IRAFM

Car plates identification and recognizition

Institute for Research and Applications of Fuzzy Modeling
University of Ostrava
Ostrava, Czech Republic

IT4Innovations

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj pro inovace

Outline

How it works?

An object matching and recognition is a process of assign one, or more *labels* of *known* ones.

The labels are assigned on the basis of attributes. The attributes are available for the database of known objects and for the unknown object.

The process of connecting attributes with labels is called *learning*. The process of assigning labels to unknown objects is called *classification*.

How the attributes can be extracted?

1. The data are attributes it-selves
2. The attributes are defined by a human
3. The attributes are automatically extracted

Outline

What is it?

Let us suppose $f : D \to \mathbb{R}$, where $D \subset \mathbb{R}^r$, $r \geq 1$

Database (haystack): $\mathbf{I_{Dat}} = \{f_1, f_2, \ldots\}$

Pattern (needle): f_p, such that f_p is full coincidence, or proper inclusion of one, or several f_i from $\mathbf{I_{Dat}}$.

Current state

All cases are already solved (somehow)...

BUT

"... problems are faster methods for general convolutions, multidimensional extensions that are dimension-independent..."

A. Amir, Multidimensional pattern matching: A survey

HOWEVER

One method is universal: brute force (naive approach).

UNFORTUNATELY

Brute force is really slow.

Current state

All cases are already solved (somehow)...

BUT

"... problems are faster methods for general convolutions, multidimensional extensions that are dimension-independent..."

A. Amir, Multidimensional pattern matching: A survey

HOWEVER

One method is universal: brute force (naive approach).

UNFORTUNATELY

Brute force is really slow.

Current state

All cases are already solved (somehow)...

BUT

"... problems are faster methods for general convolutions, multidimensional extensions that are dimension-independent..."

A. Amir, Multidimensional pattern matching: A survey

HOWEVER

One method is universal: brute force (naive approach).

UNFORTUNATELY

Brute force is really slow.

Current state

All cases are already solved (somehow)...

BUT

"... problems are faster methods for general convolutions, multidimensional extensions that are dimension-independent..."

A. Amir, Multidimensional pattern matching: A survey

HOWEVER

One method is universal: brute force (naive approach).

UNFORTUNATELY

Brute force is really slow.

Outline

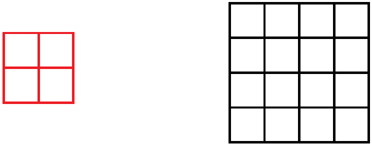
Brute force

Let us recall, $f : D \to \mathbb{R}$, where $D \subset \mathbb{R}^r$, $r \geq 1$.

We consider a database $\mathbf{I_{Dat}} = \{f_1, f_2, \ldots\}$ and a pattern f_p, such that f_p is full coincidence, or proper inclusion of one, or several f_i from $\mathbf{I_{Dat}}$.

Goal: found f_p in $f_i \in \mathbf{I_{Dat}}$ on position $\mathbf{x} = (x^1, x^2, ..., x^r)$ where $\min_{i,\mathbf{x}}(Dist(f_i, f_p, \mathbf{x}))$ is achieved.

The complexity of searching f_p in some f_i is $O(|D_i| \cdot |D_p| - |D_p|)$ for all the best, average and the worst cases.



General algorithm

Preserve complexity $O(|D_i| \cdot |D_p| - |D_p|)$.

WHY?

Apply transformation $D_p \rightarrow D'_p$ and $D_i \rightarrow D'_i$ where $|D'_p| < |D_p|$ and $|D'_i| < |D_i|$. This leads to $T(|D'_i| \cdot |D'_p| - |D'_p|) \ll T(|D_i| \cdot |D_p| - |D_p|)$, Where T represents function computes execution time.

Main idea:

- Apply the F-transform to the pattern and database.
- Obtain their reduced representations.
- Compare the objects by computing distances between components.
- Choose the corresponding database record(s).

General algorithm

Preserve complexity $O(|D_i| \cdot |D_p| - |D_p|)$.

WHY?

Apply transformation $D_p \to D'_p$ and $D_i \to D'_i$ where $|D'_p| < |D_p|$ and $|D'_i| < |D_i|$. This leads to $T(|D'_i| \cdot |D'_p| - |D'_p|) \ll T(|D_i| \cdot |D_p| - |D_p|)$, Where T represents function computes execution time.

Main idea:

- Apply the F-transform to the pattern and database.
- Obtain their reduced representations.
- Compare the objects by computing distances between components.
- Choose the corresponding database record(s).

General algorithm

Preserve complexity $O(|D_i| \cdot |D_p| - |D_p|)$.

WHY?

