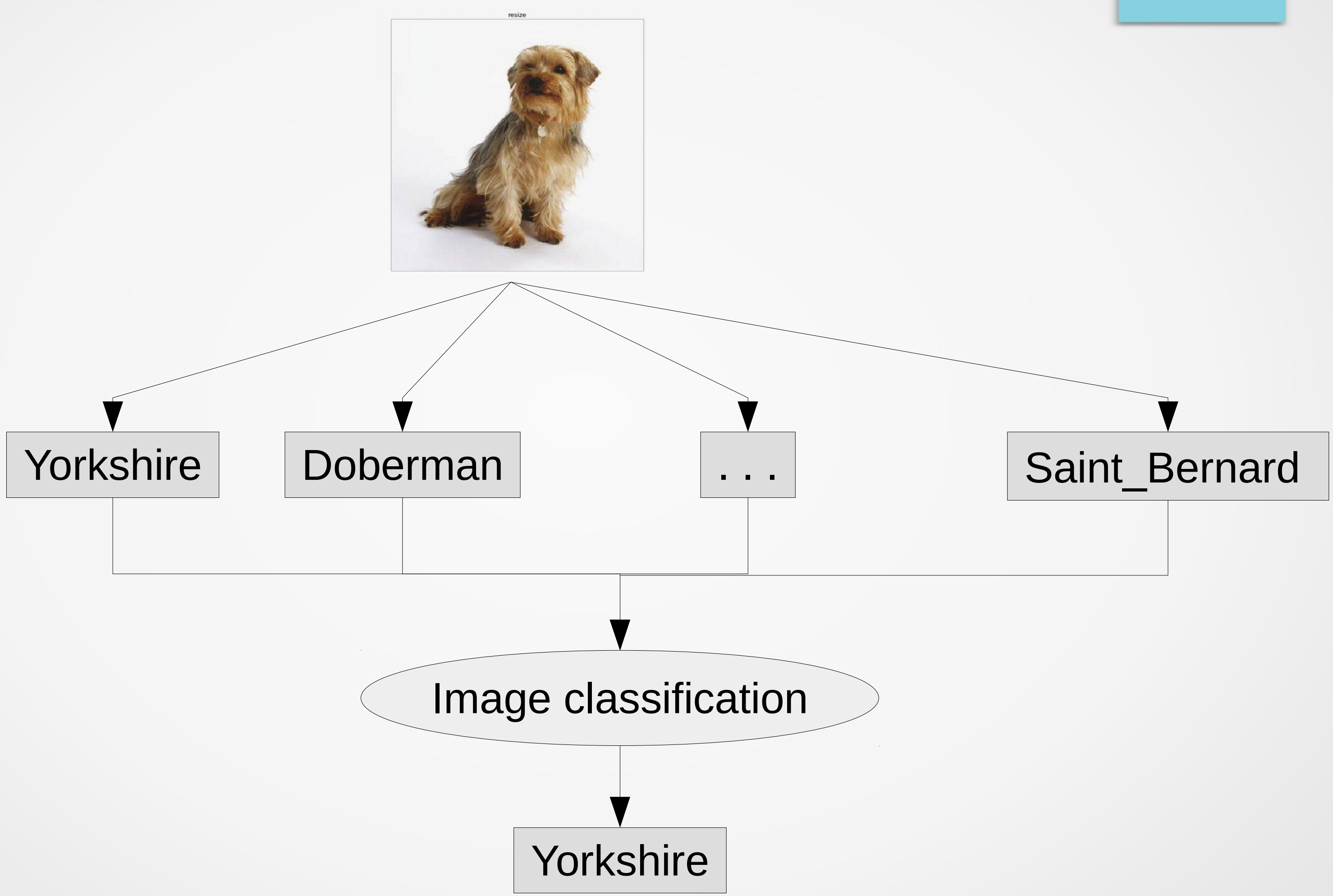
Parcours Datascientist : projet 7

Indexation automatique d’images

issues de la banque d’images Standford Dogs Dataset

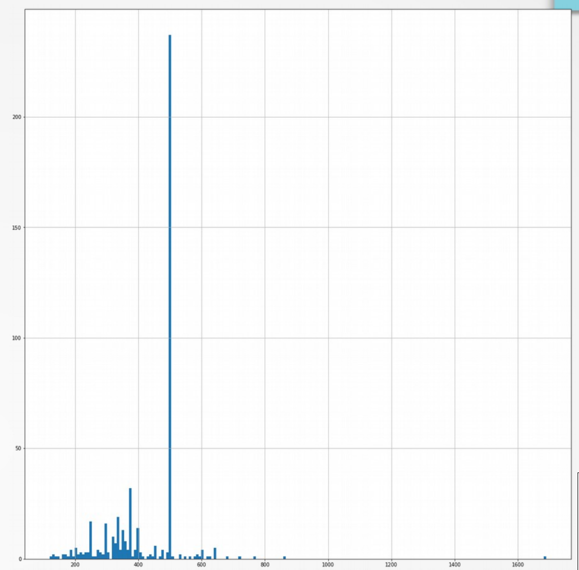
François BANGUI

