



# Master in Computer Vision *Barcelona*

**Module 3:** Machine Learning for Computer Vision

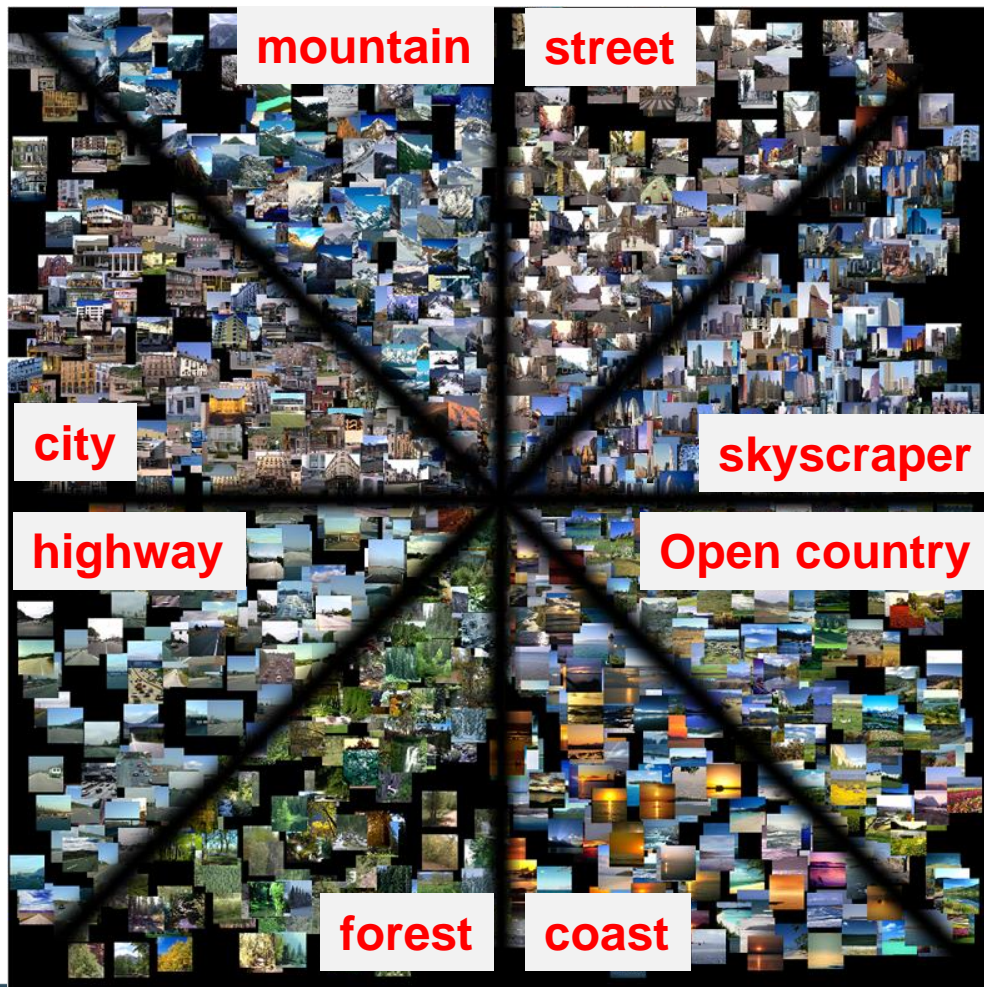
**Project:** Deep learning classification

**Lecturer:** Ramon Baldrich



## Module Goal

The aim of this module is to learn the techniques for category classification: handcrafted and learned.



# 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 Intelligence in Computer Vision

Machine learning for image classification:

- Handcrafted methods: Bag of Words: 2 sessions
- Data driven methods: Deep Convolutional Networks: 3 sessions

# Artificial Intelligence in Computer Vision

Machine learning for image classification:

- Handcrafted methods: Bag of Words: 2 sessions
  - Basic image classification with local features and SVMs
  - The Bag of Visual Words framework
  - Beyond BoVW: Spatial Pyramids, Fisher Vectors

# Artificial Intelligence in Computer Vision

Machine learning for image classification:

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

# Hardware available

## VGG-16

(input 16 x 3 x 224 x 224)

