Prof. Hemerson Pistori
Dom Bosco Catholic University
*Campo Grande - Brazil*

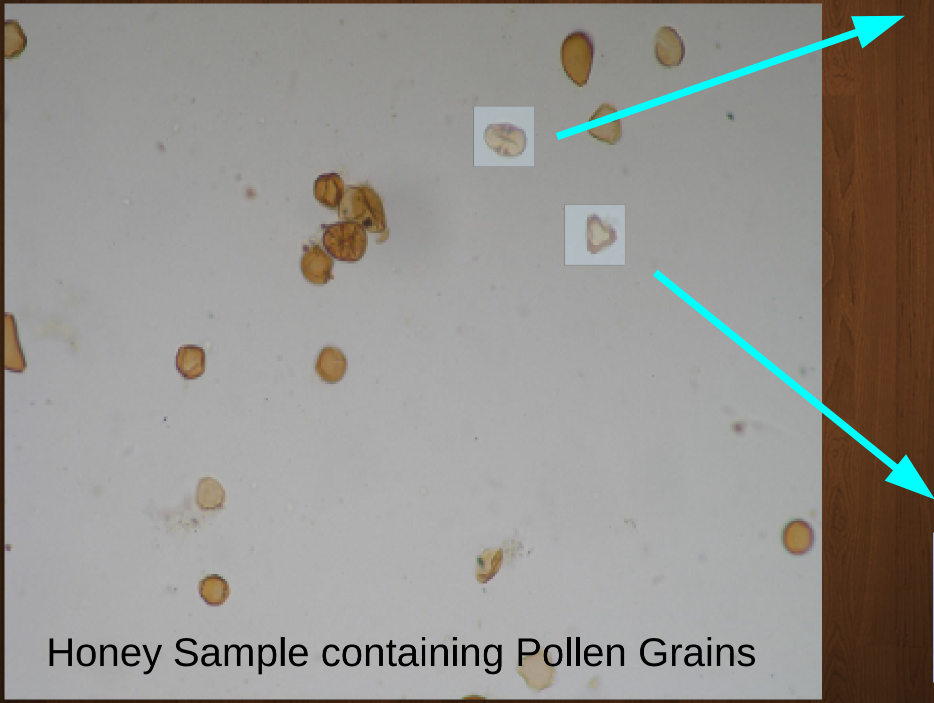
# A New Strategy for Applying Grammatical Inference to Image Classification Problems

H. Pistori, A. Calway and P. Flach

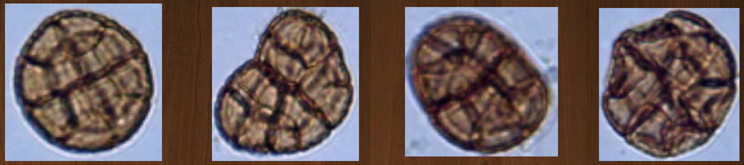
# Outline

- Background
- Related Work Review
- Proposed Approach
- Experiments
- Results
- Conclusion and Future Works

# Pattern Recognition

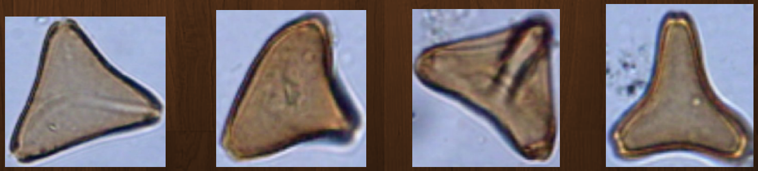


Honey Sample containing Pollen Grains



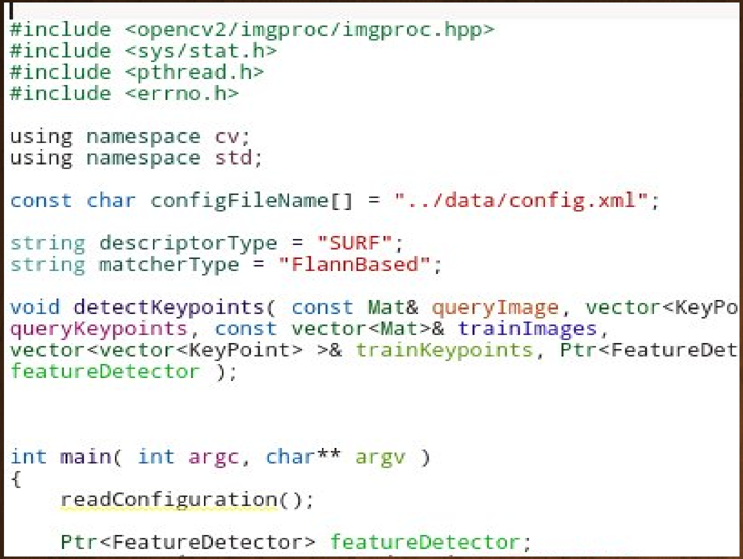
Pollen Grains from the Fabaceae Species
Training instances from class A