Apply transformation $D_p \rightarrow D'_p$ and $D_i \rightarrow D'_i$ where $|D'_p| < |D_p|$ and $|D'_i| < |D_i|$. This leads to $T(|D'_i| \cdot |D'_p| - |D'_p|) \ll T(|D_i| \cdot |D_p| - |D_p|)$, Where T represents function computes execution time.

Main idea:

- Apply the F-transform to the pattern and database.
- Obtain their reduced representations.
- Compare the objects by computing distances between components.
- Choose the corresponding database record(s).

General algorithm

Preserve complexity $O(|D_i| \cdot |D_p| - |D_p|)$.

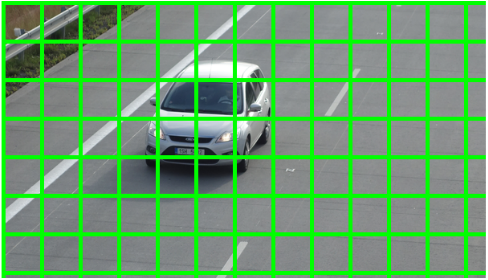
WHY?

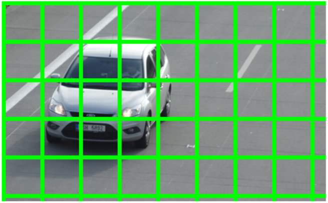
Apply transformation $D_p \to D'_p$ and $D_i \to D'_i$ where $|D'_p| < |D_p|$ and $|D'_i| < |D_i|$. This leads to $T(|D'_i| \cdot |D'_p| - |D'_p|) \ll T(|D_i| \cdot |D_p| - |D_p|)$, Where T represents function computes execution time.

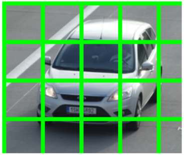
Main idea:

- Apply the F-transform to the pattern and database.
- Obtain their reduced representations.
- Compare the objects by computing distances between components.
- Choose the corresponding database record(s).

Why is it so difficult?







Preliminaries - Fuzzy transform

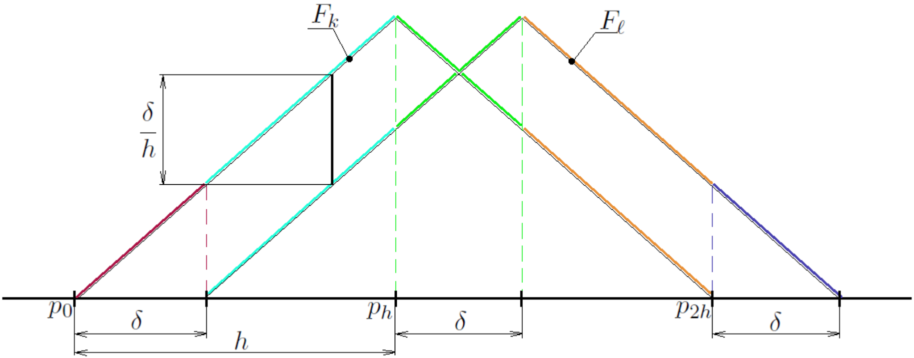
Let $\{A_1^1, \ldots, A_{n_1}^1\} \times \{A_1^2, \ldots, A_{n_2}^2\} \times \cdots \times \{A_1^r, \ldots, A_{n_r}^r\}$ be a fuzzy partition of D^r and let a function $f : D^r \to \mathbb{R}$ be known at points $(p_1^1, \ldots, p_1^r), \ldots, (p_N^1, \ldots, p_N^r)$ such that for each $(k_1, \ldots, k_r)$ where $k_j = 1, \ldots, n_j$ and $j = 1, \ldots, r$, there exists $i = 1, \ldots, N : A_{k_1}^1(p_i^1) \cdots A_{k_r}^r(p_i^r) > 0$.

We say that a ν-tuple $\mathbf{F_{n_1 n_2 \ldots n_r}}[f] = [F_{k_1 \ldots k_r}]$ of real numbers where $\nu = (n_1 \cdot n_2 \ldots n_r)$ is the discrete direct F-transform of f with respect to the given fuzzy partition if

$$F_{k_1 \ldots k_r} = \frac{\sum_{i=1}^{N} f(p_i^1, \ldots, p_i^r) A_{k_1}^1(p_i^1) \cdots A_{k_r}^r(p_i^r)}{\sum_{i=1}^{N} A_{k_1}^1(p_i^1) \cdots A_{k_r}^r(p_i^r)}$$

for each r-tuple $k_1 \ldots k_r$.

Fuzzy transform and shift



Algorithm design

Framework

Main idea:

- Apply the F-transform to the pattern and database.
- Obtain their reduced representations.
- Compare the objects by computing distances between components.
- Choose the corresponding database record(s).

- Choosing width of fuzzy partition - as big as possible (h).
- Specifying the fuzzy partition - two shifted Ruspini partitions.
- Choosing the distance - Manhattan distance.
- Choosing the threshold - computed from pattern components.