GPU	cuDNN	Forward (ms)	Backward (ms)	Total (ms)
Pascal Titan X	5.1.05	41.59	87.03	128.62
Pascal Titan X	5.0.05	46.16	111.23	157.39
GTX 1080	5.1.05	59.37	123.42	182.79
Maxwell Titan X	5.1.05	62.30	130.48	192.78
GTX 1080 Ti	RTX A6000	1.00x		
	A100 40GB SXM4		1.92x	
	A100 40GB PCIe		1.77x	
	RTX 3090	1.08x		
	V100 32GB	0.89x		
	RTX 3080	0.87x		
	Titan RTX	0.81x		
	RTX 6000	0.80x		
	RTX 8000	0.79x		
	RTX 2080 Ti	0.69x		
Maxwell Titan X	RTX 5000	0.56x		
	GTX 1080 Ti	0.52x		
	RTX 2080 Super MAX-Q	0.39x		
	RTX 2080 MAX-Q	0.36x		
Maxwell Titan X	RTX 2070 MAX-Q	0.34x		
CPU: Dual Xeon E5-2630 v3	None	3101.76	5393.72	8495.48

## RTX 3090



## TITAN X (pascal)



## GTX 1080 Ti



1 per group.

<https://github.com/jcjohnson/cnn-benchmarks>

Master in Computer Vision Barcelona

## Libraries

- Standard libraries:
  - Python + common libraries
  - Numpy + sklearn
- Specialized libraries:
  - Keras
  - Tensorflow
  - Pytorch



# GPU server: User Installation and Usage Manual

- **Connect to the server**      Instructions in virtual campus  
    users/psswd :      group01  
                         group02  
                         group03  
                         group04  
                         group05  
                         group06  
                         group07  
                         group08
  - **Mount your server folder in your local computer to edit locally**
  - **Create a permanent connection to the server (Your process will continue working even if the connection to the server is lost)**
    - # Connect to the server via ssh
    - # Run screen command to create a new connection screen
- \$ screen                      or                      \$ byobu

**Code, installing instructions and dataset in ~/mcv,**  
**Queue management in ~/example**

# GPU server: User Installation and Usage Manual

## Environment installation

```
$ bash ~/mcv/Anaconda3-2019.10-Linux-x86_64.sh
```

yes to all .....

- Do you wish the installer to initialize Anaconda3 in your /home/grupoXX/.bashrc ? yes
- Do you wish to proceed with the installation of Microsoft VSCode? no

exit/login

```
$conda install -c anaconda pydot
```

add the following two lines at the end of file .bashrc

```
export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_PATH
```

```
export CUDA_HOME=/usr/local/cuda
```

```
$ conda install tensorflow-gpu
```

```
$ conda install keras
```

# GPU server: User Installation and Usage Manual

- Check gpu status: `nvidia-smi`

```
master@mastergpu01:~$ nvidia-smi
Wed Jan 18 13:46:18 2017

+-----+
| NVIDIA-SMI 367.57                  Driver Version: 367.57          |
+-----+-----+
| GPU   Name                               Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|====+=====+
|  0  TITAN X (Pascal)         Off | 0000:0B:00.0    Off |                    N/A |
| 40%   70C   P2     157W / 250W | 11705MiB / 12189MiB |      57%      Default |
+-----+-----+

+-----+
| Processes:                                     GPU Memory |
|  GPU       PID    Type    Process name      Usage      |
|====+=====+
|    0         8647    C       python                11703MiB |
+-----+

master@mastergpu01:~$ █
```

- Run your first code:  
copy 'job' file & \*.py files to your working folder  
Edit `mlp_MIT_8_scene.py` to fit your working folder.  
\$ `sbatch job`