Supervised Machine Learning

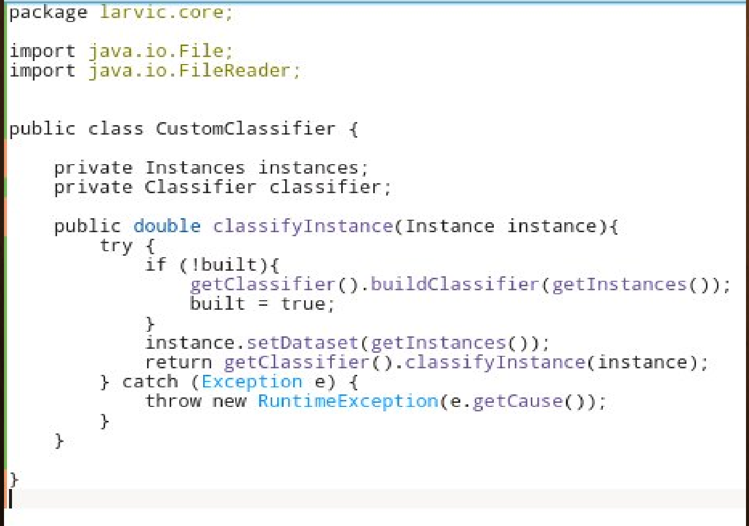


Pollen Grains from the Serjania Species
Training instances from class B

# Syntactic Pattern Recognition

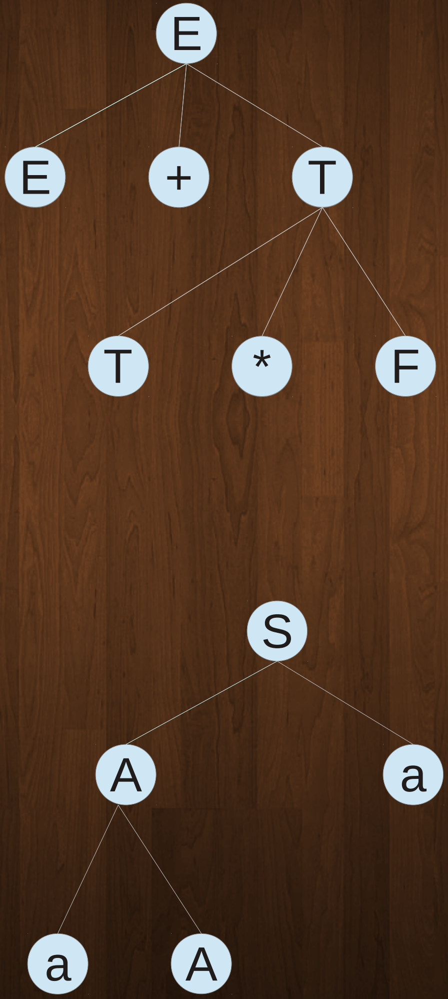


```
#include <opencv2/imgproc/imgproc.hpp>
#include <sys/stat.h>
#include <pthread.h>
#include <errno.h>

using namespace cv;
using namespace std;

const char configFileName[] = "../data/config.xml";

string descriptorType = "SURF";
string matcherType = "FlannBased";

void detectKeypoints( const Mat& queryImage, vector<KeyPo
queryKeypoints, const vector<Mat>& trainImages,
vector<vector<KeyPoint> >& trainKeypoints, Ptr<FeatureDet
featureDetector );



int main( int argc, char** argv )
{
    readConfiguration();

    Ptr<FeatureDetector> featureDetector;
```

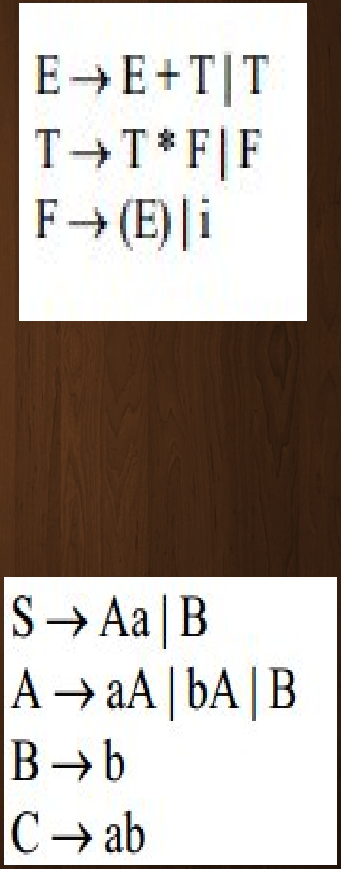


```
package larvic.core;

import java.io.File;
import java.io.FileReader;


public class CustomClassifier {

    private Instances instances;
    private Classifier classifier;

    public double classifyInstance(Instance instance){
        try {
            if (!built){
                getClassifier().buildClassifier(getInstances());
                built = true;
            }
            instance.setDataset(getInstances());
            return getClassifier().classifyInstance(instance);
        } catch (Exception e) {
            throw new RuntimeException(e.getCause());
        }
    }

}
```

Alphabetic Symbols and Strings



Automata, Parsers, Derivation Trees (Syntax Analysis)