Problem classification type

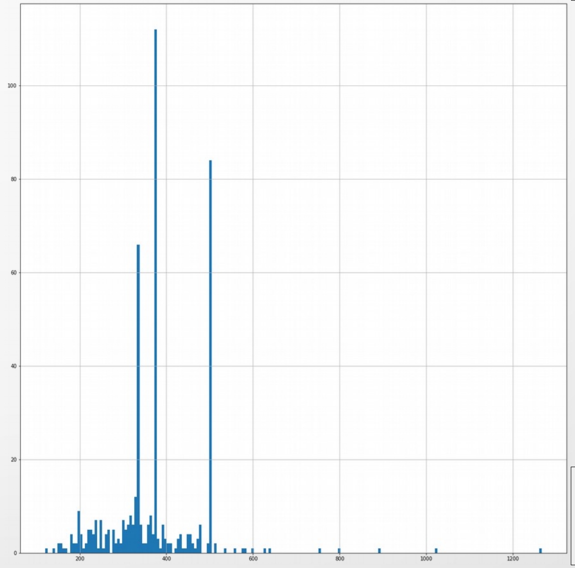


Multi-class classification

Standford dogs database



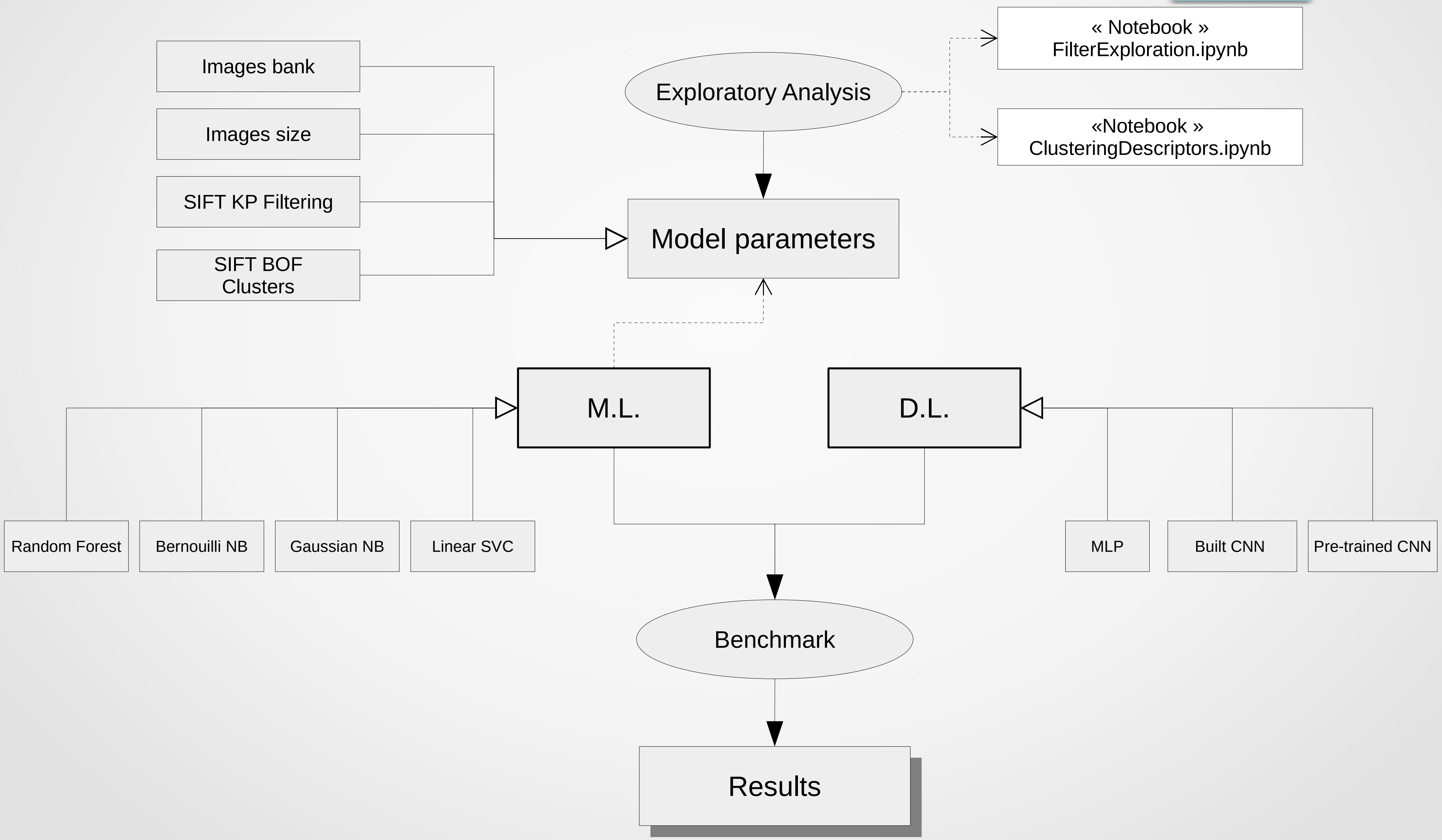
200 ≤ X <= 1700



100 ≤ Y <= 1300

- 120 breeds