# GPU server: User Installation and Usage Manual

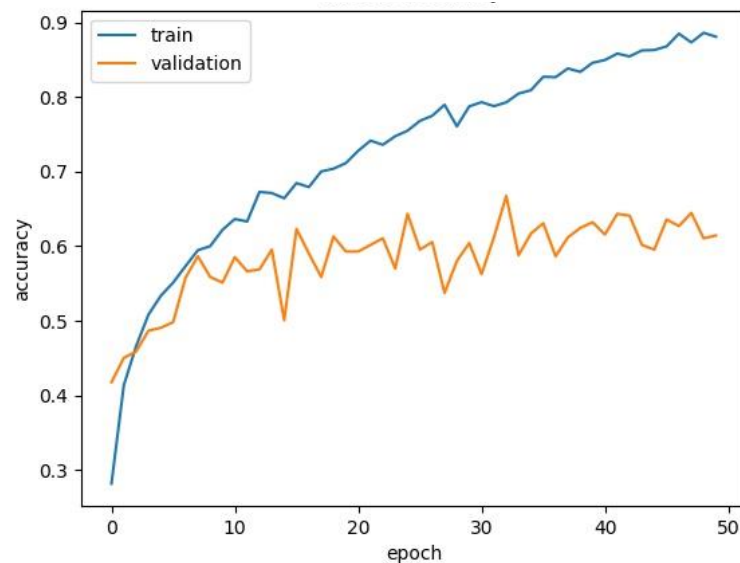
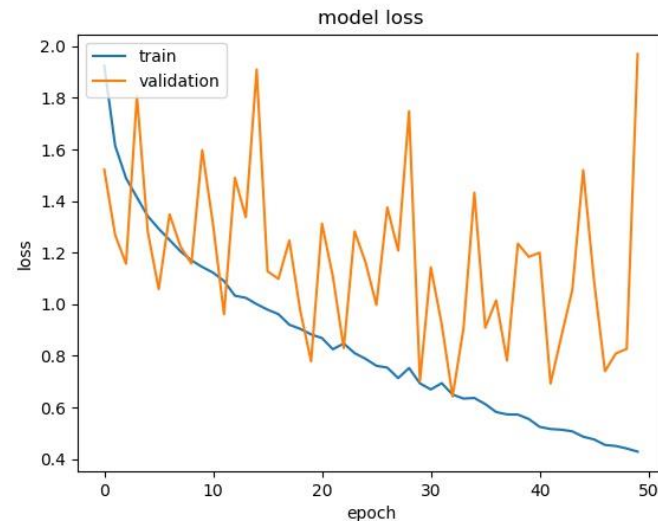
first_input: InputLayer	input:	(None, 32, 32, 3)
	output:	(None, 32, 32, 3)

first: Reshape	input:	(None, 32, 32, 3)
	output:	(None, 3072)

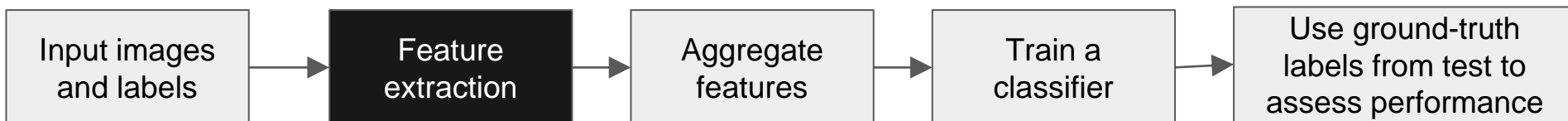
second: Dense	input:	(None, 3072)
	output:	(None, 2048)

dense_1: Dense	input:	(None, 2048)
	output:	(None, 8)