$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid i$

$S \rightarrow Aa \mid B$

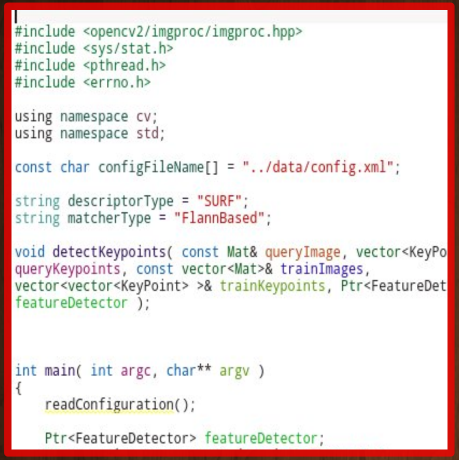
$A \rightarrow aA \mid bA \mid B$

$B \rightarrow b$

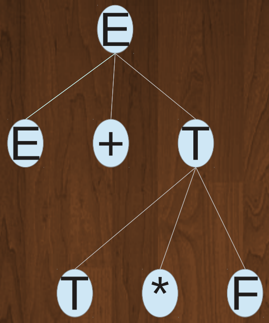
$C \rightarrow ab$

Grammars

# Syntactic Pattern Recognition



```
#include <opencv2/imgproc/imgproc.hpp>
#include <sys/stat.h>
#include <pthread.h>
#include <errno.h>

using namespace cv;
using namespace std;

const char configFileName[] = "../data/config.xml";

string descriptorType = "SURF";
string matcherType = "FlannBased";

void detectKeypoints( const Mat& queryImage, vector<KeyPo
queryKeypoints, const vector<Mat>& trainImages,
vector<vector<KeyPoint> >& trainKeypoints, Ptr<FeatureDet
featureDetector );


int main( int argc, char** argv )
{
    readConfiguration();

    Ptr<FeatureDetector> featureDetector;
```

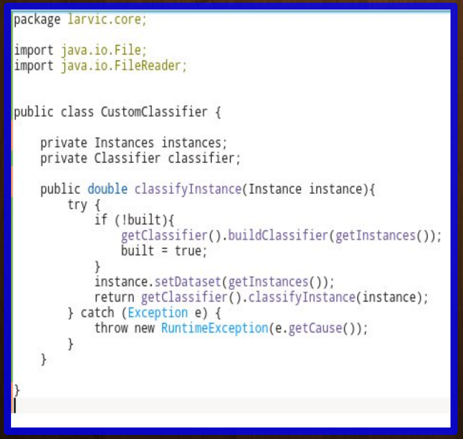


$E \to E+T \mid T$

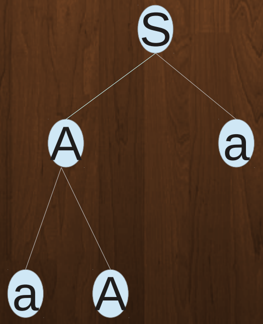
$T \to T*F \mid F$

$F \to (E) \mid i$

**Grammatical Inference**



```
package larvic.core;

import java.io.File;
import java.io.FileReader;


public class CustomClassifier {

    private Instances instances;
    private Classifier classifier;

    public double classifyInstance(Instance instance){
        try {
            if (!built){
                getClassifier().buildClassifier(getInstances());
                built = true;
            }
            instance.setDataset(getInstances());
            return getClassifier().classifyInstance(instance);
        } catch (Exception e) {
            throw new RuntimeException(e.getCause());
        }
    }

}
```

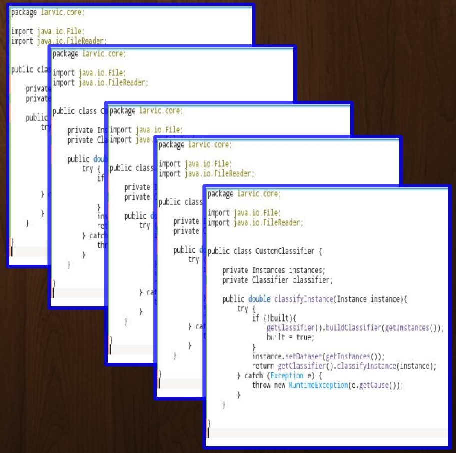


$S \to Aa \mid B$

$A \to aA \mid bA \mid B$

$B \to b$

$C \to ab$

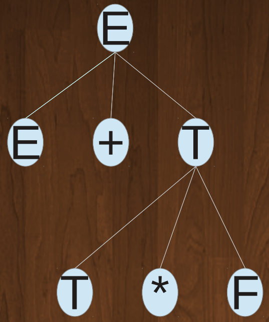


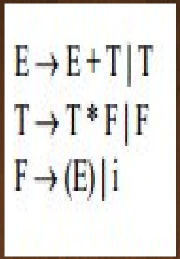
Alphabetic Symbols and Strings

Syntax Analysis (Automata, Parsers, Derivation Trees, etc)

Grammars

# Syntactic Pattern Recognition



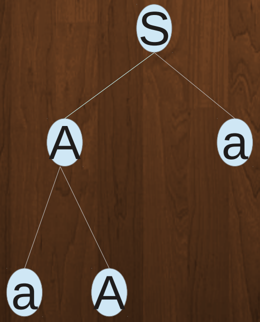


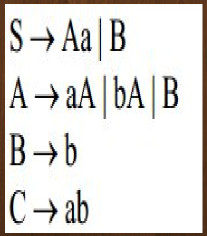
$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to (E) \mid i$

