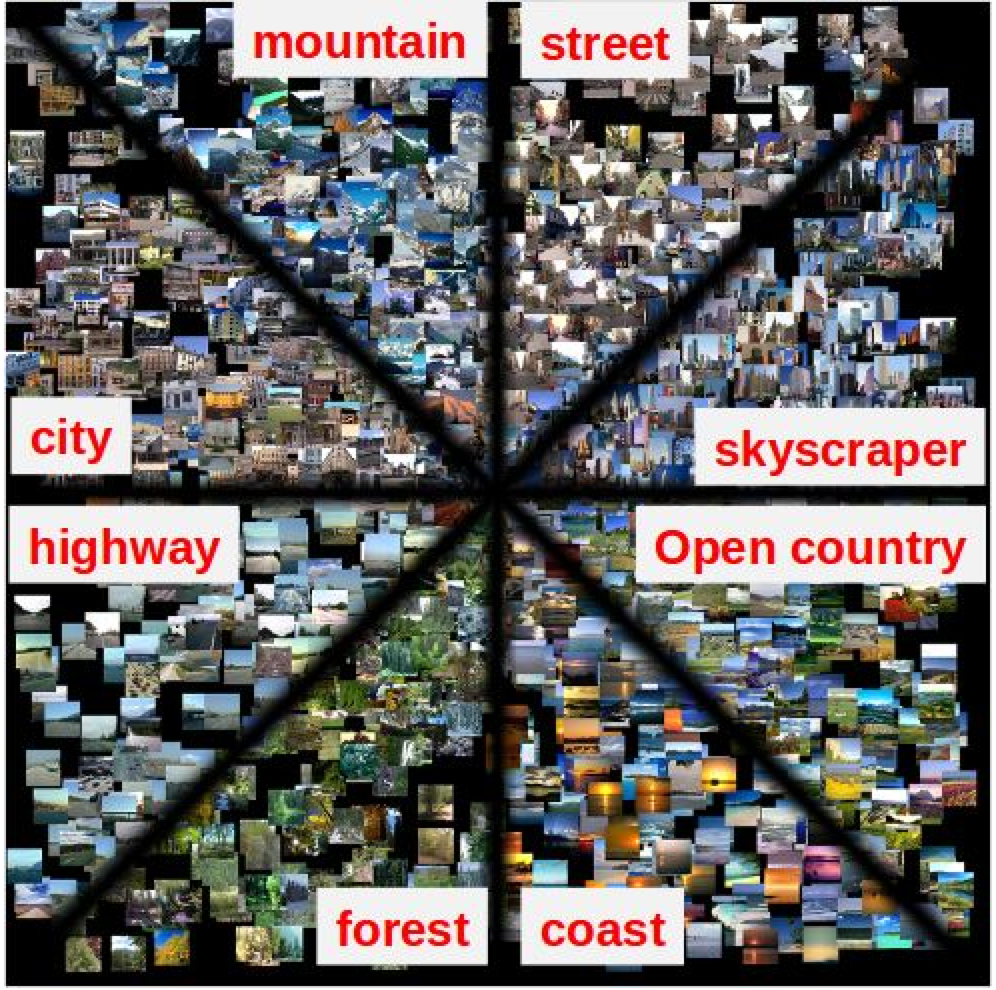
Master in Computer Vision Barcelona

Image Classification

Marçal Rossinyol - marcal@cvc.uab.cat
Ramon Baldrich - ramon@cvc.uab.cat


mountain
street
city
skyscraper
highway
Open country
forest
coast

The dataset

8 classes

Coast
244 train
116 test

Forest
227 train
101 test

Highway
184 train
76 test

Inside city
214 train
94 test

Mountain
260 train
114 test

Open country
292 train
118 test

Street
212 train
80 test

Tall building
248 train
108 test

Artificial Intellingence in Computer Vision

Machine learning for image classification:

- Handcrafted methods: Bag of Words
- Data driven methods: Deep Convolutional Networks

Artificial Intellingence in Computer Vision

Machine learning for image classification:

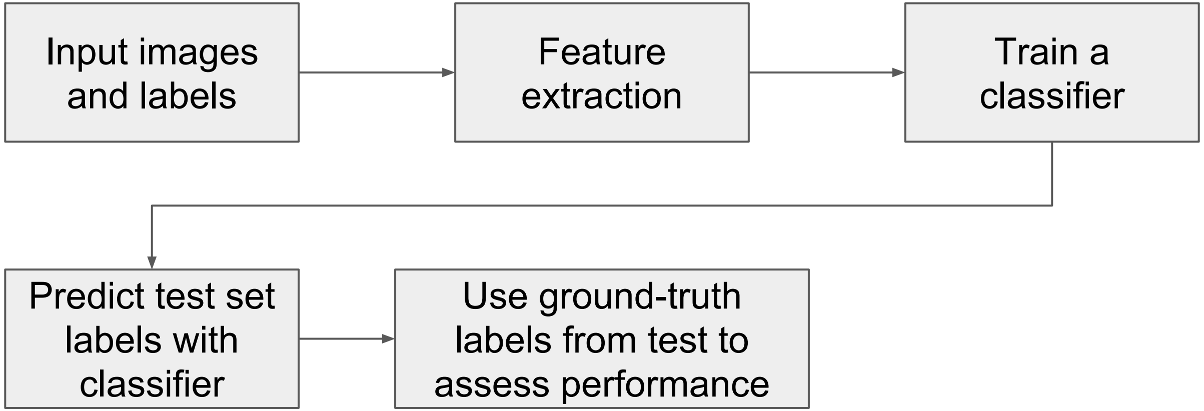
- Handcrafted methods: Bag of Words:
  - Basic image classification with local features
  - The Bag of Visual Words framework and beyond: Spatial Pyramids, Fisher Vectors

Artificial Intellingence in Computer Vision

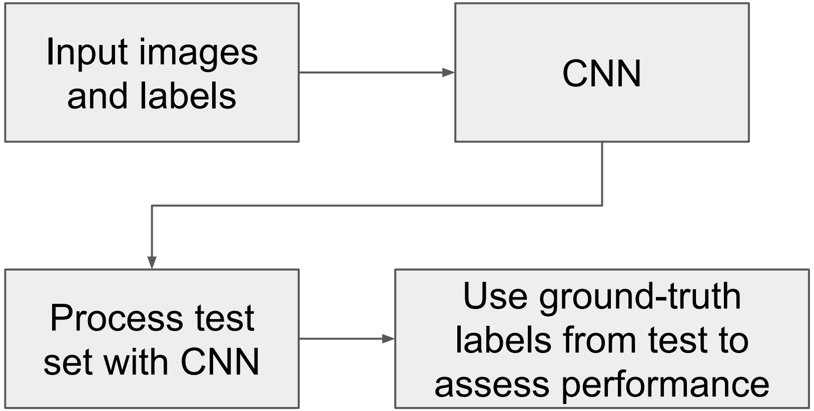
Machine learning for image classification:

- Data driven methods: Deep Convolutional Networks: 3 sessions
  - From hand-crafted to learnt features
  - Fine tuning of pre-trained CNNs
  - Training a CNN from scratch

Pipeline of the project



Pipeline of the project



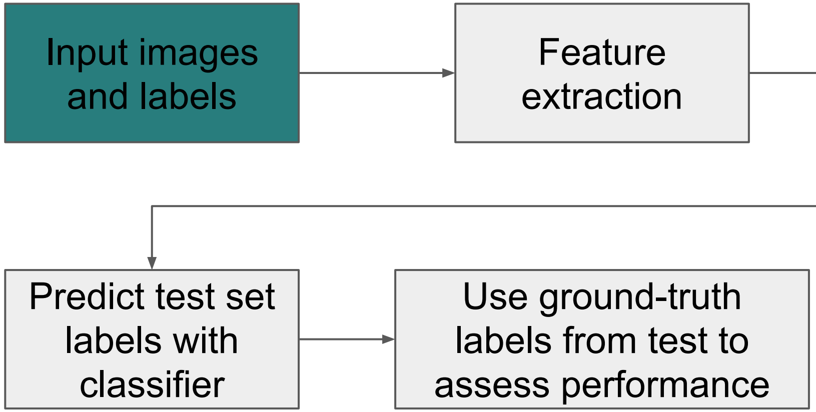
Libraries

- For the first sessions we will use:
  - Python + common libraries (tested on 2.7.6)
  - OpenCV (tested on 2.4.11)
  - Numpy (tested on 1.11.2)
  - sklearn (tested on 0.16.1)
- All on Linux, Windows & Mac at your own risk

1st session: Basic Image classification

- In this first session we will start with a naïve classification architecture based on the *SIFT local features* and *K-nearest neighbor* classification
- SIFT being a **local** descriptor, we will have ***m*** 128-dimensional descriptors per image
- We will train an k-NN that will predict from which class each local descriptor comes from (is that a good idea?)
- Given a test image, we will extract SIFT features, and predict their labels
- We will aggregate all those predictions in order to have a single answer per image
- We will evaluate the final performance of the system.