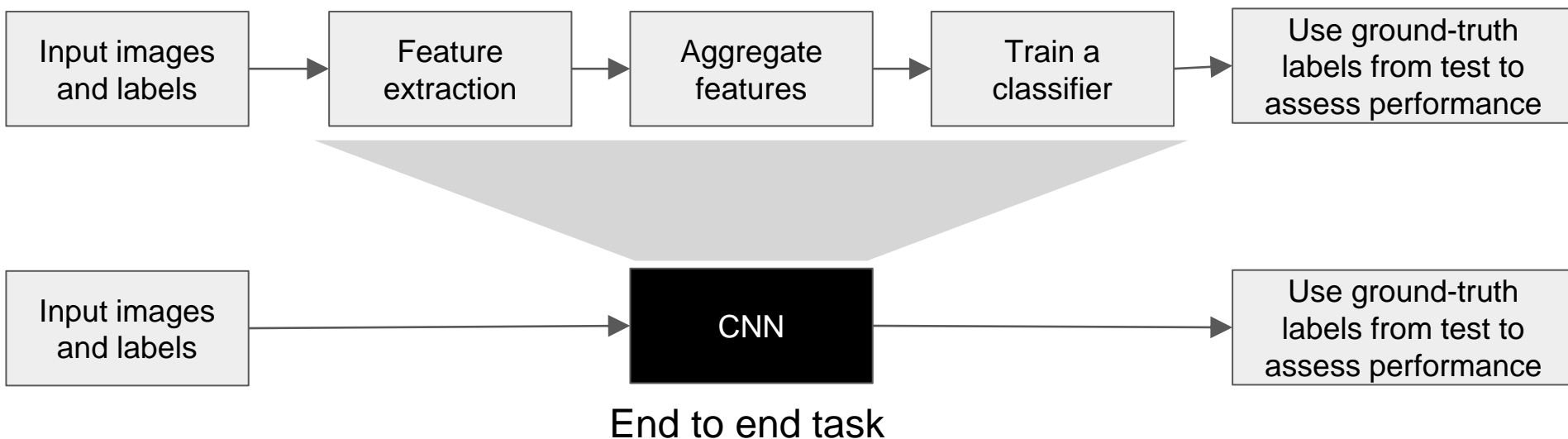
We need something more clever



## Pipeline of the project W3



## Pipeline of the project W4 and W5



# Keras: first example

## # create model

```
model = Sequential()
model.add(Dense(12, input_dim=8, init='uniform', activation='relu'))
model.add(Dense(8, init='uniform', activation='sigmoid'))

inputs = Input(shape=None))
x = Dense(12, init='uniform', activation='relu', name='fc1')(x)
x = Dense(8, init='uniform', activation='sigmoid', name='predictions')(x)
model = Model(inputs, x, name='example')
```

W3-5

## # Compile model

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

## # Fit the model

```
model.fit(X, Y, nb_epoch=150, batch_size=10)
```

W3-4

## # evaluate the model

```
scores = model.evaluate(X, Y)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

## # predict with the model

```
features = model.predict(X)
```

W3-4

K

Keras Documentation

←

→

↻

Es seguro

https://keras.io

☆

K

Keras Documentation

Search docs

Home

Keras: Deep Learning library for Theano and TensorFlow

You have just found Keras.

Guiding principles

Getting started: 30 seconds to Keras

Installation

Switching from TensorFlow to Theano

Support

Why this name, Keras?

Getting started

Guide to the Sequential model

Guide to the Functional API

FAQ

Models

About Keras models

Sequential

Model (functional API)

GitHub

Next »

Docs » Home

Edit on GitHub

# Keras: Deep Learning library for Theano and TensorFlow

## You have just found Keras.

Keras is a high-level neural networks library, written in Python and capable of running on top of either **TensorFlow** or **Theano**. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through total modularity, minimalism, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Supports arbitrary connectivity schemes (including multi-input and multi-output training).
- Runs seamlessly on CPU and GPU.

Read the documentation at **Keras.io**.

Keras is compatible with: Python 2.7-3.5



## Week 3: learnt features in Bow

Goals:

- Understand MLP topology
- Learn to extract features from a network
- Compare learnt features with handcrafted features

model = Sequential()

first_input: InputLayer	input:	(None, 32, 32, 3)
	output:	(None, 32, 32, 3)

model.add(Reshape((32\*32\*3,),input\_shape=(32, 32, 3),name='first'))

first: Reshape	input:	(None, 32, 32, 3)
	output:	(None, 3072)

model.add(Dense(units=2048, activation='relu', name='second'))

second: Dense	input:	(None, 3072)
	output:	(None, 2048)

model.add(Dense(units=8, activation='softmax'))

dense_1: Dense	input:	(None, 2048)
	output:	(None, 8)

# Use feature maps as learnt descriptor

## Deep filter banks for texture recognition and segmentation

Mircea Cimpoi  
University of Oxford

mircea@robots.ox.ac.uk

Subhransu Maji  
University of Massachusetts, Amherst

smaji@cs.umass.edu

Andrea Vedaldi  
University of Oxford

vedaldi@robots.ox.ac.uk

### Abstract

Research in texture recognition often concentrates on the problem of material recognition in uncluttered conditions, an assumption rarely met by applications. In this work we conduct a first study of material and describable texture attributes recognition in clutter, using a new dataset derived from the OpenSurface texture repository. Motivated by the challenge posed by this problem, we propose a new texture descriptor, FV-CNN, obtained by Fisher Vector pooling of a Convolutional Neural Network (CNN) filter bank. FV-CNN substantially improves the state-of-the-art in texture, material and scene recognition. Our approach achieves 79.8% accuracy on Flickr material dataset and 81% accuracy on MIT indoor scenes, providing absolute gains of more than 10% over existing approaches. FV-CNN easily transfers across domains without requiring feature adaptation as for methods that build on the fully-connected layers of CNNs. Furthermore, FV-CNN can seamlessly incorporate multi-scale information and describe regions of arbitrary shapes and sizes. Our approach is particularly suited at localizing “stuff” categories and obtains state-of-the-art results on MSRC segmentation dataset, as well as promising results on recognizing materials and surface attributes in clutter on the OpenSurfaces dataset.

### 1. Introduction

Texture is ubiquitous and provides useful cues of mate-



Figure 1. **Texture recognition in clutter.** Example of top retrieved texture segments by attributes (top two rows) and materials (bottom) in the OpenSurfaces dataset.

assumption that textures fill images. Thus, they are not necessarily representative of the significantly harder problem of recognising materials in natural images, where textures appear in clutter. Building on a recent dataset collected by the computer graphics community, the **first contribution** of this paper is a *large-scale analysis of material and perceptual texture attribute recognition and segmentation in clutter* (Fig. 1 and Sect. 2).

Motivated by the challenge posed by recognising texture in clutter, we develop a new texture descriptor. In the simplest terms a texture is characterized by the arrangement of local patterns, as captured in early works [26, 41] by the distribution of local “filter bank” responses. These filter banks were designed to capture edges, spots and bars at

- Extract features
- Agregate channels
  - Statistics
  - Dimensioanlity reduction
- Apply BoW/Fisher
- Apply classifier

Cimpoi, M., Maji, S., & Vedaldi, A. (2015). Deep filter banks for texture recognition and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3828-3836).

# Experimentation

- MLP on small images: end to end vs svm
- MLP as dense descriptor: end to end vs BoW

# Experimentation

MLP on small images: end to end vs svm

- Redefine the structure of the network (adding new layers, changing image size, ...)
- Extract the output from a given layer as descriptor and apply svm on it

# Experimentation

MLP as dense descriptor: end to end vs BoW

- Divide the image into small patches
- Extract the prediction for each patch, and aggregate the final prediction (end to end)
- Take each patch output, given a layer, as a dense descriptor and apply Bow

utils.py

mlp\_MIT\_8\_scene.py

patch\_based\_mlp\_MIT\_8\_scene.py

Code

## Creating a Network