**Grammatical Inference**



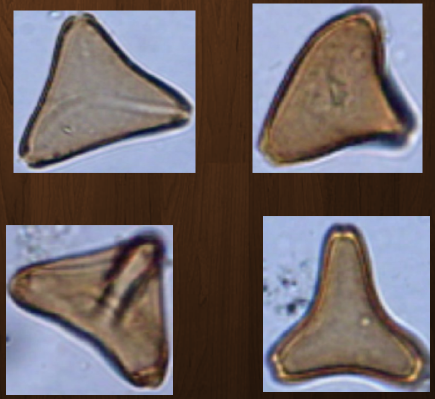


$S \to Aa \mid B$

$A \to aA \mid bA \mid B$

$B \to b$

$C \to ab$



Images

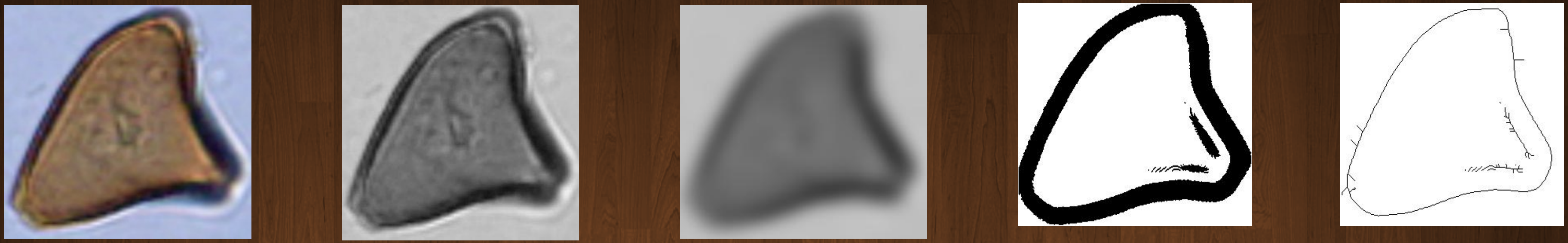
Syntax Analysis (Automata, Parsers, Derivation Trees, etc)

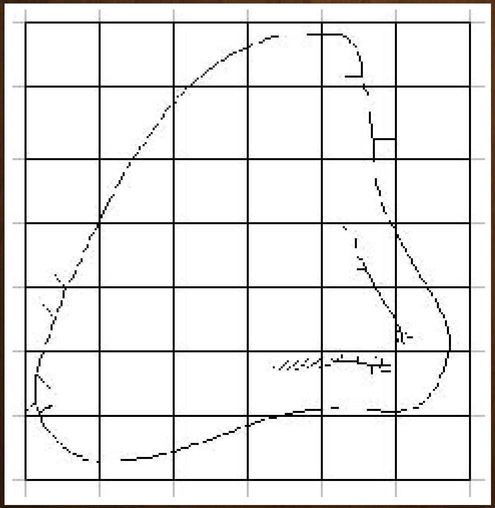
Grammars

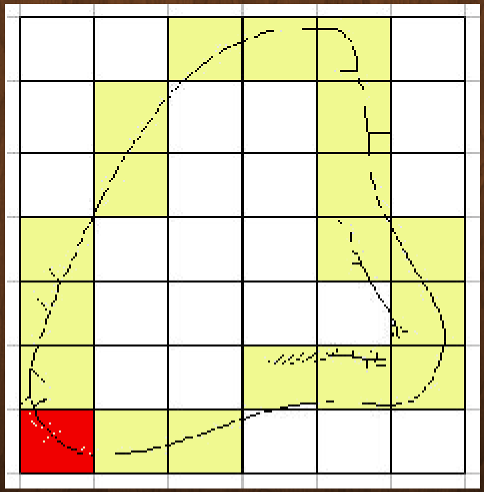
Where is our Alphabet ?

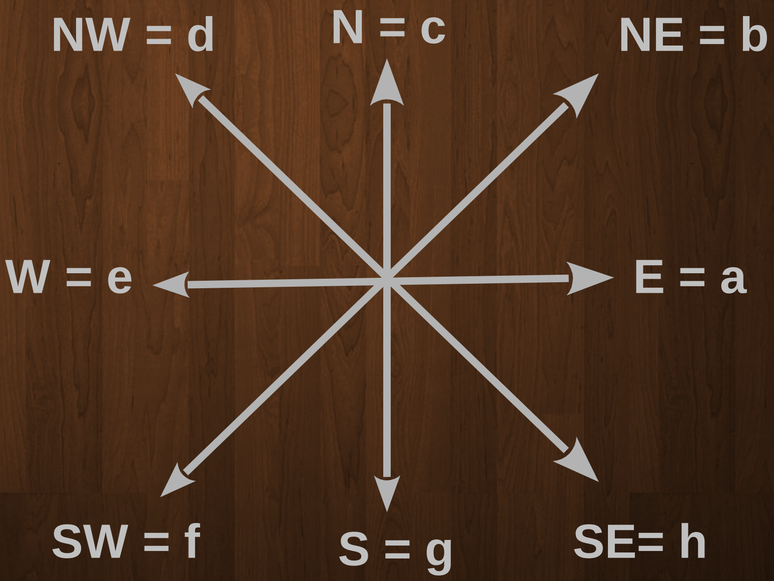
Central trade-off question: Should we somehow convert images to strings or replace the string for something else  and create other types of grammars and syntax analysers?

