia information extracted since that has to be done by specialists.

One of the most widely used databases in the literature are from the Fingerprint Verification Competition (FVC) (Maio et al., 2002a) (Maio et al., 2002b) (Maio et al., 2004) (Cappelli et al., 2007), each of which is a multi-database, containing four disjoint fingerprint databases, each collected with different sensor technology. FVC databases were used in the studies mentioned in the following section.

FVC is comprised of four distinct databases: DB1, DB2, DB3 and DB4; each database has 110 different fingers and 8 samples per finger (110x8=880 fingerprint images in total). The databases from FVC 2006 increased in size to 150 fingers and 12 impressions each, having in total 150x12=1800 fingerprint images. Each database is partitioned in two disjoint subsets A and B:

- Subsets DB1-A, DB2-A, DB3-A, and DB4-A contain 100 fingers, 8 impressions of the same finger for a total of 800 images. FVC2006 has 140 fingers in A subsets, 1680 images. These subsets are used for the algorithm performance evaluation in the competition.

- Subsets DB1-B, DB2-B, DB3-B, and DB4-B contain the last 10 fingers, 8 impressions of each totalling 80 images. These are made available to the participants of the competition as a development set to allow parameter tuning before the submission.

The image format is BMP, 256 gray-levels, uncompressed, size and resolution vary depending on the database.

In FVC of years 2004 and before, difficulties in the fingerprints were purposely asked of the volunteers. The impression gathering was made in three sessions for each volunteer. In the second session, it was requested to exaggerate skin distortion in two impressions and rotation in another two of the four impressions of the finger; during the third session, fingers were dried in half and moistened in the other half of the impressions.

FVC2006 (Cappelli et al., 2007) was collected without introducing difficulties, such as distortion, rotation and displacement, and wet/dry impressions; however, the population is more heterogeneous, this time also including manual workers and elderly. There was also no guarantee of a minimum quality in the acquired images and the final datasets are subset from a larger database by choosing the most difficult fingers (according to a quality index).

Although the FVC databases don't have minutiae information inherently, the study "Standard Fingerprint Databases"(Kayaoglu et al., 2013) manually extracted minutiae information from FVC2002 and FVC2004 and made it available at `ekds.gov.tr/bio/databases.html`, in this document referred to as the FM3 labels. They developed an application called Fingerprint Manual Minutia Marker (FM3), which was then used to mark minutiae. The minutiae they extracted are for the subsets DB1A and DB3A of two FVC competitions (a total of 4 subsets).

To analyze the results of the manual extraction, they used two anonymous commercial fingerprint matchers that were in the top 20% in the FVC competitions of 2002 and 2004. They compared the authentication performance of the manually extracted minutiae information with automatically extracted minutiae. Using matcher 2, in FVC 2004 DB1A they achieved Genuine Acceptance Rates (GAR) of 92.5%, 94.0%, 95.9% with 0.001%, 0.01%, and 0.1% of respective False Acceptance Rates. While the best results of automatic extractors achieved GAR of 86.7%, 89.5%, and 92.3%. In the same way but on FVC2004 DB3A, FM3 achieved GAR of 96.7%, 98.1%, and 98.8%. The first automatic extractor had GARs of to 88.5% 90.3% 92.9%, results strictly better than extractor 2, and noticeably worse than the manually extracted.

The files follow the international standard ISO/IEC 19794-2:2005 Information technology - Biometric data interchange formats - Part 2: Finger minutiae data.

This extracted information has

- fingerprint image quality (poor/fair/good, as perceived).
- singular point (core and delta) locations.
- for every minutia:
  - its type (ending/bifurcation).
  - location in image coordinates.
  - angle.
  - quality (poor/fair/good).

Figure 3.1 exemplifies the difference between the labels of (Kayaoglu et al., 2013) and the ones used in FingerNet (Tang et al., 2017a). The labels used by FingerNet don't have any type information.

Other widely known NIST Special Databases (SD) have been discontinued and are now unavailable to the public. The database NIST SD27 was used to test FingerNet and MinutiaeNet mentioned in the following section.

One technique that can be employed to help on cases where there is a limited amount of labeled data is data augmentation witch helps in generalizing the classifier and achieving a better accuracy (Taylor e Nitschke, 2017). By modifying input images such as changing brightness, cropping, rotating, adding noise, or such other modifications, the database can be artificially increased in size.

Figura 3.1: Different labels of FM3 (Kayaoglu et al., 2013) **X** and FingerNet (Tang et al., 2017a) + in the image 1_1 of FVC 2004 DB1A

More relevantly, in the SSD paper (Liu et al., 2016) they found data augmentation to be crucial to improve performance even when training with big databases such as Pascal VOC2007 (Everingham et al., 2007) in which there are 9,963 images (containing 24,640 annotated objects).

A future idea would be utilizing specialized denoising (Xie et al., 2012) and super-resolution (Wang et al., 2018) neural networks as a preprocessing step to see if an improvement in accuracy performance can be gained in tradeoff with speed performance.

3.2 EXISTING WORK

While in the networks discussed in Section 2.1.2 detection tasks needed to generate bounding boxes for objects, in minutiae extraction a single point is enough to describe is location. The number of minutiae classes is also much lower when compared to 80 object categories from COCO and 20 classes from Pascal VOC. As mentioned in Section 3.1 the labeled database by M. Kayaoglu et al. (Kayaoglu et al., 2013) has minutiae types simplified into only endings and bifurcations with other types, such as bridges, being considered composites of these two types. Even then, the types of minutiae are ignored in the studies examined in sequence, focusing only on localization with two classes, the presence of minutiae or not.

With the information discussed about detection networks in Section 2.1.2 we expect earlier versions of YOLO (lower than v3) to perform badly; since they have recorded bad performances when detecting small objects. This is due to YOLO's output grid limiting the detection of small objects cluttered together, which in fingerprint minutiae occurs frequently. YOLOv3 hopefully circumvents such a shortcoming with feature pyramids.

The scale invariance problem rectified in SSD by using multiple feature map resolutions isn't such a problem in this context, when compared to detection datasets such as COCO (Lin et al., 2014) and Pascal VOC (Everingham et al., 2010) minutiae are present in similar scales. So the progressively smaller feature maps could be cut-off earlier since semantic and global information are not needed, instead local and higher resolution information is desired to detect minutiae. In contrast to versions of YOLO

before v3, the possible precision in detection of small objects should be a benefit of this type of architecture. The same can be said of YOLOv3's FPN-like design pattern.

There exists some literature about using deep learning for purposes of minutiae extraction. We will focus on these three most recent open access works:

- Fingerprint minutiae extraction using deep learning (Darlow e Rosman, 2017) (MENet);
- FingerNet: An Unified Deep Network for Fingerprint Minutiae Extraction (Tang et al., 2017b);
- Robust Minutiae Extractor: Integrating Deep Networks and Fingerprint Domain Knowledge (Nguyen et al., 2018) (MinutiaeNet).

3.2.1 MENet

MENet (Darlow e Rosman, 2017) has 5 convolutional layers, 2 fully connected layers of 1024 nodes, and a softmax output layer. The convolutional layers have 32 5×5 filters, and the first two of these layers use pooling. The ReLU activation function was used. Data augmentation was employed with grayscale colour inversion, contrast degradation, contrast improvement, and noise degradation.

Their network doesn't classify the type of minutiae, for example as ending or bifurcation, only detects the location of each one.

MENet used FVC 2002 and 2004 for training and testing data, they split them both 80% for training and 20% for testing. A novel automated method of gathering labeled data of fingerprint minutiae was proposed. The method uses a fusion of multiple commercially available minutiae extractors with a voting threshold; only points with more than 4 out of 5 positives for minutiae were considered a overall positive.

While the network is small, its downside is that it uses a 30×30 sliding window over the input fingerprint. Which is known for being very slow in performance when compared to networks with RoI extraction and regression networks.

3.2.2 FingerNet

FingerNet (Tang et al., 2017a) combines domain knowledge and the representation ability of deep learning. They translated domain methods into convolutional kernels and integrated them as a network. This network is shallow and has fixed weights and unifies normalization, orientation estimation, segmentation, and Gabor enhancement.