- ~ 150 images per breed
- ~ 18 000 images

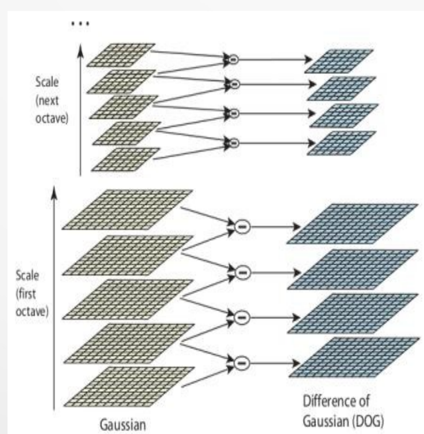
M.L vs D.L : Global benchmark scheme



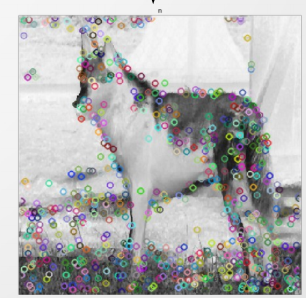
M.L. : Features extraction with SIFT

Key Points Extraction

- Connected pixels detection (blob)
  - Scaling image
  - Gaussian diff

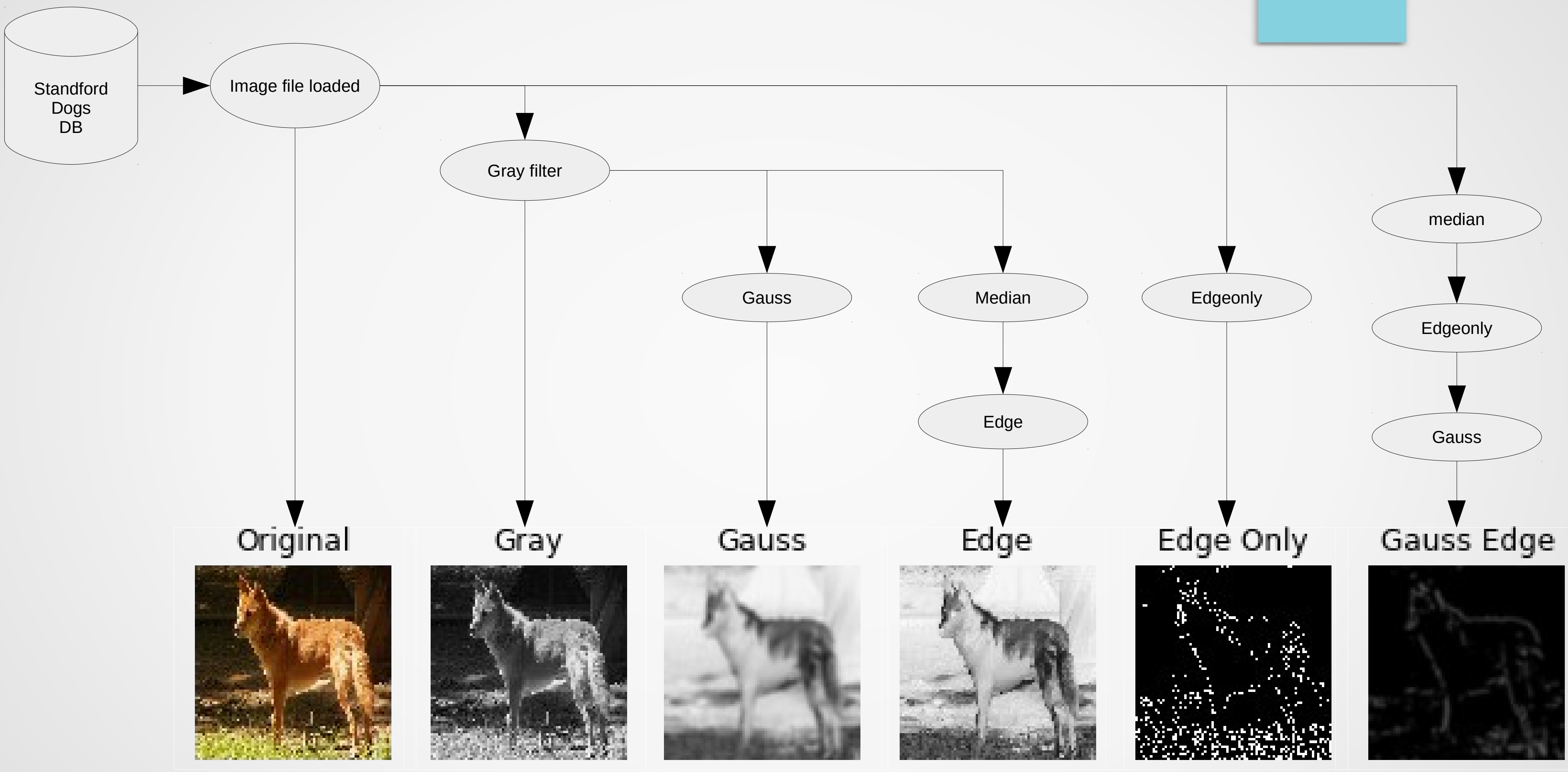


Key Points
Geometric invariant properties
Scaled invariant
Vector 128



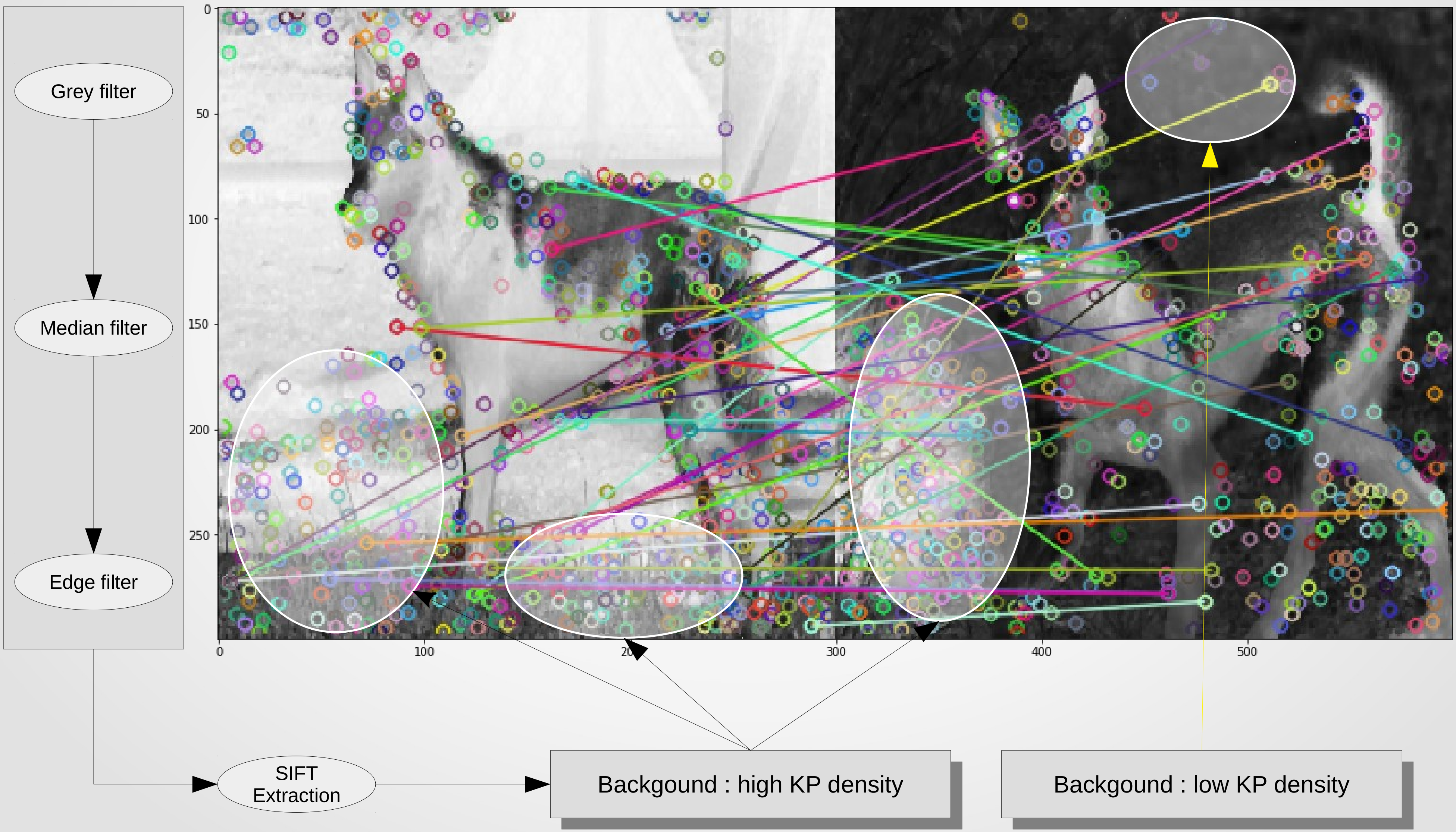