Outline

2 dimensional pattern matching algorithm

Inputs | Outputs | Pre-Processing | Processing

$I_{Dat} = \{f_1, \ldots, f_d\}$: *a database of* d *images*

f_p: *a pattern image*

$H = \{20, 40, \ldots\}$: *a set of values of a parameter* h

2 dimensional pattern matching algorithm

Inputs | **Outputs** | Pre-Processing | Processing

i: *an index of a database image*

x, y: *coordinates in the i-th database image where the pattern f_p was matched*

$Dist_i$: *a distance between the pattern and the i-th database image*

IRAFM

2 dimensional pattern matching algorithm

Inputs | Outputs | **Pre-Processing** | Processing

1. ***for each*** $h \in H$
2. ***for each*** $f_i \in I_{Dat}$, $i = 1, \ldots, d$
3. $\mathbf{F}_{\mathbf{n_{1_i} n_{2_i}}}[f_i] = [F_{k_{1_i} k_{2_i}}]_{k_{1_i}=1,\ldots,n_{1_i};\ k_{2_i}=1,\ldots,n_{2_i}}$;
   w.r.t. h*-uniform fuzzy partition of* $[1, N_{1_i}] \times [1, N_{2_i}]$
4. $\mathbf{F}_{\mathbf{n_{1_i'} n_{2_i'}}}[f_i] = [F_{k_{1_i} k_{2_i}}]_{k_{1_i}=1,\ldots,n_{1_i}-1;\ k_{2_i}=1,\ldots,n_{2_i}-1}$;
   w.r.t. h*-uniform fuzzy partition of* $[h/2, N_{1_i}] \times [h/2, N_{2_i}]$

2 dimensional pattern matching algorithm

Inputs | Outputs | Pre-Processing | **Processing**

1. $h_p = \sqrt{N_{1_p} N_{2_p} / 15}$
2. *choose* $h_j \in H$ *such that* $|h_j - h_p| = \min_{h \in H} |h - h_p|$
3. $\mathbf{F_{n_{1_p} n_{2_p}}}[f_p] = [F_{k_{1_p} k_{2_p}}]_{k_{1_p}=1,\dots,n_{1_p};\ k_{2_p}=1,\dots,n_{2_p}}$;
   w.r.t. h*-uniform fuzzy partition of* $[1, N_{1_i}] \times [1, N_{2_i}]$
4. $\theta = \frac{\sum_{s=1}^{n_{1_p}-1} \sum_{t=1}^{n_{2_p}-1} |F_{s_p,t_p} - F_{s+1_p,t_p}| + |F_{s_p,t_p} - F_{s_p,t+1_p}|}{2(n_{1_p}-1)(n_{2_p}-1)}$
5. ***for each*** $i = 1, \dots, d$
6. ***for each*** $x = 1, \dots, n_{1_i} - n_{1_p}$
7. ***for each*** $y = 1, \dots, n_{2_i} - n_{2_p}$

2 dimensional pattern matching algorithm

Inputs | Outputs | Pre-Processing | **Processing**

8. $T_i^{xy} \subset \mathbf{F}_{\mathbf{n_{1_i} n_{2_i}}}[f_i]$ *such that*
$T_i^{xy} = [F_{k_{1_i} k_{2_i}}]_{k_{1_i}=x,\ldots,x+n_{1_p}-1;\ k_{2_i}=y,\ldots,y+n_{2_p}-1}$
9. $T_{i,h/2}^{xy} \subset \mathbf{F}_{\mathbf{n_{1'_i} n_{2'_i}}}[f_i]$ *such that*
$T_{i,h/2}^{xy} = [F_{k_{1_i} k_{2_i}}]_{k_{1_i}=x,\ldots,x+n_{1_p}-1;\ k_{2_i}=y,\ldots,y+n_{2_p}-1}$
10. $Dist_i(\mathbf{F}_{\mathbf{n_{1_p} n_{2_p}}}[f_p], T_i^{xy}) = \sum_{k_{1_i}=1}^{n_{1_p}} \sum_{k_{2_i}=1}^{n_{2_p}} |F_{k_{1_p} k_{2_p}} - F_{k_{1_i} k_{2_i}}|$
11. $Dist_{i,h/2}(\mathbf{F}_{\mathbf{n_{1_p} n_{2_p}}}[f_p], T_{i,h/2}^{xy}) = \sum_{k_{1_i}=1}^{n_{1_p}} \sum_{k_{2_i}=1}^{n_{2_p}} |F_{k_{1_p} k_{2_p}} - F_{k_{1_i} k_{2_i}}|$
12. ***if*** $Dist_i + Dist_{i,h/2} < \theta$
then *store* $< i, x \cdot h, y \cdot h, Dist_i + Dist_{i,h/2} >$
13. ***end****;*
14. ***out:*** *stored triplet with the lowest* $Dist$

What is it good for?