FingerNet uses the enhancement together with segmentation score map as input to a 3 layered conv-pooling network to perform feature extraction (analogous the shared convolutional subnetwork described in Fast R-CNN). The resulting feature map is then input into four different shallow subnetworks with sigmoid activation functions that generate the following output maps.

- The first is a minutiae score map, in encodes the probability of each position (x/8, y/8) to have a minutiae.
- The second and third are X and Y probability map, which enables precise minutiae extraction.
- The fourth is the angle distribution map.

These subnetworks work as regressors, more similar to YOLO than SSD. It circumvents the low resolution of YOLO's output grid by using the second and third maps, which give additional information about precise location of the minutiae by providing offsets.

The minutiae score map does not distinguish between minutiae types. Therefore this network does not attempt type matching. Reflecting that, their database doesn't have any type information, only coordinates and orientation.

FingerNet training data is private and was collected from crime scenes, including about 8000 pairs of matched rolled fingerprints and latent fingerprints. Each latent fingerprint is 512 x 512 pixels in size and 500 dpi with minutiae labeled by their experts.

3.2.3 MinutiaeNet

MinutiaeNet (Nguyen et al., 2018) has 2 separate modules, CoarseNet and FineNet. CoarseNet generates a minutiae score map which is then refined by FineNet by analyzing the candidate patches (centered by candidate minutia points of the map).

FineNet is comparable to MENet but with CoarseNet as a filter before it, so that moving a sliding window isn't necessary, instead it refines the scores of the candidates output by CoarseNet. Similar to region proposal-based methods discussed in Section 2.1.2, CoarseNet acts as a RoI extractor for a RoI-wise subnetwork, FineNet.

CoarseNet uses Tang et al. (Tang et al., 2017a) domain knowledge unified network described in FingerNet as a baseline.

It is notable the sheer difference in depth FingerNet and MinutiaeNet have; only in CoarseNet, MinutiaeNet has 19 convolutional layers. MinutiaeNet's approach is way deeper, especially FineNet, and to avoid the exploding or vanishing gradients problem CoarseNet uses residual learning, the same goes with FineNet that is inception-resnet based. The run time per image is around 1.2 seconds for FVC 2004 images on a Nvidia GTX GeForce, which is 0.833 FPS.

3.3 PERFORMANCE ASSESSMENT

To compare performance between different automatic extractors there are two main approaches. Firstly a more direct comparison of extracted minutiae and ground truth minutiae extracted manually by experts. Secondly a comparison between the performance of the extracted minutiae in matching algorithms, while indirect, it gives evidence of the performance in real world applications. As mentioned in Section 3.1, labeled data is expensive which is why comparing to ground truth is not usually an extensive test. Measuring matching performance, then, is a welcome addition to performance assessment.

On the MENet study, to evaluate performance, minutiae were manually extracted from 100 fingerprints from FVC datasets, totalling 4730 minutiae points. The manually minutia points extracted were then visually compared to automatically extracted minutiae from MENet and other extractors. They reported a Equal Error Rate of 5.450% on FVC2004 of the minutiae extracted by MENet.

FingerNet and MinutiaeNet compared extracted minutiae to ground truth labels automatically, if there is a point within D pixels of difference in location and O degrees of difference in orientation of a ground truth point it is considered a match.

FingerNet used $D = 15$ and $O = 30$ in manually extracted minutiae from the NIST SD27, and FVC 2004 databases. MinutiaeNet used D equal to 8 and 16 in combination of

O equal to 10 and 30, totalling 4 different settings, also tested on the same databases as FingerNet.

For testing FingerNet, the images used were from NIST SD27, and FVC 2004 set A, the labels were all made by their specialists, the difference between them is exemplified in Figure 3.1. They have precision-recall curves on their paper, for NIST SD27 and FVC2004. No value was highlighted from the curve, but we extracted from the image that on FVC 2004 they achieved a precision of 76% approximately with 80% recall.

MinutiaeNet achieved in FVC 2004 a Precision of 85.9% and 84.8% Recall, with F1 Score 0.853, with settings $D = 16$ and $O = 30$; the paper also has precision-recall curve.

3.3.1 Matching Performance

To test matching performance in the MENet they used the performance evaluation protocol from FVC 2004 (http://bias.csr.unibo.it/fvc2004/perfeval.asp) calculating the True Positive and False Positive Rates with 2,800 genuine tests and 4,950 false acceptance tests. The used matching algorithm was Minutiae Cylinder Code (Cappelli et al., 2010) (MCC). The test was performed on FVC 2002 DB1A and FVC 2004. Every fingerprint was compared against all impressions of the same finger for 2800 genuine comparisons and the first fingerprint of each fingerprint was compared against all other first impressions for 4950 impostor comparisons. They displayed a ROC curve that can be seen on their paper, for FVC 2002 and 2004. To illustrate, in FVC 2004 they achieved 90% True Positive Rate with a 1% False Positive Rate, and approximately 84% TPR with a 0.1% FPR. However it must be noted that both training and testing data were used in the matching performance evaluation. Since MENet was trained using data augmentation, and both datasets, they argue that a maximum of 80% of the data was seen while augmented; and also that the automatically generated labels for training wouldn't have registered many of the challenging minutiae, and therefore they would have never been seen by the network.

To test matching performance in the FingerNet study, they generated a Cumulative Match Characteristic curve on NIST SD27. The used matching algorithm was based on extended clique models (Fu et al., 2013), using only minutiae. The test was performed on 40 thousand fingerprints including NIST SD27, NIST SD4 and their private 8000 pair of fingers database.

The MinutiaeNet study did not perform any matching performance comparison, but did show precision-recall curves against ground truth minutiae.

3.4 CONCLUSION

There are a low amount of fingerprint databases available, even less with minutiae data. Furthermore, a prolific dataset NIST SD27 was made unavailable. Even so, FVC datasets have proven to be enough to train deep networks using proper data augmentation. The available labels to us are from Fingerprint Manual Minutia Marker (FM3) (Kayaoglu et al., 2013) and from FingerNet (Tang et al., 2017b), although FingerNet labels don't have minutiae type information.

Some recent works in fingerprint detection using Deep Learing have been researched recently. MENet uses a very slow but accurate approach, sliding window. From the same year, FingerNet performs bounding-box regression. FingerNet also integrates some domain knowledge in the network, then also used in MinutiaeNet. More recently, MinutiaeNet

implemented a deeper design that employs an pattern similar to Region Proposal Networks in the form of CoarseNet. Yet, none of them classify the type of minutiae. These design patterns were detailed in Section 2.

4 METHODOLOGY

On this chapter it is shown the adjustments made to the base configuration of a YOLO implementation. Additionally a technique for approximating angle information is detailed. The available databases information, format and the preparation of the datasets for training the network are included.

4.1 YOLO CONFIGURATION

The YOLO implementation here adapted is from AlexeyAB and the repository is available publicly here `github.com/AlexeyAB/darknet`. It is multiplataform implementation made in C/C++ making use of CUDA. There are multiple configurations available, from YOLOv1 to YOLOv3, the adaptations for faster, smaller sized, networks called the tiny versions and the configurations for specific datasets VOC and COCO.

The adaptations need to take into consideration the type of objects and context, in this case minutiae in fingerprints. Since the pictures are greyscale it is not needed to use the usual three channels for image detection. Data augmentation for the same reason doesn't need to alter hue and saturation values, leaving only rotation and exposure changes. Random rotations of 30 degrees and changes of up to 30% in exposure, changing the instensity of the pixels. These augmentations can go either way, for example, -30 degrees to +30 degrees rotation are equally likely. Minutiae are described as points on the image as opposed to having bounding boxes, artificial bounding boxes will have to be constructed for the training stage and reverted in the detection stage.

YOLO, since version 2, uses anchor boxes to make its predictions. Anchor boxes are predefined boxes used as a baseline for the network's predictions. Instead of predicting bounding boxes for the objects directly it predicts the shifts in the anchor boxes. By default YOLOv2 uses five anchor boxes with different aspect ratios and sizes.