2 differents vectors → Same KP

M.L. : Filters exploration

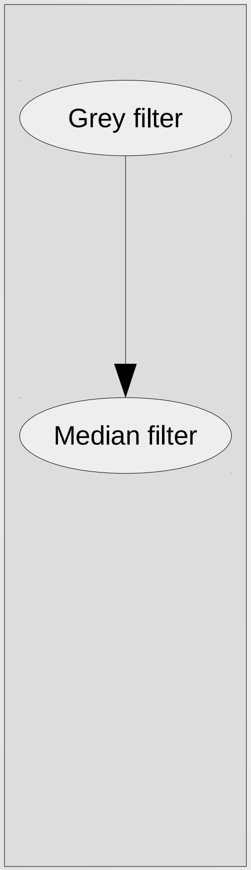


Aim : Most efficient SIFT extraction

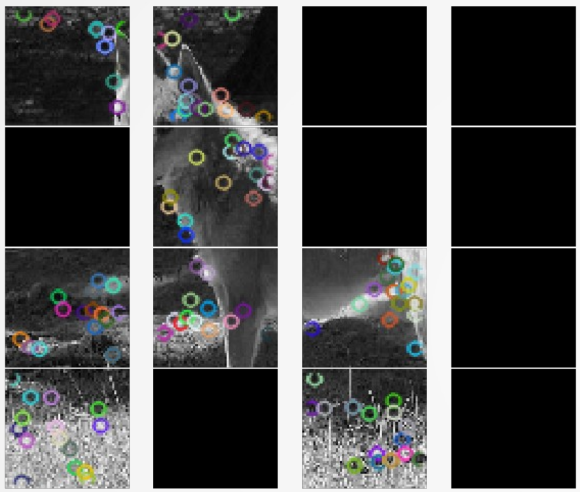
M.L. : KP extraction efficiency