```
model = Sequential()
model.add(Reshape((IMG_SIZE*IMG_SIZE*3,),input_shape=(IMG_SIZE, IMG_SIZE, 3),name='first'))
model.add(Dense(units=2048, activation='relu',name='second'))
model.add(Dense(units=8, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

Sub model

```
model2 = Model(input=model.input, output=base_model.get_layer('second').output)
features = model2.predict(x)
```

## Feeding data to the network

## Code

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    DATASET_DIR+'/train', # this is the target directory
    target_size=(IMG_SIZE, IMG_SIZE), # all images will be resized to IMG_SIZExIMG_SIZE
    batch_size=BATCH_SIZE,
    classes = ['coast','forest','highway','inside_city','mountain','Opencountry','street','tallbuilding'],
    class_mode='categorical') # since we use binary_crossentropy loss, we need categorical labels

# this is a similar generator, for validation data
validation_generator = test_datagen.flow_from_directory(
    DATASET_DIR+'/test',
    target_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE,
    classes = ['coast','forest','highway','inside_city','mountain','Opencountry','street','tallbuilding'],
    class_mode='categorical')
```



# Code

## Training the network

```
history = model.fit_generator(  
    train_generator,  
    steps_per_epoch=1881 // BATCH_SIZE,  
    epochs=50,  
    validation_data=validation_generator,  
    validation_steps=807 // BATCH_SIZE)
```

## Getting predictions from inner layers

```
model_layer = Model(input=model.input, output=model.get_layer('second').output)  
#get the features from images  
features = model_layer.predict(x)
```

# Code

## How to plot learning curves

```
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.savefig('accuracy.jpg')
plt.close()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.savefig('loss.jpg')
```

# Tasks

## Understanding network topology

1. Add/change layers in the network topology
2. Given an image, get the output of a given layer
3. Manage to merge multiple outputs from a single image in an end to end network

## Compare learnt features vs handcrafted features

4. Extract a single feature from an input and apply to svm, compare to end to end network
5. Extract multiple features from an image and apply BoW, compare to end to end network

# Grades, deliverables and deadline

Oriental grade:

A: B+ 5

B: C + 3

C: 1+2+4

D: otherwise

- Deliver source code and a **short** slide presentation of the work done
  - For each task, all the carried tests with their associated results
  - 1 slide summarizing the best yielded result and configuration for each task
- Should be delivered by Monday 24th at 10AM