For a specialized detector its possible to configure these anchor boxes to match the target objects dimensions. In this case, our object boxes are artificial squares around the minutiae. The implementation by AlexeyAB allows anchor boxes to be automatically calculated given the objects and the width and height of the network input. It does this by clustering the objects bounding boxes and choosing an anchor box for each cluster. The single shot detectors used here resize input images into a fixed sized square; Since the images from the database aren't squares[1] the dimensions for the anchor boxes will be rectangular, even though the objects are squares[2]. The single chosen anchor was simply 18×25 (the anchor is absolute in pixel in v3 configurations and relative in v2). Future efforts to avoid this distortion, such as padding the images, could be made since the impact is unmeasured in this context.

The full configurations and scripts can be found at `github.com/bfs15/fingerYolo`.

[1] The network's input is square, the image is resized to fit.

[2] The image could have been padded into a square, this approach was not tested.

4.2 DATABASE

The dataset we have available to us with labels is FVC2004, with two different sets of labels, the ones used by FingerNet and the FM3 labels. Their difference is exemplified in Figure 3.1. The FVC2004 database has images in the BMP file format with 500dpi and 640 x 480 resolution. Each image has a identification number for the finger and the number of the different impressions.

The FingerNet labels of each image are in .mnt files, they have the absolute coordinates in pixels and angles of each minutiae and don't contain the type (ending or bifurcation).

absoluteX absoluteY orientation

The FM3 labels are in the international standard ISO/IEC 19794-2:2005 Finger minutiae data. They contain orientation and type additionally to the location. A parser of these .iso-fmr binary files is included in the project, it converts (copying) the files inside a given directory to modified .mnt files that also have type information.

The YOLO label format is in the format:

objectClass centerRelativeX centerRelativeY widthRelative heightRelative

The position and size values are relative to the image shape (width, height), instead of pixel absolute values, and each pair of values denotes the center position and size of the box respectively. The labels in the COCO dataset have the same format except for them not being relative.

The coordinates of each minutiae need to be translated into this format, and for that an artificial bounding box needs to be constructed. Since the coordinates only determine the center of the box, to do this translation a parameter that determines the box size in pixels needs to be arbitrarily set, it is called *box* in the test configuration names. The chosen default of this parameter was 30 pixels, this is roughly double the usual pixel distance threshold for a minutiae match. This threshold is used when testing if the predicted minutiae location is considered to match a ground truth annotation, since it's expected that the prediction won't be pixel perfect but will be close with some variance. This parameter was also tested with values 16, 30, 40 and 60 pixels [3] in the YOLO spp configurations [4] to display if and how it affects the final network performance meaningfully.

4.3 SOURCEAFIS

SourceAFIS is an open source software that compares two fingerprints to create a matching score (Vazan, 2009). It does this in two steps, minutiae extraction to create a template and template scoring (through minutiae matching). [5] This matching score is then used to accept fingerprints as a match or reject them as being different depending on a threshold value for the score. For example if the score between two fingerprints are above

[3] 16 because it's the value of our precision evaluation threshold, 30 is roughly double that, 40 is a 10 pixel step from 30 (steps of 15 would have been more consistent in hindsight), and 60 to see the effects of a more extreme value (training is expensive so 50 was skipped).

[4] Chosen because it was the most powerful one available, most probable to give satisfactory performance

[5] Angles and types are extracted along with localization, since the minutiae matching step uses the information.

40 you consider the fingerprints to be from the same finger. By varying this threshold score you can plot Receiver operating characteristic curves (ROC curves). In ROC curves True Positive Rate (TPR) and False Positive Rate (FPR) are plotted against each other. False Positive Rate being the proportion of how many times an impostor is considered a genuine match by the algorithm; True Positive Rate being the proportion of how many times a genuine fingerprint is accepted. So this curve is used to display how changes in the threshold affect the matcher error rates. The ROC curve helps in the choice of threshold for the deployment of a system in a real world application, which may require harsher or softer matching algorithms; if the acceptance of impostors are more undesirable, the threshold can be raised for a lower False Positive Rate in exchange of a lower True Positive Rate to the TPR vs FPR increasing curve.

SourceAFIS uses a template to match fingerprints, a list of minutiae, each with their position in pixels, angle and type (ending or bifurcation). The position can be easily translated from YOLO's resulting predictions from the center of the bounding boxes relative values. Angle extraction is explained in the next section. FingerNet annotations don't have any information about the type of minutiae, the resulting predictions will have no type information. So in this case, the translation to the template minutiae type will be set to ending. This will hurt the potential matching performance. Configurations trained on FM3 labels on the other hand will classify types, it's expected that these gain an edge performance. The code for the scoring algorithm doesn't use the width and height of the template and so the templates have these values set to zero. The documentation of the template SourceAFIS uses can be found at `https://sourceafis.machinezoo.com/template`

4.4 ANGLE EXTRACTION

Since YOLO doesn't have any capabilities to extract angle information a separate approach will be used. After inputting the fingerprint image through YOLO the resulting detection will have a list of coordinates. For each minutiae coordinate an angle needs to be calculated. For that purpose an Orientation Map will be calculated from the fingerprint, this approach is reported here (Huang et al., 2007).

From the greyscale fingerprint image a Sobel operator is applied. This is a convolution with the filters:

$$F_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \quad \text{and} \quad F_y = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

The convolutions result in two matrices D_x and D_y, you can then extract the magnitude ρ and orientation θ. Each of the elements of these matrices represent $\rho cos(\theta)$ and $\rho sin(\theta)$ respectively.

$$\rho = \sqrt{D_x^2 + D_y^2} \qquad \theta = arctan(D_y/D_x)$$

But instead of calculating these values pixelwise we are going to average them in blocks R of $w \times w$ pixels. So that opposite gradient vectors don't offset each other we double the angles. Allowing the magnitude to be squared we have:

$$\alpha_y = \rho^2 sin(2\theta) \qquad \alpha_x = \rho^2 cos(2\theta)$$

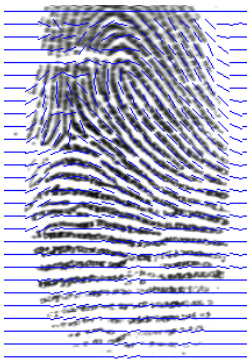


Figura 4.1: Orientation blocks for image 10_1.png in FVC2004DB1A

$$\alpha_y = \rho^2(2cos(\theta)sin(\theta)) \qquad \alpha_x = \rho^2\left(cos(\theta)^2 - sin(\theta)^2\right)$$

$$\alpha_y = 2D_xD_y \qquad \alpha_x = D_x^2 - D_y^2$$

$$\tilde{\alpha_y} = \frac{1}{w^2}\sum_R \alpha_y \qquad \tilde{\alpha_x} = \frac{1}{w^2}\sum_R \alpha_x$$

$$\theta = \frac{1}{2}arctan2\left(\tilde{\alpha_y}, \tilde{\alpha_x}\right)$$

The following code is the implementation in python using numpy of this step after the convolution.

```
def gradientAux(img, blk_sz, dx, dy):
      alpha_x = dx*dx - dy*dy
      alpha_y = 2 * dx * dy

      alpha_x_block = [[np.sum(
            alpha_x[idx_y: idx_y + blk_sz, idx_x: idx_x + blk_sz]
            ) / blk_sz**2
      for idx_x in range(0, img.shape[0], blk_sz)]
      for idx_y in range(0, img.shape[1], blk_sz)]

      alpha_y_block = [[np.sum(
            alpha_y[idx_y: idx_y + blk_sz, idx_x: idx_x + blk_sz]
            )/ blk_sz**2
      for idx_x in range(0, img.shape[0], blk_sz)]
      for idx_y in range(0, img.shape[1], blk_sz)]

      alpha_x_block = np.array(alpha_x_block)
      alpha_y_block = np.array(alpha_y_block)

      orientation_blocks = np.arctan2(alpha_y_block, alpha_x_block) / 2

      return orientation_blocks
```

With the orientation of the blocks, to extract the orientation of a given minutiae by its pixel coordinates, just find the orientation of the block it is in. An example of orientation extraction can be seen in Figures 4.1 and 4.2.