Pipeline of the project



Reading the images

- We provide two folders (train and test) with 8 subfolders containing the images from the same class
- Python lists with all the paths and info are also available and preferable

```python
import cv2
import numpy as np
import cPickle
import time
from sklearn.neighbors import KNeighborsClassifier

start = time.time()

# read the train and test files

train_images_filenames = cPickle.load(open('train_images_filenames.dat','r'))
test_images_filenames = cPickle.load(open('test_images_filenames.dat','r'))
train_labels = cPickle.load(open('train_labels.dat','r'))
test_labels = cPickle.load(open('test_labels.dat','r'))

print 'Loaded '+str(len(train_images_filenames))+' training images filenames with classes ',set(train_labels)
print 'Loaded '+str(len(test_images_filenames))+' testing images filenames with classes ',set(test_labels)
```

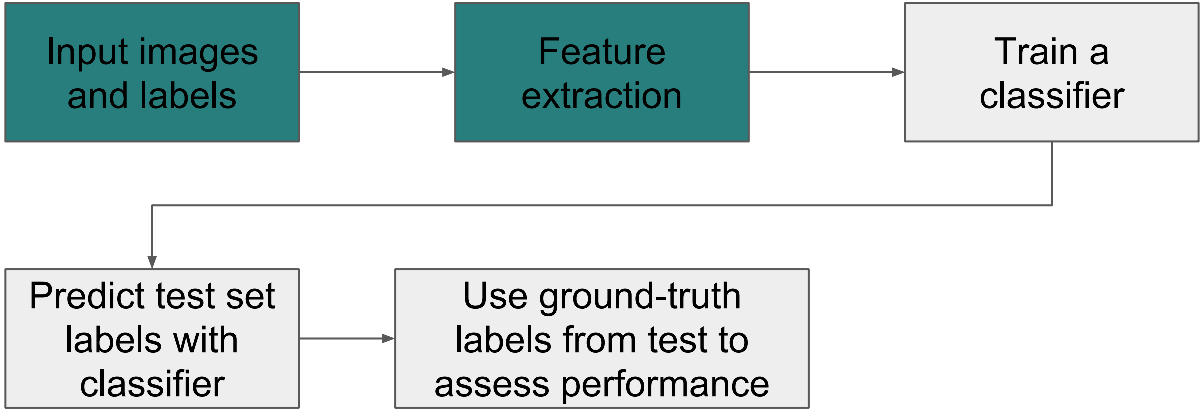
Reading the images

```
train_images_filenames

['../../Databases/MIT_split/train/Opencountry/art582.jpg',../../Databases/MIT_split/train/Opencountry/cdmc276.jpg',...

train_labels

['Opencountry', 'Opencountry', 'Opencountry',...

test_images_filenames

['../../Databases/MIT_split/test/Opencountry/cdmc109.jpg', '../../Databases/MIT_split/test/Opencountry/cdmc518.jpg',...

test_labels

['Opencountry', 'Opencountry', 'Opencountry',...
```

So, to read an image and have its label

```
ima1 = cv2.imread(train_images_filenames[0])

label1 = train_labels[0]
```

Pipeline of the project



Feature extraction

Extract SIFT features from an image

```
SIFTdetector = cv2.SIFT(nfeatures=100)

ima1 = cv2.imread(train_images_filenames[0])

gray1 = cv2.cvtColor(ima1,cv2.COLOR_BGR2GRAY)

keypoints , descriptors = SIFTdetector.detectAndCompute(gray1,None)
```

Feature extraction

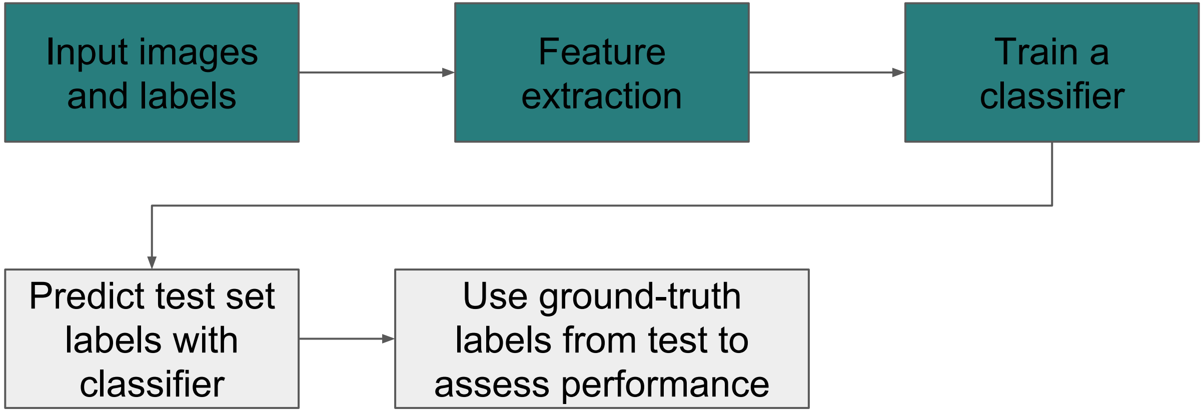
But we want:

- To have in a single array all the descriptors for the train set
  - Since this might be huge, we will start with just a few train images per class...
- To have in another list all the associated labels (not at image level, but at keypoint / descriptor level)
- Numpy arrays are preferable for later integration with the classifiers from sklearn.

Feature extraction

```python
# create the SIFT detector object

SIFTdetector = cv2.SIFT(nfeatures=100)

# read the just 30 train images per class
# extract SIFT keypoints and descriptors
# store descriptors in a python list of numpy arrays

Train_descriptors = []
Train_label_per_descriptor = []

for i in range(len(train_images_filenames)):
    filename=train_images_filenames[i]
    if Train_label_per_descriptor.count(train_labels[i])<30:
        print 'Reading image '+filename
        ima=cv2.imread(filename)
        gray=cv2.cvtColor(ima,cv2.COLOR_BGR2GRAY)
        kpt,des=SIFTdetector.detectAndCompute(gray,None)
        Train_descriptors.append(des)
        Train_label_per_descriptor.append(train_labels[i])
        print str(len(kpt))+' extracted keypoints and descriptors'

# Transform everything to numpy arrays

D=Train_descriptors[0]
L=np.array([Train_label_per_descriptor[0]]*Train_descriptors[0].shape[0])

for i in range(1,len(Train_descriptors)):
    D=np.vstack((D,Train_descriptors[i]))
    L=np.hstack((L,np.array([Train_label_per_descriptor[i]]*Train_descriptors[i].shape[0])))
```

Pipeline of the project



k-NN classifier

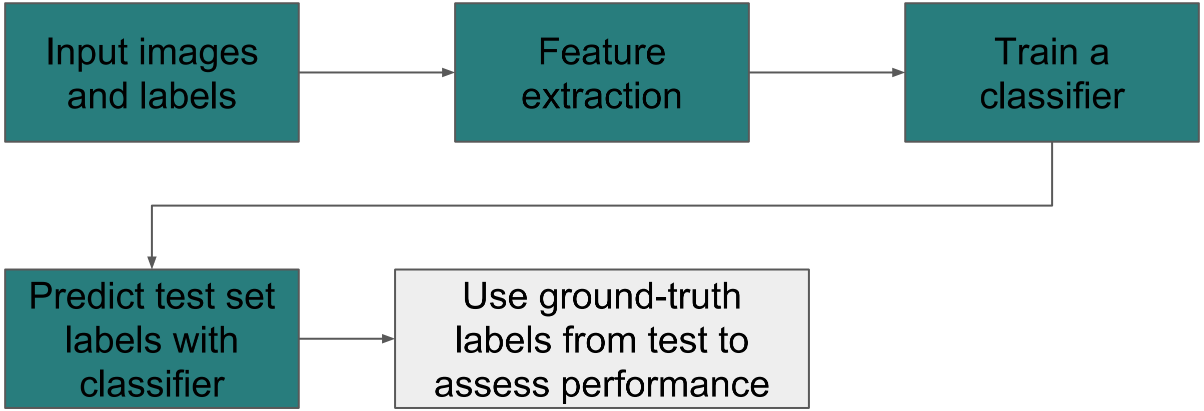
Train a k-NN classifier

```
myknn = KNeighborsClassifier(n_neighbors=5,n_jobs=-1) # k=5 and use all the available CPUs on the machine

myknn.fit(data,labels)
```

k-NN classifier

```python
# Train a k-nn classifier

print 'Training the knn classifier...'
myknn = KNeighborsClassifier(n_neighbors=5,n_jobs=-1)
myknn.fit(D,L)
print 'Done!'
```

Pipeline of the project



Testing k-NN classifier

Test a k-NN classifier

```
predictions = myknn.predict(test_data) # this returns an mx1 numpy array with m the length of test_data
```

Testing k-NN classifier

```
ima=cv2.imread(test_images_filenames[0])
gray=cv2.cvtColor(ima,cv2.COLOR_BGR2GRAY)
kpt,des=SIFTdetector.detectAndCompute(gray,None)
predictions = myknn.predict(des) # mx1 predictions
```