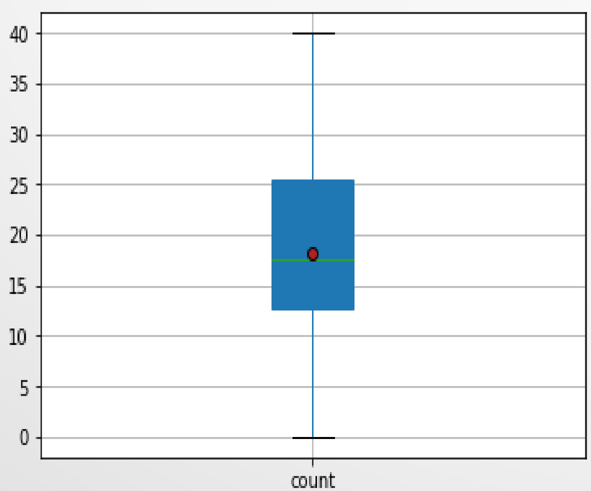


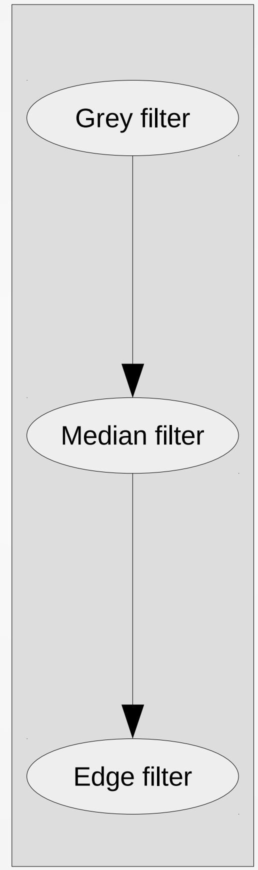
SIFT KP density per splitted image



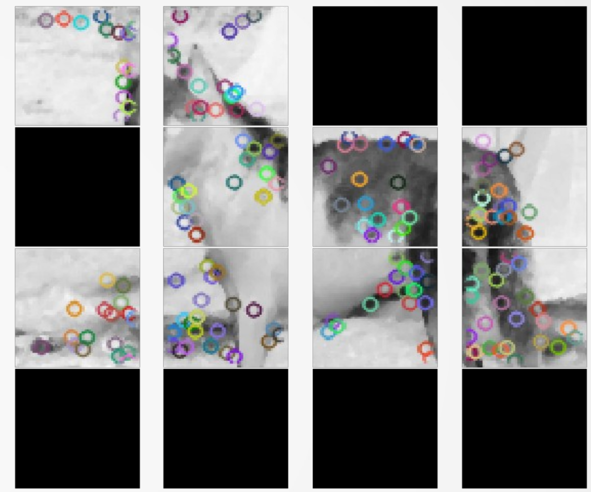
Gray : KP = 341 → 290 /-15 %

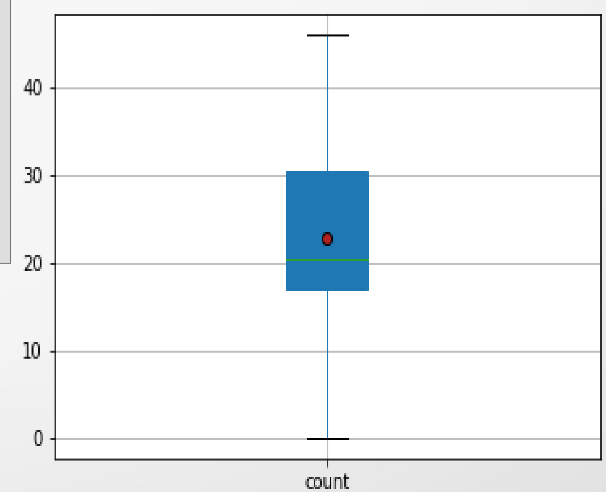