Figura 4.2: Orientation blocks for image 10_2.png in FVC2004DB1A

4.5 CONCLUSION

Different network configurations will have to be tested against each other to evaluate the performance gains of different YOLO designs. It's expected that the specialized anchors in YOLO will affect the performance greatly, since objects in this task are of fixed aspect ration and invariant size. With two different sets of labels available to us for FVC2004, the matching performance of these different systems will vary. One system will be able to classify minutiae types while the other only locates them. It's likely the type information will improve matching performance considerably.

Since YOLO does not provide any angle information we employ a classical approach with Sobel operators. This is just an approximation and may not have great success.

5 EXPERIMENTS

These experiments will test how well the adapted detection network configurations locate fingerprint minutiae, and how the extracted minutiae information affects the performance of the SourceAFIS's fingerprint matching. For the evaluation of located minutiae a precision-recall curve will be plotted comparing with the ground truth positions in the test sets described in each scenario. For the evaluation of the fingerprint matcher a Receiver Operating Characteristic (ROC) curve is plotted to visualize tradeoffs in True Positive Rate and False Positive Rate in verification, as well as overall performance with Area Under Curve values.

5.1 SCENARIO

Using the finger verification competition 2004 set A images and labels, the dataset was divided into two different ways, both with 400 images in train and test sets.

1. One in a open world scenario where no finger in the training set appears in the validation set, half of the fingers (50) in each set.

2. And another in a closed world scenario where half of the fingerprints of the same finger (4) were separated into the train and validation sets.

The base configurations for the experiments are YOLOv2, YOLOv2 tiny, YOLOv2 VOC, YOLOv3, YOLOv3 tiny, YOLOv3 SPP. The configurations have been adapted with the changes described in section 4.1.

The configurations are named in the format: base configuration, anchor type, box size. Where the base configurations are one of the ones described above. The anchor type can be *anchor0* for the default anchors of YOLO configurations or *anchor1* and *anchor5* for the customized anchors for minutiae detection [1]. The box size is the size used in the translation if point labels of the original dataset to bounding boxes required by YOLO described in section 4.2. For example *v3-spp2-anchor1-box-30* uses YOLOv3 SPP base configuration, custom anchors and 30 pixels of minutiae bounding box size in the labels.

In the configuration v3-spp3 we set to test these prediction layers. It is seen in Figure 5.7. The first two of the three prediction layers in YOLOv3 are suited for bigger objects than we desire to detect. Therefore in this configuration these two first prediction layers were removed and another prediction layer further down was added.

Networks were trained from zero in 3500 epochs, with burn in of 1000, learning rate of 0.001 with multiplier 0.1 at epochs 2000 and 3000.

Precision recall curves were made of different configurations, these were sampled by changing each network's threshold. Minutiae detected within 16 pixels of euclidean distance are considered matches; [2] Furthermore they are one to one matches and the angle corectness is not taken into account, testing only localization. The precision-recall curves, trained and tested with FingerNet labels, for the open scenario can be seen in

[1] *anchor1* is the custom v3 anchor (18×25 in pixels), *anchor5* is the custom v2 anchor (0.5750×0.7667 relative to image size)

[2] Equal to the 16 pixels used by MinutiaeNet and close to FingerNet's 15 pixel distance (MENet was visually compared, not automatically).

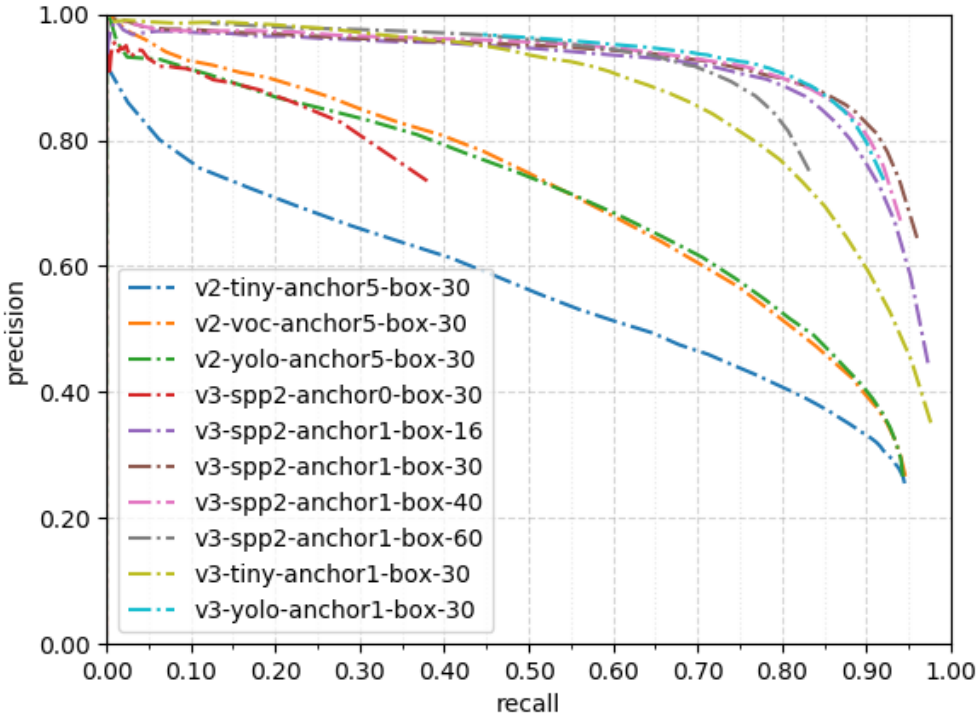


Figura 5.1: Precision-recall curves, using FingerNet labels, of different network configurations in a open world scenario

Figure 5.1 and the curves for the closed scenario in Figure 5.2. Area Under Curve values can be compared in Tables 5.1 and 5.2. The point of closest equal precision-recall in the curve of v3-spp2-anchor1-box-30 achieved a precision of 86.67% a recall of 86.09% using a 0.2 threshold in bounding box confidence. As expected, the default anchors, seen in the v3-spp2-anchor0 configuration, performed terribly when compared to specialized anchors of v3-spp2-anchor1 configurations.

Although, the precision-recall tests perform did not take angle correctness into consideration, only location; The top performing network configurations have close performance when compared to state of the art FingerNet and MinutiaeNet. MinutiaeNet (Nguyen et al., 2018) achieved 85.9% precision and 84.8% recall. Also in FVC2004 set A [3] with pixel distance 16, but they take angle correctness into account, 30 degrees maximum difference. While the YOLOv3 SPP configuration achieved 86.67% precision and 86.09% recall using a 0.2 threshold. Since YOLO is the fastest available network design, these results are promising for fingerprint matching. Further specializations could be made in the network to detect smaller objects exclusively.

To evaluate matching performance multiple verification tests were made. They were constructed using the FVC2004 protocol for recognition attempts (Maio et al., 2004); Since our test dataset is 0.5 of the original set A size, 1400 genuine recognition attempts and 1225 impostor recognition attempts were made for each curve. With 50 being the number of different fingers we have:

Genuine attempts $= 50 * \sum_{n=1}^{7} n$

Impostor attempts $= \sum_{n=1}^{50-1} n$

[3] MinutiaeNet tested with whole set A as opposed to our 50%, they trained on FVC2002