# Syntactic Pattern Recognition









NW = d

N = c

NE = b

W = e

E = a

SW = f

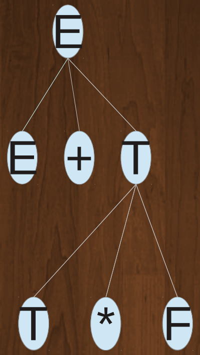
S = g

SE= h

A string for this Serjania pollen sample:  c c c b c b a a g g g a g g e e f e e

# Syntactic Pattern Recognition

caaccbd



$E \rightarrow E+T \mid T$

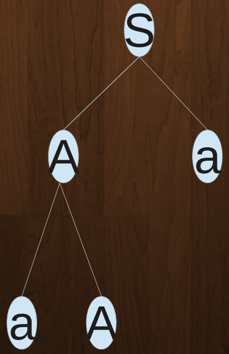
$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid i$

caaccbd
ccaacbeef
aaacccbb
...

**Grammatical Inference**

aafffbgg



$S \rightarrow Aa \mid B$

$A \rightarrow aA \mid bA \mid B$

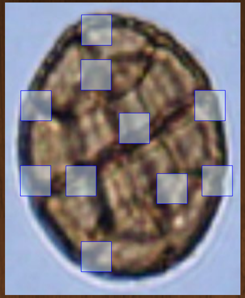
$B \rightarrow b$

$C \rightarrow ab$

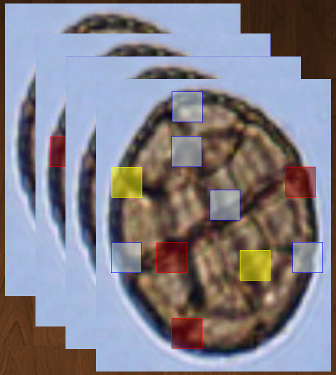
aafffbggc
aafffddddf
bbbaaccc
...

# Our Proposal

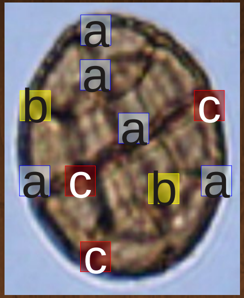
Original



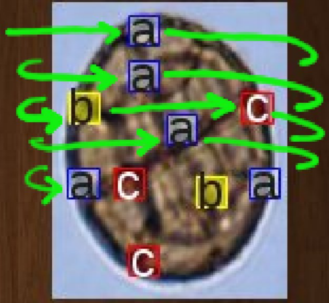
Keypoint detection



Bag of Words

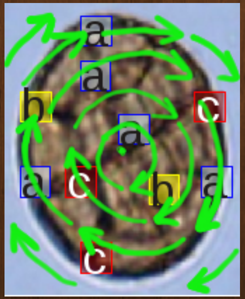


Symbols Mapping

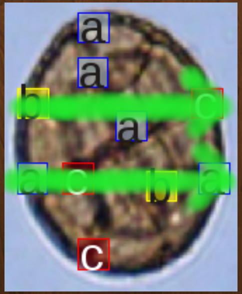


Scan: aabcaacbac

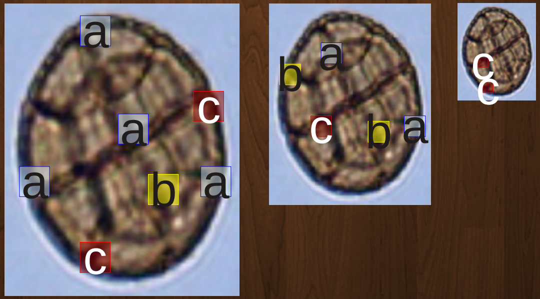
Alternative Scan Order:



Radial



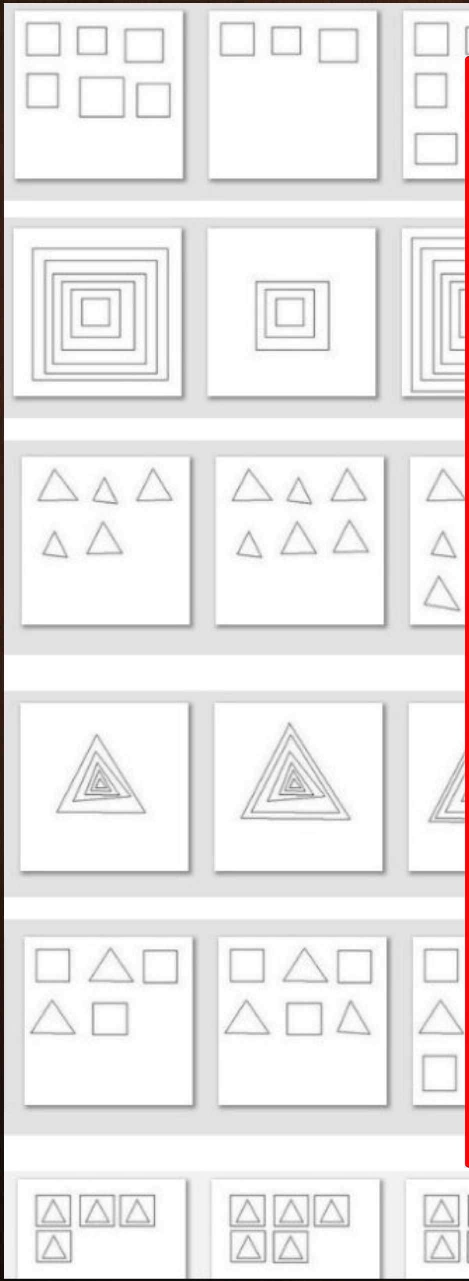
Coarse Reading

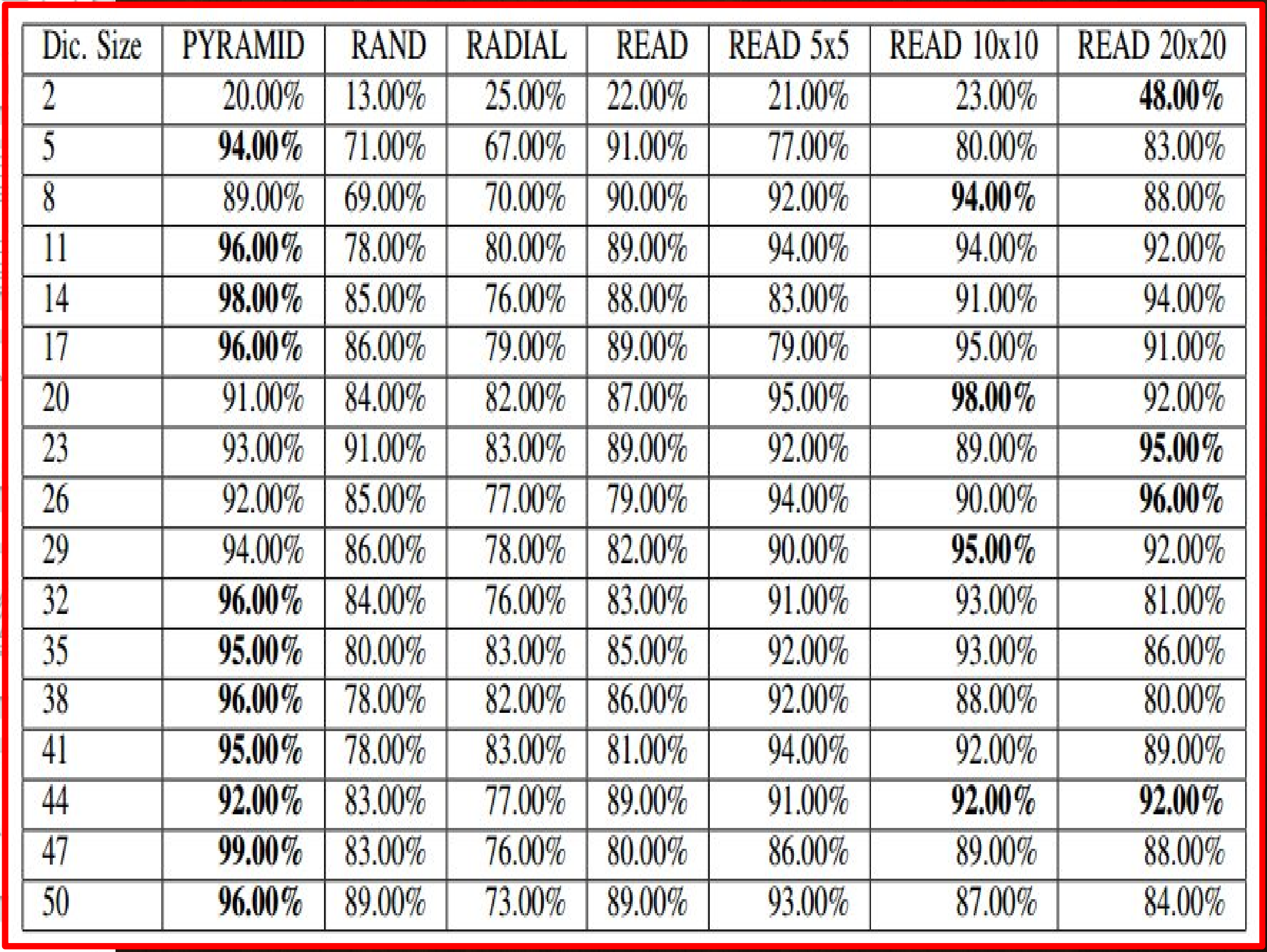


SURF PYRAMID

Experiments: Surf Keypoint Detector, K-Testable GI, Ignore Token Error Recovery
Scan order: Random, Reading, Radial, Pyramid, Coarse Reading (5p, 10p, 20p)

