Biometric Technologies and Behavioural Security

Tutorial 3 - Fingerprint Liveness Detection (handcrafted/deep learning)

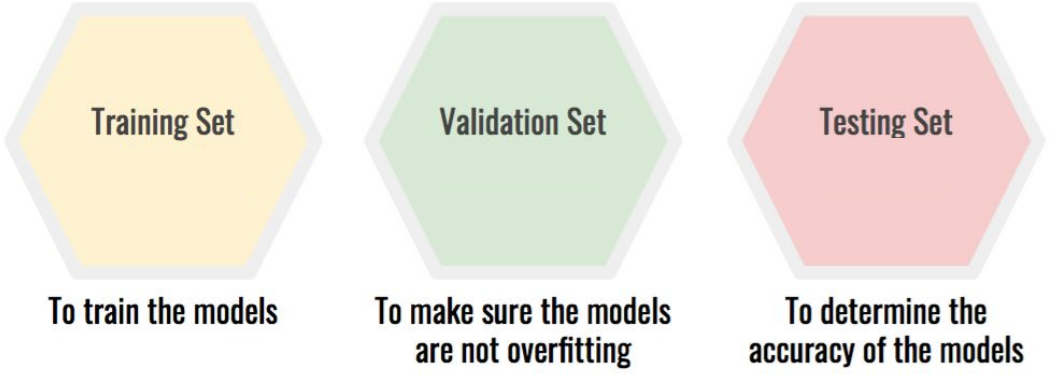
LECTURER:
GIAN LUCA MARCIALIS
TEACHING ASSISTANTS:
ROBERTO CASULA, SIMONE MADAU, MARCO MICHELETTO, GIULIA ORRU'
(roberto.casula, marco.micheletto, giulia.orru)@unica.it , madausimone@gmail.com

First homework solution: Fingerprint and faces classifier

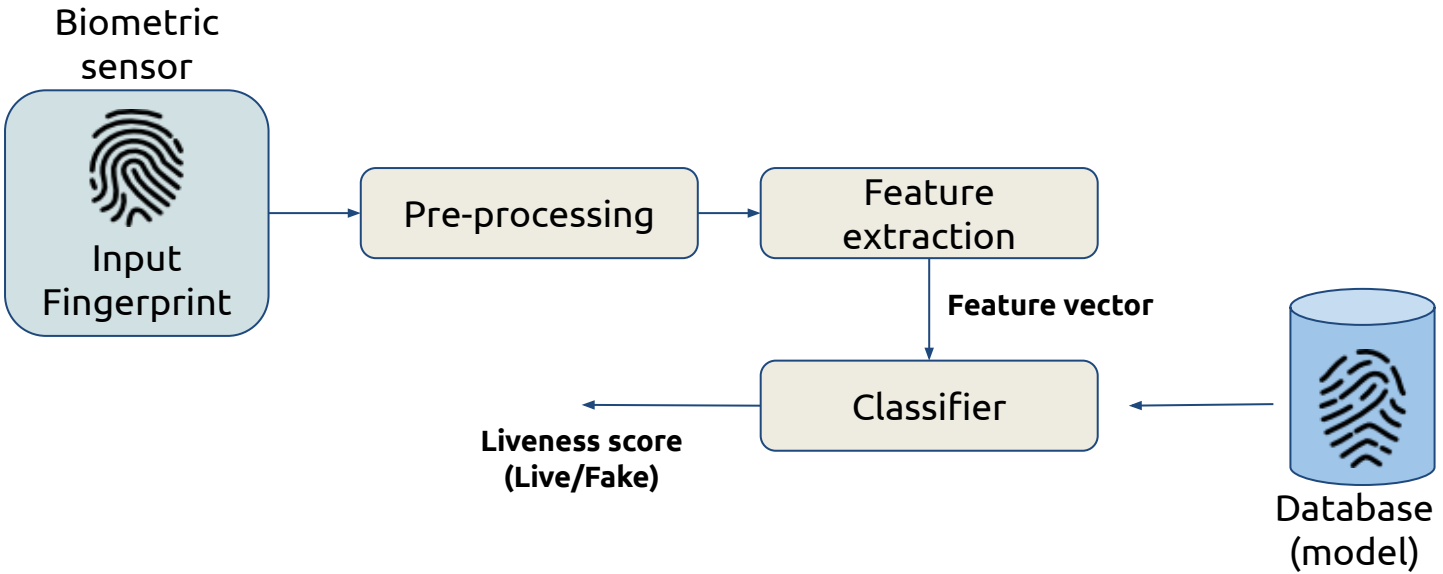
https://bit.ly/2XnIdIs

Machine Learning sets

- Training set: A set of examples used for learning, that is to fit the parameters of the classifier.
- Validation set: A set of examples used to tune the parameters of a classifier, for example to choose the number of hidden units in a neural network. []
- Test set: A set of examples used only to assess the performance of a fully-specified classifier.