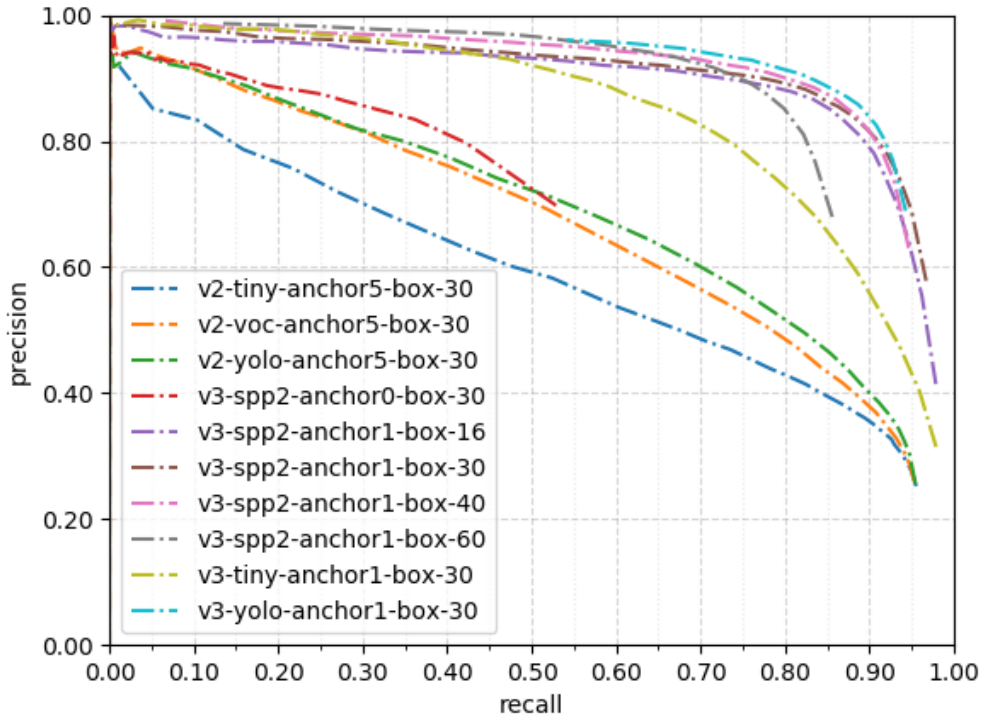


Figura 5.2: Precision-recall curves, using FingerNet labels, of different network configurations in a closed world scenario

Tabela 5.1: Open set Area Under Curve (precision-recall)

configuration	**AUC**
v2-tiny-anchor5-box-30	0.54122
v2-voc-anchor5-box-30	0.68872
v2-yolo-anchor5-box-30	0.68209
v3-spp2-anchor0-box-30	0.32793
v3-spp2-anchor1-box-16	0.88872
v3-spp2-anchor1-box-30	**0.89371**
v3-spp2-anchor1-box-40	0.86373
v3-spp2-anchor1-box-60	0.66968
v3-tiny-anchor1-box-30	0.84892
v3-yolo-anchor1-box-30	0.43711

Tabela 5.2: Closed set Area Under Curve (precision-recall)

configuration	**AUC**
v2-tiny-anchor5-box-30	0.56851
v2-voc-anchor5-box-30	0.65874
v2-yolo-anchor5-box-30	0.67513
v3-spp2-anchor0-box-30	0.45210
v3-spp2-anchor1-box-16	**0.88515**
v3-spp2-anchor1-box-30	0.88328
v3-spp2-anchor1-box-40	0.82253
v3-spp2-anchor1-box-60	0.67977
v3-tiny-anchor1-box-30	0.82913
v3-yolo-anchor1-box-30	0.36619

The matching performance evaluations of extracted minutiae used the angle extraction technique described Section in 4.4, unless specified otherwise. [4] Configurations trained with FM3 labels do classify types while FingerNet labels only locate minutiae.

To analyze ROC curves, first different thresholds using the same network configuration were compared in Figures 5.3 and 5.4. Figure 5.3 has a bigger threshold stride, 10%, to visualize big threshold changes. The same goes for Figure 5.5, however in this instance networks were trained with FM3 labels. It can be seen that this matcher performs strictly better with batches of minutiae with higher recall, although they have less precision. Illustrated are the curve with 0.8 to 0.1 threshold, each decreasing step in threshold dominating the previous curve's performance, ending in 0.1 with the biggest area. While Figure 5.4 has smaller threshold strides to see how the matcher behaves in higher detail, with a bigger quantity of less precise minutiae, that also bring higher recall.

To compare different networks matching performance the ROC curves of different networks were plotted, all using the same 0.1 threshold. In Figure 5.6 they were trained with FingerNet labels, while in Figure 5.7 they were trained with FM3 labels. Our speculations were confirmed, YOLOv2 performed far worse then even the tiny version of YOLOv3. Also, the removal of the first two prediction layers from YOLOv3 didn't impact the performance of the v3-spp3 configuration.

Relevant configurations can be compared against SourceAFIS native implementation in Figure 5.8, all networks using 0.1 threshold. [5] Sadly none of them outperformed SourceAFIS. The best performing configuration has equal evaluation to SourceAFIS in a True Positive Rate (TPR) of 84.5% and False Positive Rate (FPR) of 15%. But SourceAFIS dominates in values of lower FPR in the ROC curve, having higher TPR. Even so, the proposed approach has a higher area than SourceAFIS in the curve. We suspect this lack of performance caused by the method of angle approximation. This is evidenced by the curve which ignores angles, v2-spp3-100-noAngle, it outperforms other configurations. Also, the precision and recall obtained surpassed both FingerNet and MinutiaeNet, when not considering angles (Figure 5.1). A better technique for angle extraction could be easily be swapped for the current Orientation Map. Such as the one employed in MENet that uses local orientation estimation (Khan, 2011). A better angle extraction and a type robust matcher a such as Minutiae Cylinder Code (Cappelli et al., 2010) might prove to

[4] "noAngle"configurations

[5] SourceAFIS reports to be robust to imprecise minutiae, so a tradeoff for a better recall is made by choosing a threshold half of 0.2 that had equal precision-recall reported earlier.

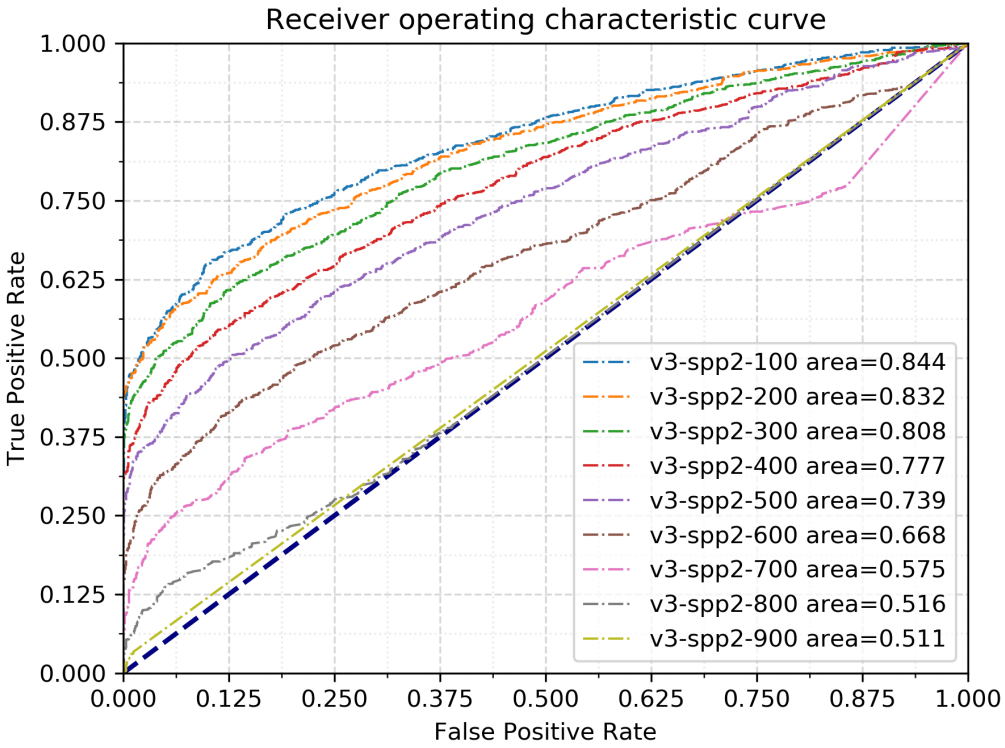


Figura 5.3: ROC curves, using FingerNet labels, of the v3-spp network using different YOLO detection thresholds, 100 means it's a threshold of 0.100