# Experiments and Results – Squares and Triangles Relations





| Dic. Size | PYRAMID | RAND | RADIAL | READ | READ 5x5 | READ 10x10 | READ 20x20 |
|---|---|---|---|---|---|---|---|
| 2 | 20.00% | 13.00% | 25.00% | 22.00% | 21.00% | 23.00% | **48.00%** |
| 5 | **94.00%** | 71.00% | 67.00% | 91.00% | 77.00% | 80.00% | 83.00% |
| 8 | 89.00% | 69.00% | 70.00% | 90.00% | 92.00% | **94.00%** | 88.00% |
| 11 | **96.00%** | 78.00% | 80.00% | 89.00% | 94.00% | 94.00% | 92.00% |
| 14 | **98.00%** | 85.00% | 76.00% | 88.00% | 83.00% | 91.00% | 94.00% |
| 17 | **96.00%** | 86.00% | 79.00% | 89.00% | 79.00% | 95.00% | 91.00% |
| 20 | 91.00% | 84.00% | 82.00% | 87.00% | 95.00% | **98.00%** | 92.00% |
| 23 | 93.00% | 91.00% | 83.00% | 89.00% | 92.00% | 89.00% | **95.00%** |
| 26 | 92.00% | 85.00% | 77.00% | 79.00% | 94.00% | 90.00% | **96.00%** |
| 29 | 94.00% | 86.00% | 78.00% | 82.00% | 90.00% | **95.00%** | 92.00% |
| 32 | **96.00%** | 84.00% | 76.00% | 83.00% | 91.00% | 93.00% | 81.00% |
| 35 | **95.00%** | 80.00% | 83.00% | 85.00% | 92.00% | 93.00% | 86.00% |
| 38 | **96.00%** | 78.00% | 82.00% | 86.00% | 92.00% | 88.00% | 80.00% |
| 41 | **95.00%** | 78.00% | 83.00% | 81.00% | 94.00% | 92.00% | 89.00% |
| 44 | **92.00%** | 83.00% | 77.00% | 89.00% | 91.00% | **92.00%** | **92.00%** |
| 47 | **99.00%** | 83.00% | 76.00% | 80.00% | 86.00% | 89.00% | 88.00% |
| 50 | **96.00%** | 89.00% | 73.00% | 89.00% | 93.00% | 87.00% | 84.00% |