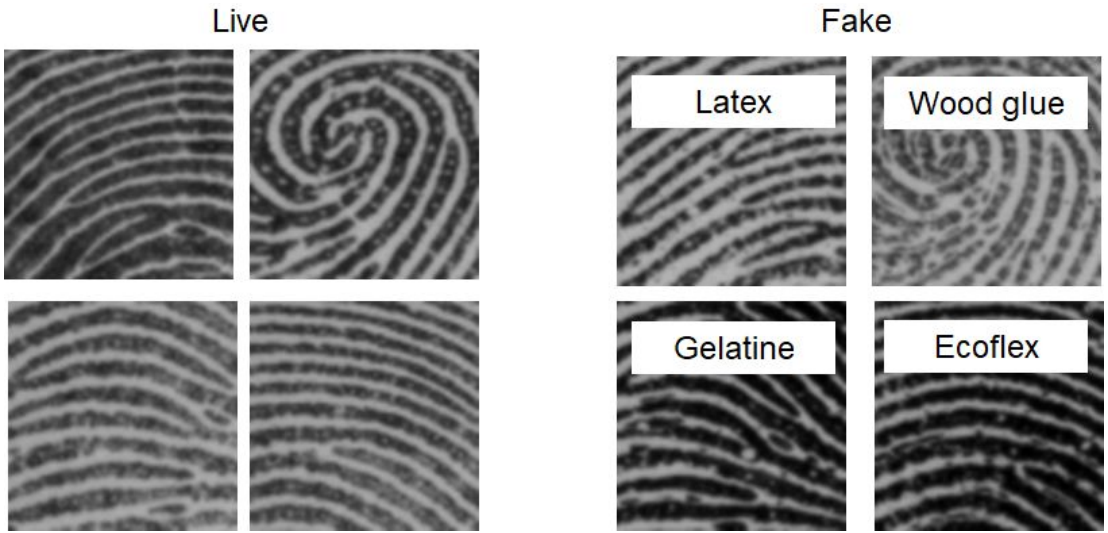


Liveness Detection system (Presentation Attack Detection PAD)



Feature extraction: Keyword is learning

Grey level distributions of live and fake images exhibit strong local variations that cannot be pointed out by visual inspection



Feature extraction: different methods

EXAMPLE:

Both LBP and LPQ describe each pixel neighboring by a binary code obtained by convolution of the image with a manually predefined set of filters.

- LBP works in the image domain
- LPQ works in the frequency domain

Binarized Statistical Image Features (BSIF) generalize this concept in the image domain, by applying a «learning» step to derive a statistical meaningful set of filters.

BSIF: Binarized Statistical Image Features

- calculates a binary code string for the pixels of a given image
- the value of each pixel is considered as a local descriptor and may be used to build the histograms that allow to characterize the textural properties within subregions of the image
- each bit in the binary code string is calculated binarizing the response of a linear filter with a threshold at zero

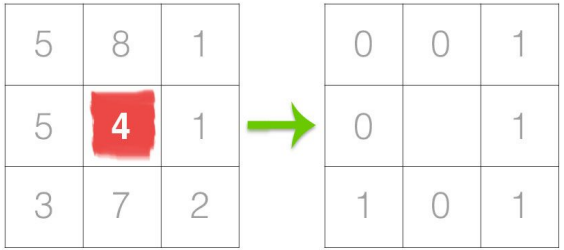
https://bit.ly/2yg8Nc4

LBP: Local Binary Pattern for texture classification

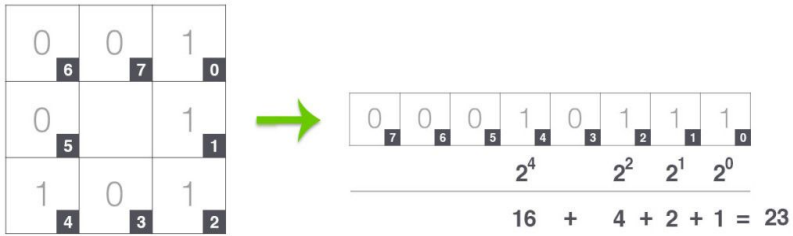
- Textural descriptor
- extracts certain uniform patterns corresponding to micro-features in the image
- LBP looks at points surrounding a central point and tests whether the surrounding points are greater than or less than the central point