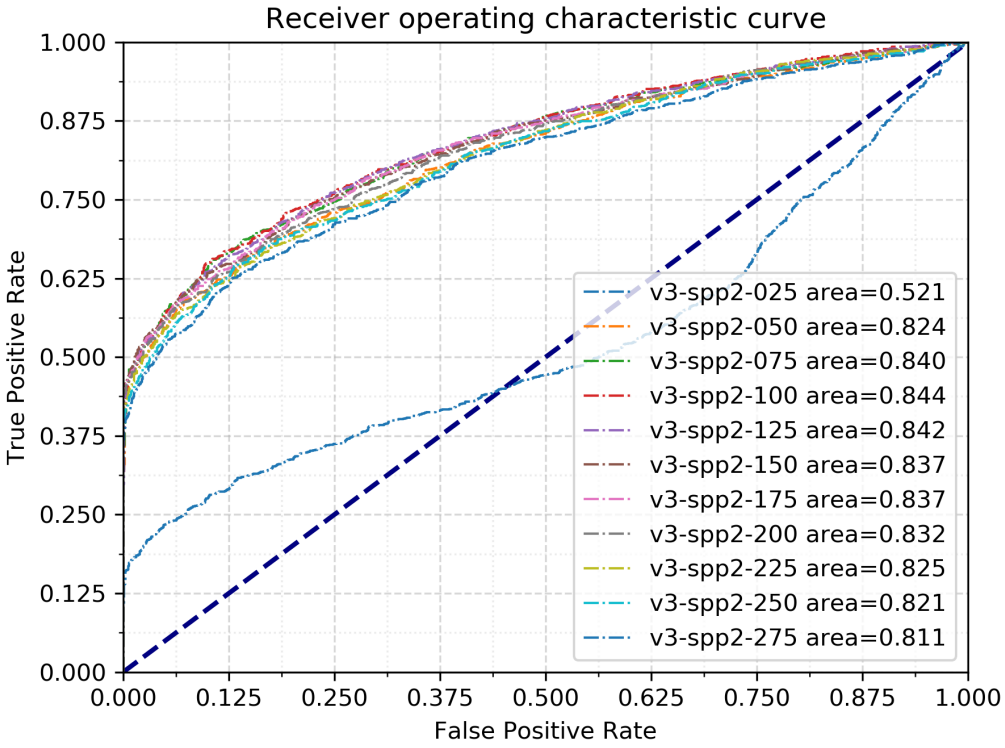


Figura 5.4: ROC curves, using FingerNet labels, of the v3-spp network using different small detection thresholds, 100 means it's a threshold of 0.100

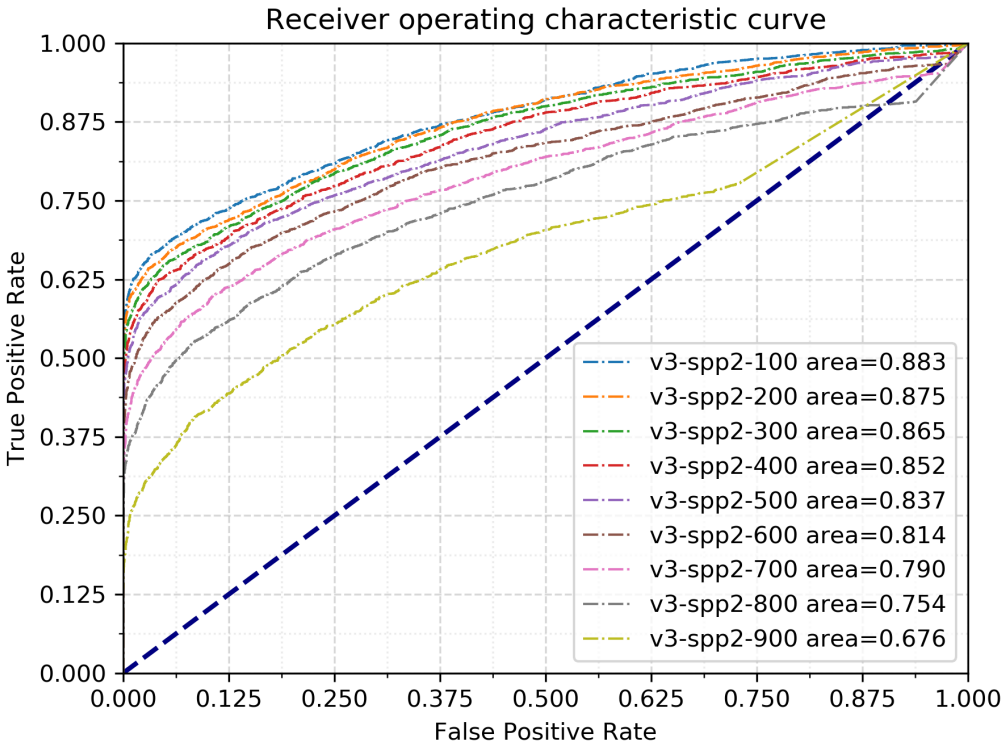


Figura 5.5: ROC curves, using FM3 labels, of the v3-spp network using different YOLO detection thresholds

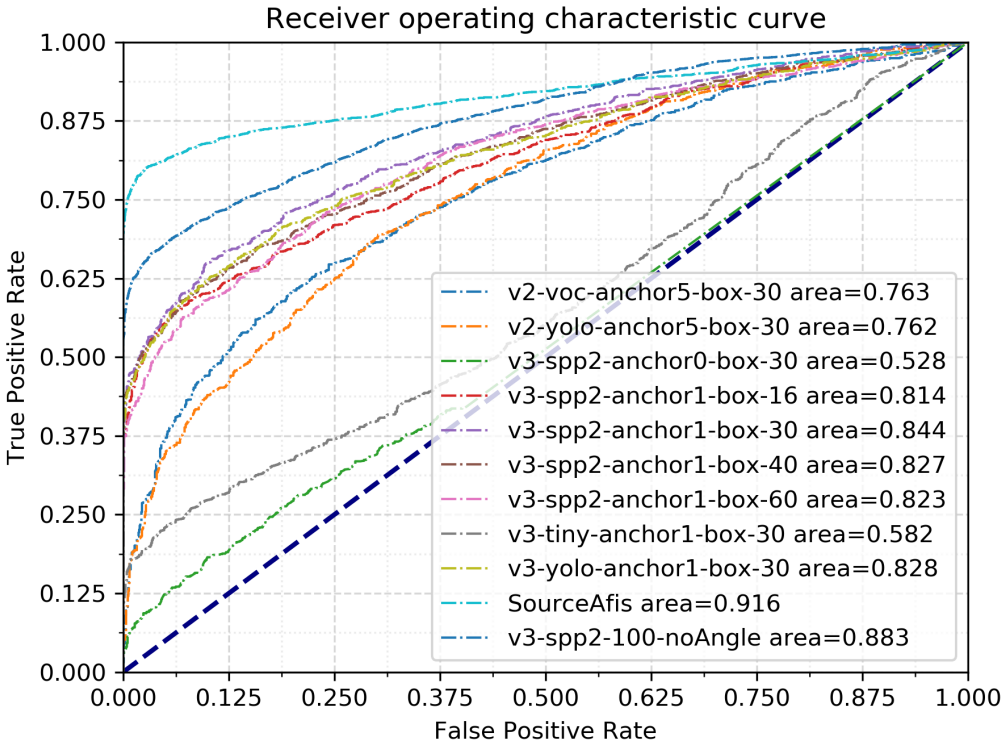


Figura 5.6: ROC curves, using FingerNet labels, of different network configurations in a open world scenario