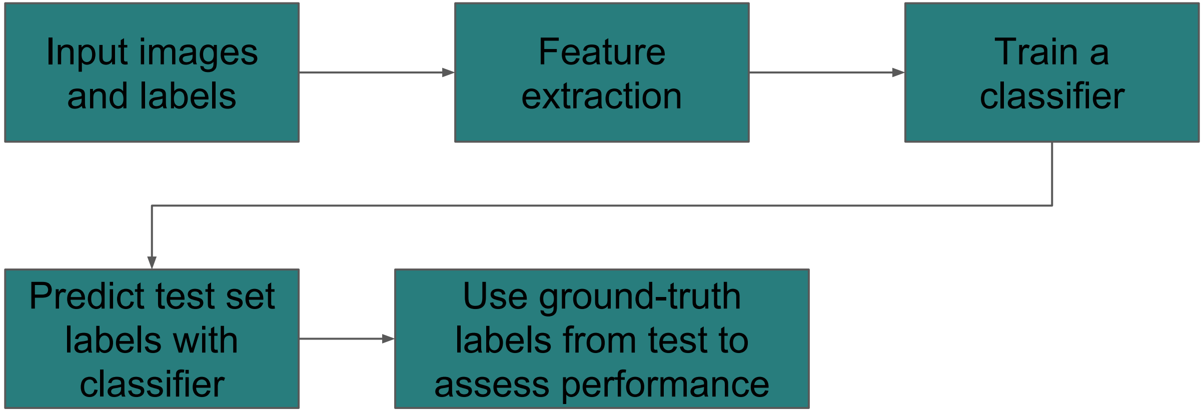
Testing k-NN classifier

```
ima=cv2.imread(test_images_filenames[0])
gray=cv2.cvtColor(ima,cv2.COLOR_BGR2GRAY)
kpt,des=SIFTdetector.detectAndCompute(gray,None)
predictions = myknn.predict(des) # mx1 predictions
```

Now we need to aggregate them all into a single image classification

```
values, counts = np.unique(predictions, return_counts=True)
predictedclass = values[np.argmax(counts)]
```

Pipeline of the project



Performance Evaluation

- We will focus on the ***accuracy***
  - i.e. which is the percentage of test images that we correctly predicted their class
- sklearn has already implemented many performance evaluation measures, but we will make our own, for latter integration with the rest of the project
- Let's put all together

Test with performance evaluation

```
# get all the test data and predict their labels

numtestimages=0
numcorrect=0
for i in range(len(test_images_filenames)):
    filename=test_images_filenames[i]
    ima=cv2.imread(filename)
    gray=cv2.cvtColor(ima,cv2.COLOR_BGR2GRAY)
    kpt,des=SIFTdetector.detectAndCompute(gray,None)
    predictions = myknn.predict(des)
    values, counts = np.unique(predictions, return_counts=True)
    predictedclass = values[np.argmax(counts)]
    print 'image '+filename+' was from class '+test_labels[i]+' and was predicted '+predictedclass
    numtestimages+=1
    if predictedclass==test_labels[i]:
        numcorrect+=1

print 'Final accuracy: ' + str(numcorrect*100.0/numtestimages)

end=time.time()
print 'Done in '+str(end-start)+' secs.'
```

Running the script...

- Running the script should yield near 30.5% accuracy and took 300 secs. on my machine (8 CPUs)
- All the steps in the example script are sequential, you should think of parallelizing stuff...

Tasks!

1. Modularize the code, defining functions for the different steps of the program
2. Implement other performance evaluation measures and experimental protocols
   - k-fold cross validation (**just on train set!!!**) for parameter setting.
   - ROC curves, Precision Recall and F-score, Confusion matrix… (**which one(s) to use? why?**)
3. Global description of the image
   - How to aggregate *n* SIFT descriptors into a single vector?
   - Try other global image descriptors
   - **Report performance following point 2.**
4. Try other classifiers: Bayes, Random Forests, …
   - **Report performance following point 2.**
5. Implement **your own** logistic regression classifier (c.f. Dimos class)
   - **Report performance following point 2.**

Grades, deliverables and deadline

A: B+5

B: C+3+4

C: 1+2

D: otherwise

- Deliver source code and a **<u>short</u>** slide presentation (5~10) of the work done
  - State clearly the targeted tasks
  - 1 slide with a table summarizing the best yielded results and configurations
  - 1 final slide with your own observations and conclusions
- Delivered by Monday 18th at 9AM

The end

Enjoy, and do not hesitate to contact us...

Marçal Rossinyol - marcal@cvc.uab.cat
Ramon Baldrich - ramon@cvc.uab.cat