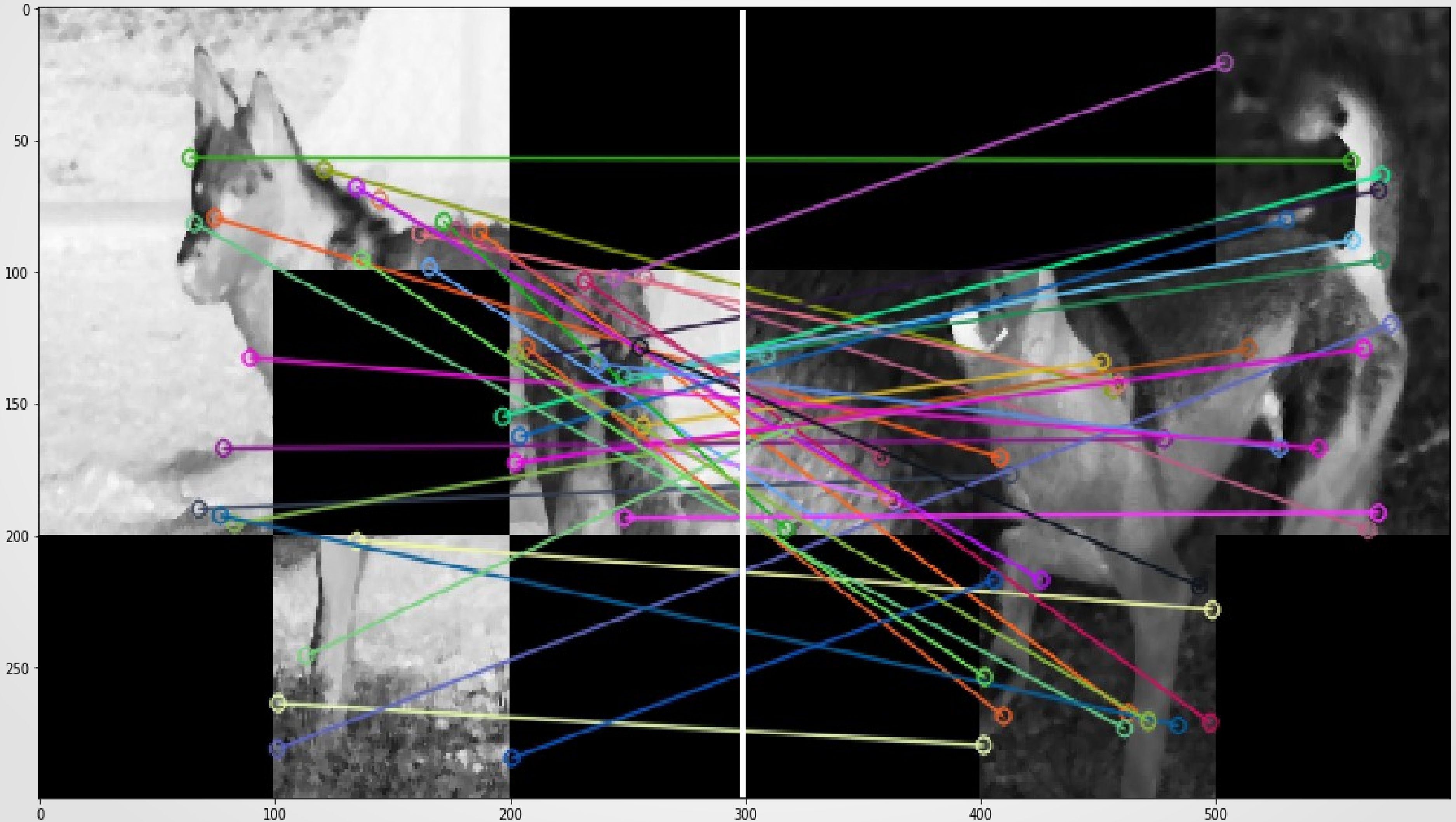


Edge: KP = 450 → 366 / -20 %

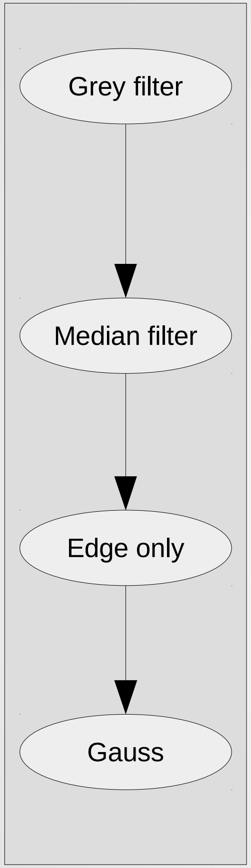


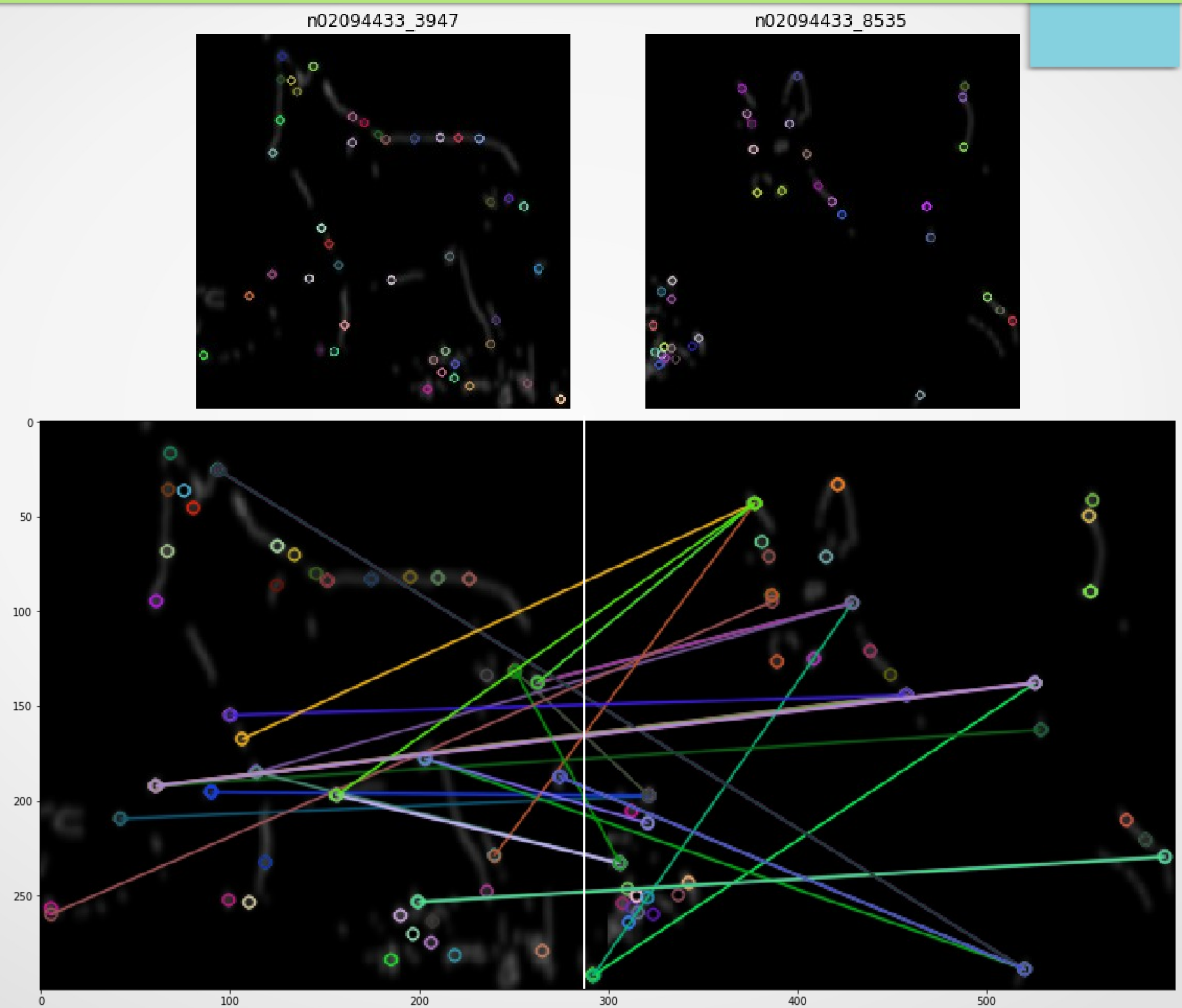


Filtered SIFT key points matching