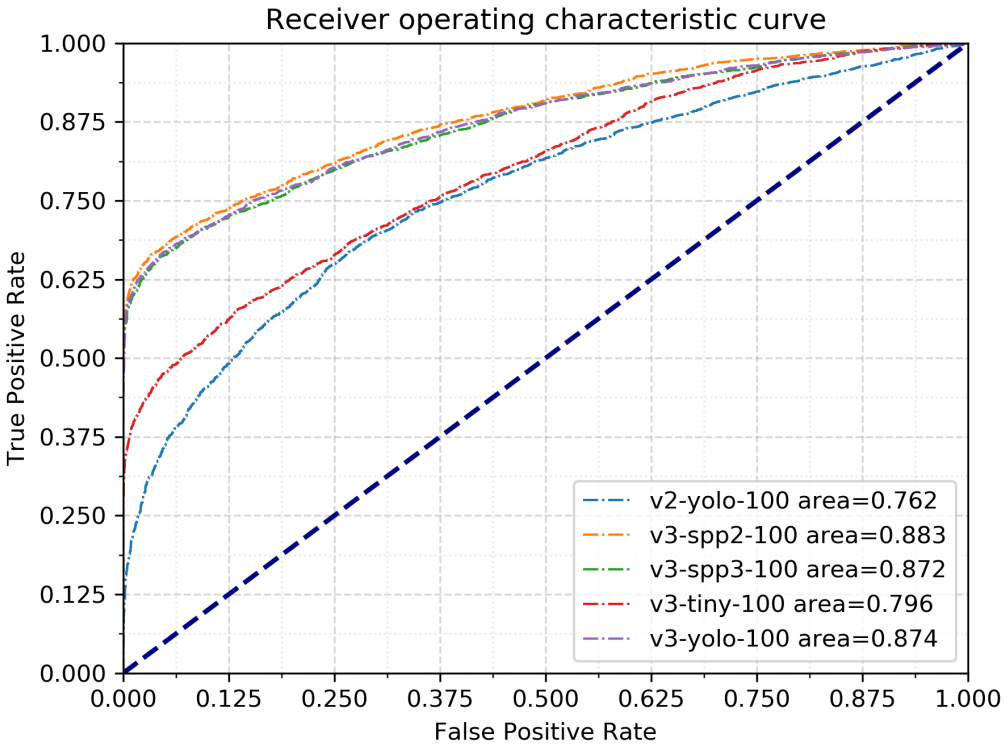


Figura 5.7: ROC curves, using FM3 labels, of different network configurations in a open world scenario

be sufficient to beat classical approaches with good margin. Tests using another matcher, like Minutiae Cylinder Code (Cappelli et al., 2010) that does not use type information, and a better angle extraction method would need to be done to confirm this.

Its hard to compare the matching performance to MENet, where they achieved 90% True Positive Rate with a 1% False Positive Rate. Since they use a different matcher with a performance difference unknown to us from SourceAFIS's matcher algorithm. Even so, MENet uses a slow sliding windows approach, that, while takes a lot of processing time, is known to be quite effective.

5.2 CONCLUSION

The networks tested here obtained competitive results with state of the art Deep Learning minutiae detectors against the FVC2004 dataset. As expected, YOLOv3 performed well in this task of small object detection, while YOLOv2 struggled, not outperforming even YOLOv3 Tiny.

Closed set configurations did not show significant improvement in precision and recall when compared to Open set. Such is small evidence that data augmentation properly decreases the likelihood of the network memorizing input data.

Matching performance leaves to be desired, not beating SourceAFIS native implementation while in FPR lower than 15%. The most relevant comparisons between tested configurations are in Figure 5.8. They have equal values in the point True Positive Rate (TPR) 84.5% and False Positive Rate (FPR) 15%. Better angle extraction and a type robust matcher a such as Minutiae Cylinder Code (Cappelli et al., 2010) might prove to be sufficient to beat classical approaches with good margin. As evidence, the precision and

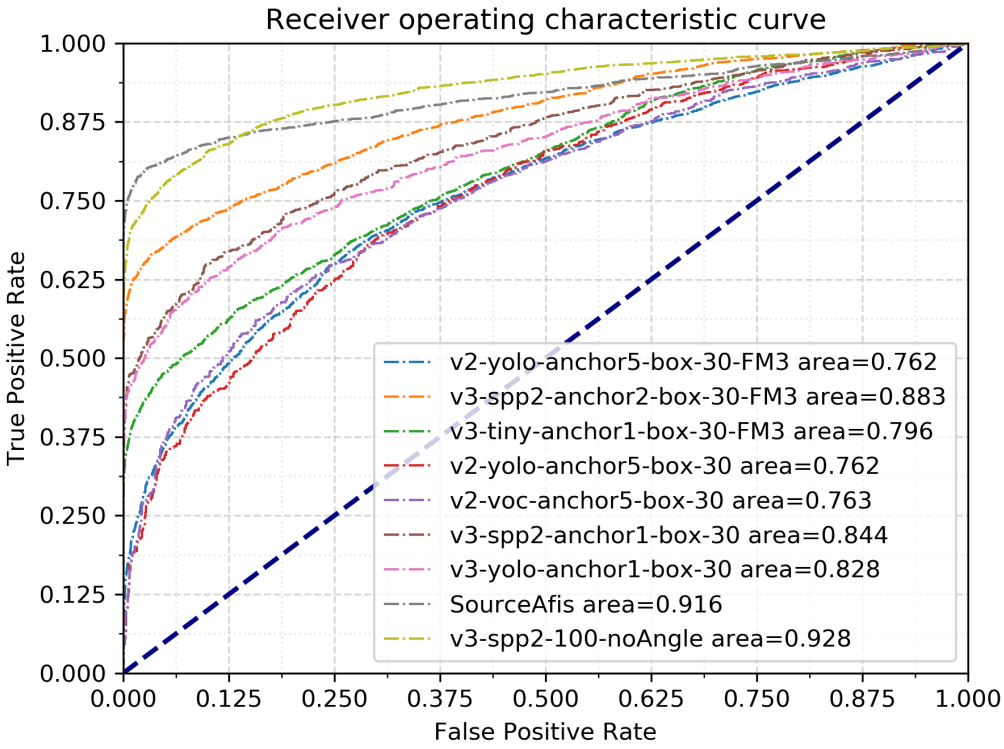


Figura 5.8: ROC curves of different network configurations in a open world scenario, configurations suffixed with FM3 were trained with those labels

recall achieved in the FVC2004 were comparable to state of the art performance. Even though the approach proposed doesn't extract angles well, a better technique could be easily be swapped for the current Orientation Map. A distinguishable feature is that it extracts type information, which none of the other state of the art systems referenced do.

6 CONCLUSION

In this study we analyzed multiple Convolutional Neural Network detector architectures. A lot of progress has been made both in performance and speed of the designs, from Selective Search to pure regressors and multi-scale predictions. The goal of adapting a well known modern network, YOLO, to detect fingerprint minutiae was successful. The best results used the YOLOv3 SPP design, though it doesn't have much difference to the non-SPP one. It achieved 86.67% precision and 86.09% recall against the FingerNet minutiae labels for FVC2004. This performance achieved was better than expected considering the slower specialized MinutiaeNet (Nguyen et al., 2018) achieved 85.9% precision and 84.8% recall in the same FVC2004 database. Although the comparison can't be made directly, since MinutiaeNet considers angle correctness when calculating precision while the proposed approach doesn't, it is still good evidence for the localization task.

YOLO adapted well for this change of tasks, the detection of multiple small details in a single image. While the earlier version, YOLOv2, didn't perform that well, YOLOv3 improved the results by a large margin, even when not using the configuration with the spatial pooling pyramid. None of the previous works mentioned here had a design quite like the pure regressor of YOLO.

As for the matching the performance of the extracted minutiae data, it is affected heavily by low quality of angle information. By using a better technique for the angle calculation one could significantly improve the networks performance, since it was able to achieve state of the art detection. Though type information given by networks trained with FM3 labels did improve matching accuracy, when compared to same design networks trained with no type classification (FingerNet labels). A distinguishable feat since the other researched networks do not attempt to classify the type of minutiae.

REFERÊNCIAS

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. et al. (2016). Tensorflow: a system for large-scale machine learning. Em *OSDI*, volume 16, páginas 265–283.

Adelson, E. H., Anderson, C. H., Bergen, J. R., Burt, P. J. e Ogden, J. M. (1984). Pyramid methods in image processing. *RCA engineer*, 29(6):33–41.

Cappelli, R., Ferrara, M., Franco, A. e Maltoni, D. (2007). Fingerprint verification competition 2006. *Biometric Technology Today*, 15(7-8):7–9.