LBP

1° step: take the n pixel neighborhood surrounding a center pixel and threshold it to construct a set of n binary digits

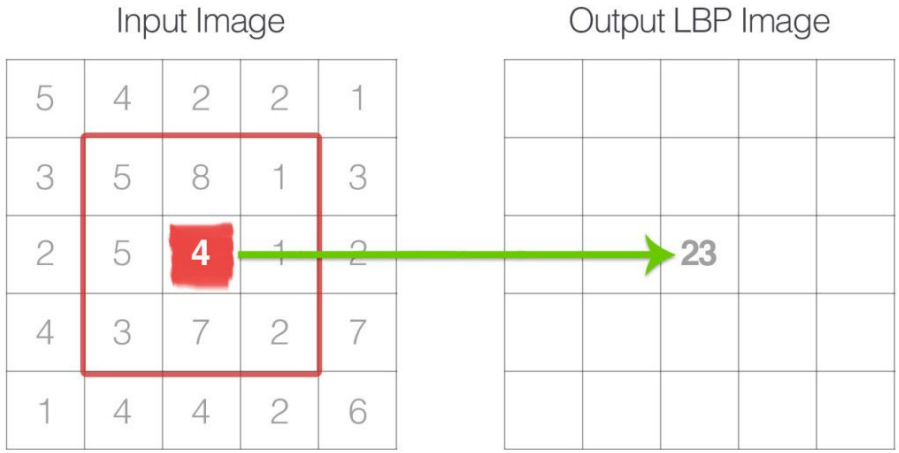


2° step: take the n-bit binary neighborhood of the center pixel and converting it into a decimal representation.



LBP

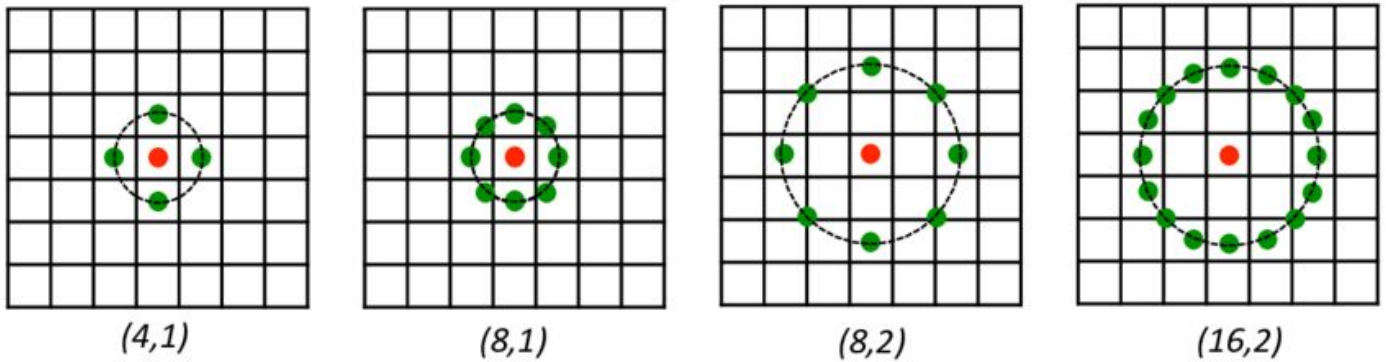
3° step: the calculated LBP value is then stored in an output array with the same width and height as the original image.