Applications where we used the matching algorithm:

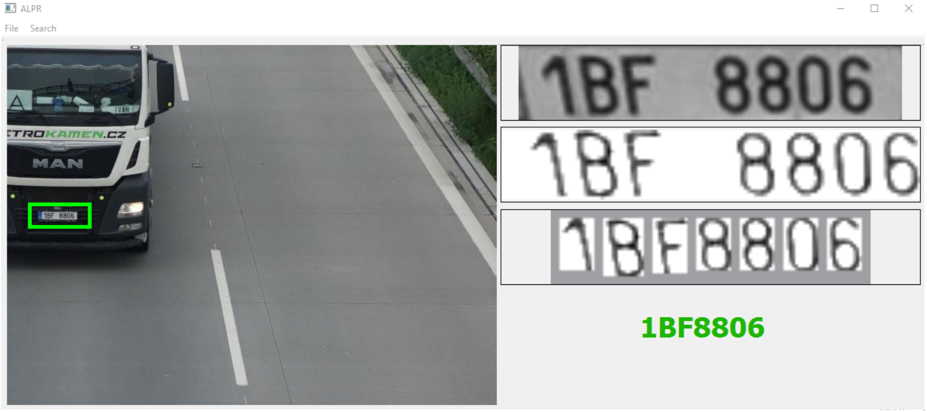
- network attack detection,
- ALPR,
- sound recognition,
- searching in large databases,
- ...

Outline

ALPR

Two main steps:

1. car plate localization,
2. car plate recognition.



Car plate searching

Define database

Car plate searching

For the defined database, gradient magnitude image is created. Components for those images are computed. These components are then matched with the same-style created components of an input image.

```
sims size: 12
9.98235    130_8_-9
9.99127    130_8_-8
10.0024    130_9_-11
10.1698    130_8_-10
10.4399    130_9_-10
10.8484    130_8_-11
11.1698    130_8_-7
11.2212    130_9_-9
11.6609    130_8_-6
11.8735    130_9_-8
12.2563    130_9_-7
12.5493    130_9_-6
```

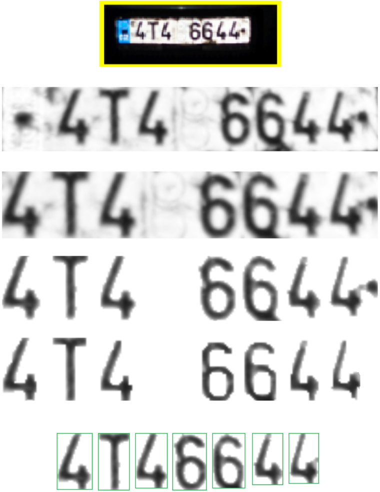
Additional informations can be extracted:

- rotation,
- scale.

These informations are used in further pre-processing.

Car plate searching

Car plate pre-processing



Pre-processing:

- Borders detection
- Background suppression
- De-skew
- Adaptive threshold with hysteresis
- Letters extraction

Letter recognition

The same process as in the case of plates:

1. Put letters to database
2. Compute components
3. Take unknown letter and search the closest one

Moreover, the recognized letters are added to the database automatically.

The final letter database consists of 28000 labelled letters.

Performance

Nr. comparisons of letters per second performed, single core computing:

- 1.03M/s on notebook i5 processor
- 1.65M/s in desktop i7 processor

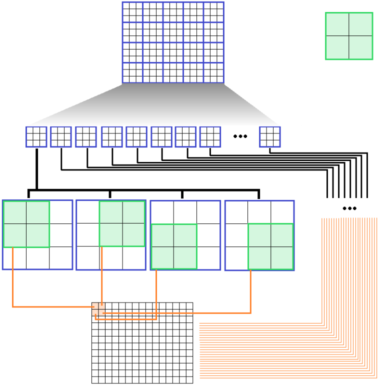
Complete images (2048×1088px) per second recognized, single core:

- 1.27/s on notebook i5 processor
- 2.08/s on desktop i7 processor

Real images, all conditions, whole European plates, $x \cdot 10^5$ plates

- approx 90 % success rate for plate localization
- approx 90 % success rate for plate recognition

Actual development



Implementation on GPU using OpenCl. Achieved $3\times$ speed-up on integrated Intel HD4000 graphic.

Conclusion

The general pattern matching algorithm was demonstrated.

The algorithm is as easy as possible:

1. put database images into folder,
2. run components computation,
3. take set of patterns,
4. classify!

Advantages:

- simplicity
- controlled precision / speed
- independent on task
- interpretability

Acknowledgement

Follow us on facebook.com/fuzzyOstrava

IRAFM