Cappelli, R., Ferrara, M. e Maltoni, D. (2010). Minutia cylinder-code: A new representation and matching technique for fingerprint recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(12):2128–2141.

Chollet, F. (2015). Keras: The python deep learning library.

Collobert, R., Kavukcuoglu, K. e Farabet, C. (2016). Torch: A scientific computing framework for luajit.

Dai, J., Li, Y., He, K. e Sun, J. (2016). R-fcn: Object detection via region-based fully convolutional networks. Em *Advances in neural information processing systems*, páginas 379–387.

Darlow, L. N. e Rosman, B. (2017). Fingerprint minutiae extraction using deep learning. Em *Biometrics (IJCB), 2017 IEEE International Joint Conference on*, páginas 22–30. IEEE.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. e Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. Em *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, páginas 248–255. Ieee.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E. e Darrell, T. (2014). Decaf: A deep convolutional activation feature for generic visual recognition. Em *International conference on machine learning*, páginas 647–655.

Duchi, J., Hazan, E. e Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Everingham, M., Van Gool, L., Williams, C. K., Winn, J. e Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J. e Zisserman, A. (2007). The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J. e Zisserman, A. (2012). The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

FBI (1984). *The Science of Fingerprints: Classification and Uses*. Washington, D.C., U.S. Government Printing Office.

Fu, X., Liu, C., Bian, J., Feng, J., Wang, H. e Mao, Z. (2013). Extended clique models: A new matching strategy for fingerprint recognition. Em *Biometrics (ICB), 2013 International Conference on*, páginas 1–6. IEEE.

Garcia-Gasulla, D., Parés, F., Vilalta, A., Moreno, J., Ayguadé, E., Labarta, J., Cortés, U. e Suzumura, T. (2017). On the behavior of convolutional nets for feature extraction. *arXiv preprint arXiv:1703.01127*.

Girshick, R. (2015). Fast r-cnn. Em *Proceedings of the IEEE international conference on computer vision*, páginas 1440–1448.

Girshick, R., Donahue, J., Darrell, T. e Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 580–587.

Glorot, X., Bordes, A. e Bengio, Y. (2011). Deep sparse rectifier neural networks. Em Gordon, G., Dunson, D. e Dudík, M., editores, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 de *Proceedings of Machine Learning Research*, páginas 315–323, Fort Lauderdale, FL, USA. PMLR.

He, K., Zhang, X., Ren, S. e Sun, J. (2014). Spatial pyramid pooling in deep convolutional networks for visual recognition. Em *European conference on computer vision*, páginas 346–361. Springer.

He, K., Zhang, X., Ren, S. e Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Em *Proceedings of the IEEE international conference on computer vision*, páginas 1026–1034.

He, K., Zhang, X., Ren, S. e Sun, J. (2016). Deep residual learning for image recognition. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 770–778.

Huang, P., Chang, C.-Y. e Chen, C.-C. (2007). Implementation of an automatic fingerprint identification system. Em *2007 IEEE International Conference on Electro/Information Technology*, páginas 412–415. IEEE.

Huang, Z. e Wang, J. (2019). Dc-spp-yolo: Dense connection and spatial pyramid pooling based yolo for object detection. *arXiv preprint arXiv:1903.08589*.

Ioffe, S. e Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jain, A. K., Nandakumar, K. e Ross, A. (2016). 50 years of biometric research: Accomplishments, challenges, and opportunities. *Pattern Recognition Letters*, 79:80–105.

Kayaoglu, M., Topcu, B. e Uludag, U. (2013). Standard fingerprint databases: Manual minutiae labeling and matcher performance analyses. *arXiv preprint arXiv:1305.1443*.

Khan, M. A. (2011). Fingerprint image enhancement and minutiae extraction.

Kingma, D. P. e Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I. e Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Em *Advances in neural information processing systems*, páginas 1097–1105.

Lazebnik, S., Schmid, C. e Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. Em *null*, páginas 2169–2178. IEEE.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B. e Belongie, S. (2017a). Feature pyramid networks for object detection. Em *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, páginas 2117–2125.

Lin, T.-Y., Goyal, P., Girshick, R., He, K. e Dollár, P. (2017b). Focal loss for dense object detection. Em *Proceedings of the IEEE international conference on computer vision*, páginas 2980–2988.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. e Zitnick, C. L. (2014). Microsoft coco: Common objects in context. Em *European conference on computer vision*, páginas 740–755. Springer.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y. e Berg, A. C. (2016). Ssd: Single shot multibox detector. Em *European conference on computer vision*, páginas 21–37. Springer.

Maio, D., Maltoni, D., Cappelli, R., Wayman, J. L. e Jain, A. K. (2002a). Fvc2000: Fingerprint verification competition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):402–412.

Maio, D., Maltoni, D., Cappelli, R., Wayman, J. L. e Jain, A. K. (2002b). Fvc2002: Second fingerprint verification competition. Em *Pattern recognition, 2002. Proceedings. 16th international conference on*, volume 3, páginas 811–814. IEEE.

Maio, D., Maltoni, D., Cappelli, R., Wayman, J. L. e Jain, A. K. (2004). Fvc2004: Third fingerprint verification competition. Em *Biometric Authentication*, páginas 1–7. Springer.

Maltoni, D., Maio, D., Jain, A. K. e Prabhakar, S. (2009). *Handbook of fingerprint recognition*. Springer Science & Business Media.

Nguyen, D.-L., Cao, K. e Jain, A. K. (2018). Robust minutiae extractor: Integrating deep networks and fingerprint domain knowledge. Em *2018 International Conference on Biometrics (ICB)*, páginas 9–16. IEEE.

Perez, L. e Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*.

Prabhakar, S., Jain, A. K. e Pankanti, S. (2003). Learning fingerprint minutiae location and type. *Pattern recognition*, 36(8):1847–1857.

Redmon, J., Divvala, S., Girshick, R. e Farhadi, A. (2016). You only look once: Unified, real-time object detection. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 779–788.

Redmon, J. e Farhadi, A. (2017). Yolo9000: better, faster, stronger. *arXiv preprint*.

Redmon, J. e Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Ren, S., He, K., Girshick, R. e Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. Em *Advances in neural information processing systems*, páginas 91–99.

Simonyan, K. e Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. e Rabinovich, A. (2015). Going deeper with convolutions. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 1–9.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. e Wojna, Z. (2016). Rethinking the inception architecture for computer vision. Em *Proceedings of the IEEE conference on computer vision and pattern recognition*, páginas 2818–2826.

Tang, Y., Gao, F. e Feng, J. (2017a). Latent fingerprint minutia extraction using fully convolutional network. Em *Biometrics (IJCB), 2017 IEEE International Joint Conference on*, páginas 117–123. IEEE.

Tang, Y., Gao, F., Feng, J. e Liu, Y. (2017b). Fingernet: An unified deep network for fingerprint minutiae extraction. Em *Biometrics (IJCB), 2017 IEEE International Joint Conference on*, páginas 108–116. IEEE.

Taylor, L. e Nitschke, G. (2017). Improving deep learning using generic data augmentation. *arXiv preprint arXiv:1708.06020*.

Uijlings, J. R., Van De Sande, K. E., Gevers, T. e Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, 104(2):154–171.

Vazan, R. (2009). Source afis—fingerprint recognition library for. net and experimentally for java.

Wang, Y., Perazzi, F., McWilliams, B., Sorkine-Hornung, A., Sorkine-Hornung, O. e Schroers, C. (2018). A fully progressive approach to single-image super-resolution. Em *CVPR Workshops*.

Xie, J., Xu, L. e Chen, E. (2012). Image denoising and inpainting with deep neural networks. Em *Advances in neural information processing systems*, páginas 341–349.

Yang, X., Feng, J. e Zhou, J. (2014). Localized dictionaries based orientation field estimation for latent fingerprints. *IEEE transactions on pattern analysis and machine intelligence*, 36(5):955–969.

Zhao, F. e Tang, X. (2007). Preprocessing and postprocessing for skeleton-based fingerprint minutiae extraction. *Pattern Recognition*, 40(4):1270–1281.