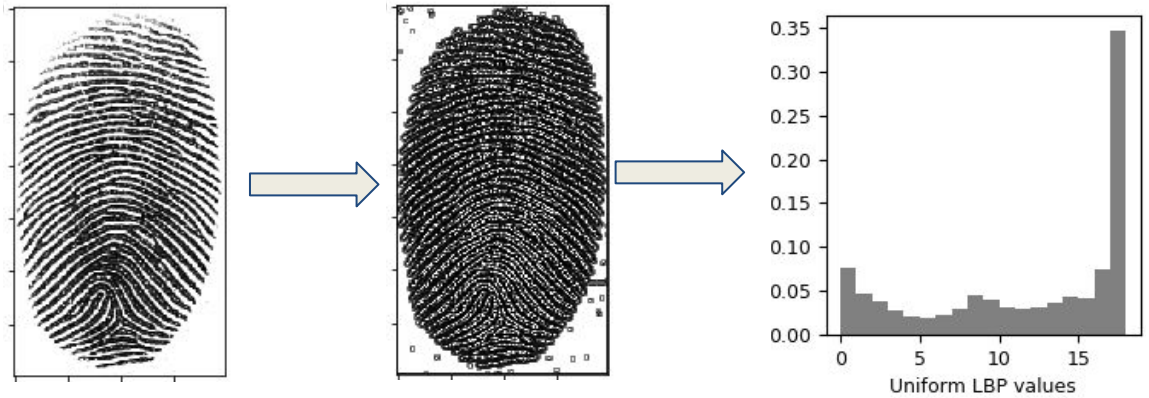
LBP extension

Proposed by Ojala et al. to handle variable neighborhood sizes.

- The number of points p in a circularly symmetric neighborhood to consider (thus removing relying on a square neighborhood).
- The radius of the circle r, which allows us to account for different scales.



Feature extraction: LBP



Input image

LBP image

LBP histogram

Feature vector

LBP-based PAD classifier

https://bit.ly/3clKlVs

LBP extention

```
from skimage import feature
lbp = feature.local_binary_pattern(image,numPoints,
     radius, method="uniform")
```

https://scikit-image.org/docs/dev/api/skimage.feature.html#skimage.feature.local_binary_pattern

Keras– Build a Neural Network

- **Preprocess and load data** -Process the date before feeding to the neural network.
- **Define model** - Define the number and the size of hidden layers in the neural network,, the input and output size.
- **Loss and optimizer** - Define the loss function according to the task and specify the optimizer to use with learning rate and other hyperparameters.
- **Fit model** - The training step of the neural network, you should define the number of epochs to train the neural network.

Keras – Build a Neural Network

Empirical rules

Table: Determining the Number of Hidden Layers

Num Hidden Layers	Result
none	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.
>2	Additional layers can learn complex representations (sort of automatic feature engineering) for layer layers.

Keras – Build a Neural Network

Empirical rules

NUMBER OF HIDDEN NEURONS:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

Keras – Build a Neural Network

```
#Dependencies
import keras
from keras.models import Sequential
from keras.layers import Dense
# Neural network
model = Sequential()
model.add(Dense(x, input_dim=i, activation='relu'))
model.add(Dense(y, activation='relu'))
model.add(Dense(k, activation='sigmoid'))
```

Type of layer: the Dense is used to specify the fully connected layer.

Sequential specifies to keras that we are creating model sequentially and the output of each layer we add is input to the next layer we specify.

model.add is used to add a layer to our neural network.

Keras – Build a Neural Network

```
#Dependencies
import keras
from keras.models import Sequential
from keras.layers import Dense
# Neural network
model = Sequential()
model.add(Dense(x, input_dim=i, activation='relu'))
model.add(Dense(y, activation='relu'))
model.add(Dense(k, activation='sigmoid'))
```

As first layer in a sequential **model.add(Dense(x,input_shape=(i,)))** the model will take as input arrays of shape (*, i) and output arrays of shape (*, x)

after the first layer, you don't need to specify the size of the input anymore: **model.add(Dense(32))**

Keras – Build a Neural Network

```
#Dependencies
import keras
from keras.models import Sequential
from keras.layers import Dense
# Neural network
model = Sequential()
model.add(Dense(x, input_dim=i, activation='relu'))
model.add(Dense(y, activation='relu'))
model.add(Dense(k, activation='sigmoid'))
```

x-> first hidden layers output
i->first layer input = # features
y->second hidden layers output
k-> last layer output

Keras – Build a Neural Network

```
model.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
history = model.fit(X_train,y_train,
        batch_size=b,
        epochs=e,
        validation_data=(X_test, y_test),
        shuffle=True)
```

- Specify the loss function and the optimizer
- Train the NN

Exercise: Fingerprint PAD Neural Network

- Use the previous fingerprint dataset
- Train a neural network with two hidden layers
- Predict the test set labels and compute the accuracy

Exercise: Fingerprint PAD Neural Network

https://bit.ly/34F8Pq7