F-Measures – 6 Classes

# Experiments and Results – 15 Scenes

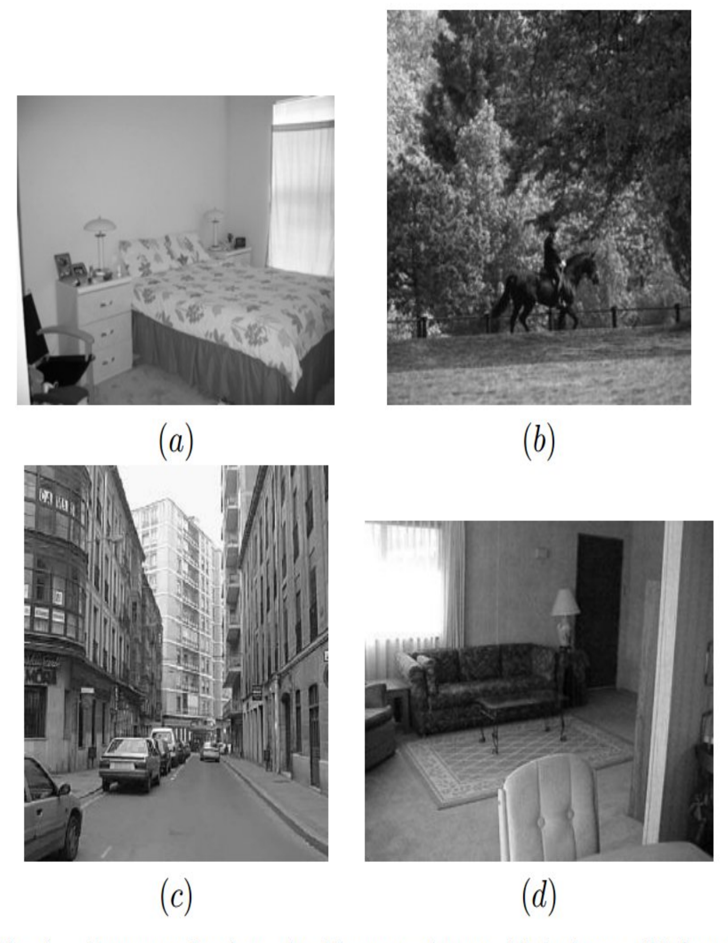
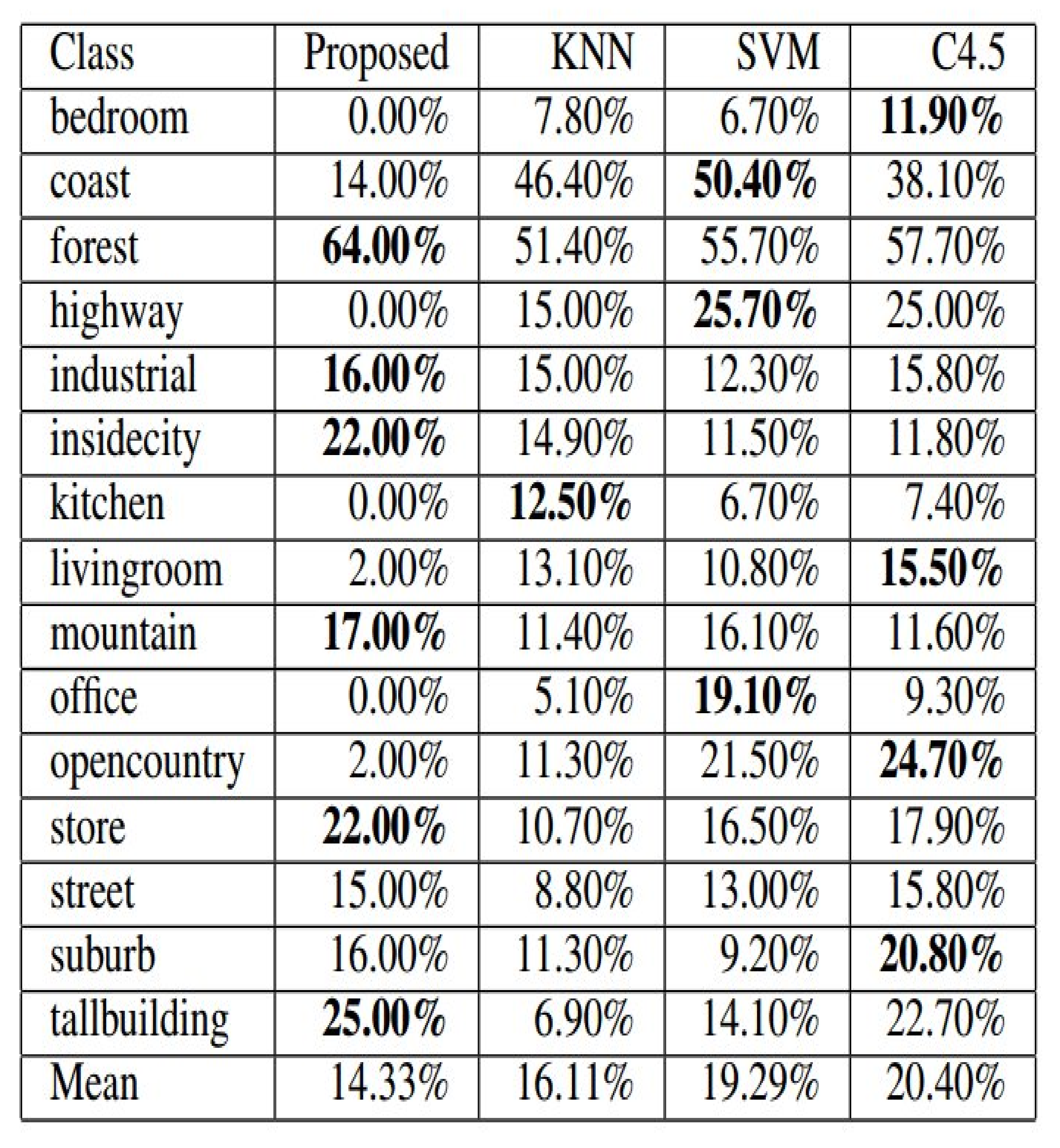


Fig. 4. Four samples from the 15 scenes dataset: (a) bedroom, (b) forest, (c) street and (d) living room



| Class | Proposed | KNN | SVM | C4.5 |
|---|---|---|---|---|
| bedroom | 0.00% | 7.80% | 6.70% | **11.90%** |
| coast | 14.00% | 46.40% | **50.40%** | 38.10% |
| forest | **64.00%** | 51.40% | 55.70% | 57.70% |
| highway | 0.00% | 15.00% | **25.70%** | 25.00% |
| industrial | **16.00%** | 15.00% | 12.30% | 15.80% |
| insidecity | **22.00%** | 14.90% | 11.50% | 11.80% |
| kitchen | 0.00% | **12.50%** | 6.70% | 7.40% |
| livingroom | 2.00% | 13.10% | 10.80% | **15.50%** |
| mountain | **17.00%** | 11.40% | 16.10% | 11.60% |
| office | 0.00% | 5.10% | **19.10%** | 9.30% |
| opencountry | 2.00% | 11.30% | 21.50% | **24.70%** |
| store | **22.00%** | 10.70% | 16.50% | 17.90% |
| street | 15.00% | 8.80% | 13.00% | 15.80% |
| suburb | 16.00% | 11.30% | 9.20% | **20.80%** |
| tallbuilding | **25.00%** | 6.90% | 14.10% | 22.70% |
| Mean | 14.33% | 16.11% | 19.29% | 20.40% |

F-Measures – Dic. Size = 50

# Conclusions

- New approach to generate strings from images
- Encouraging first results
- Preserves spatial and structured information (unlike BOW)

# Future Works

- Other grammar inference approaches and error recovery strategies during parsing
- Negative samples
- Stochastic grammars
- Code optimizations
- Experiments using other datasets