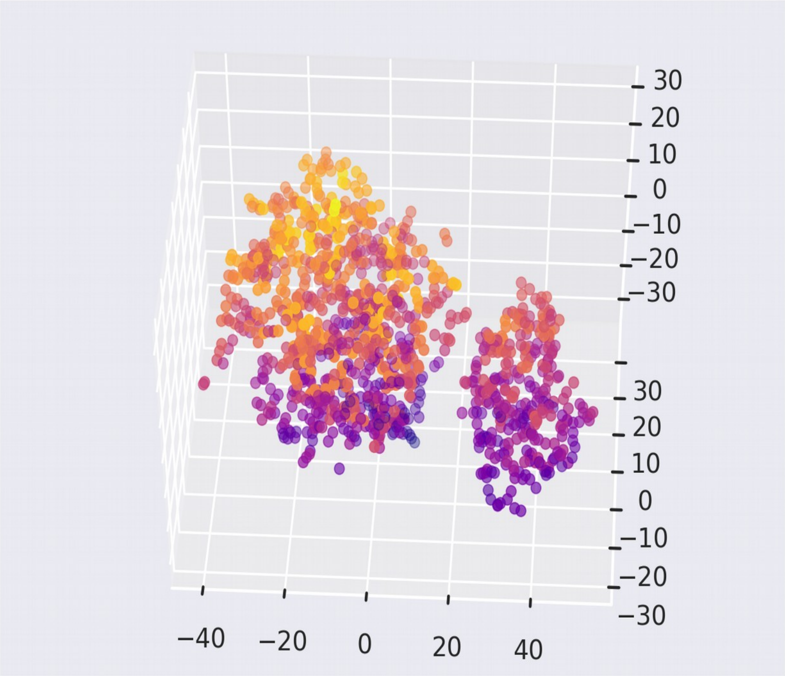


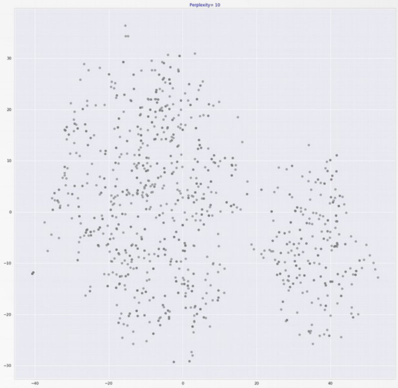
Filtered SIFT key points matching



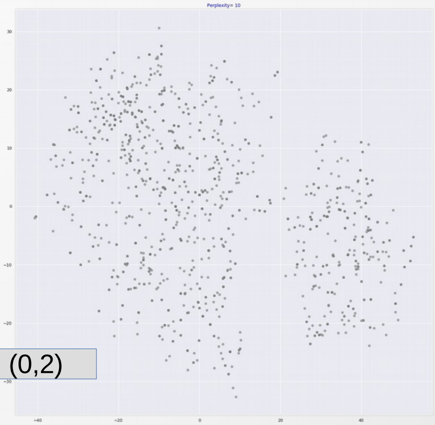


SIFT descriptors t-SNE



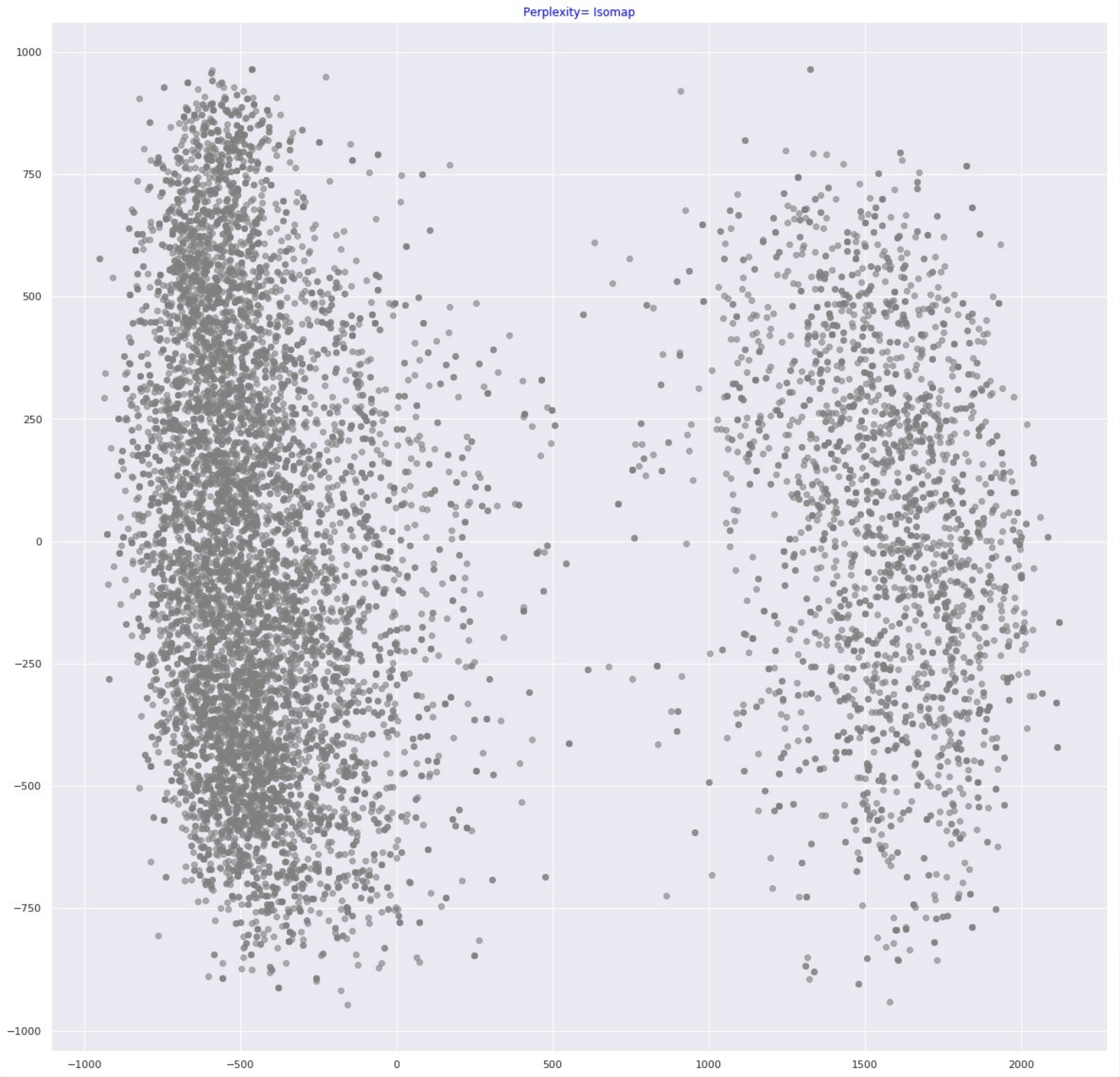


(0,1)

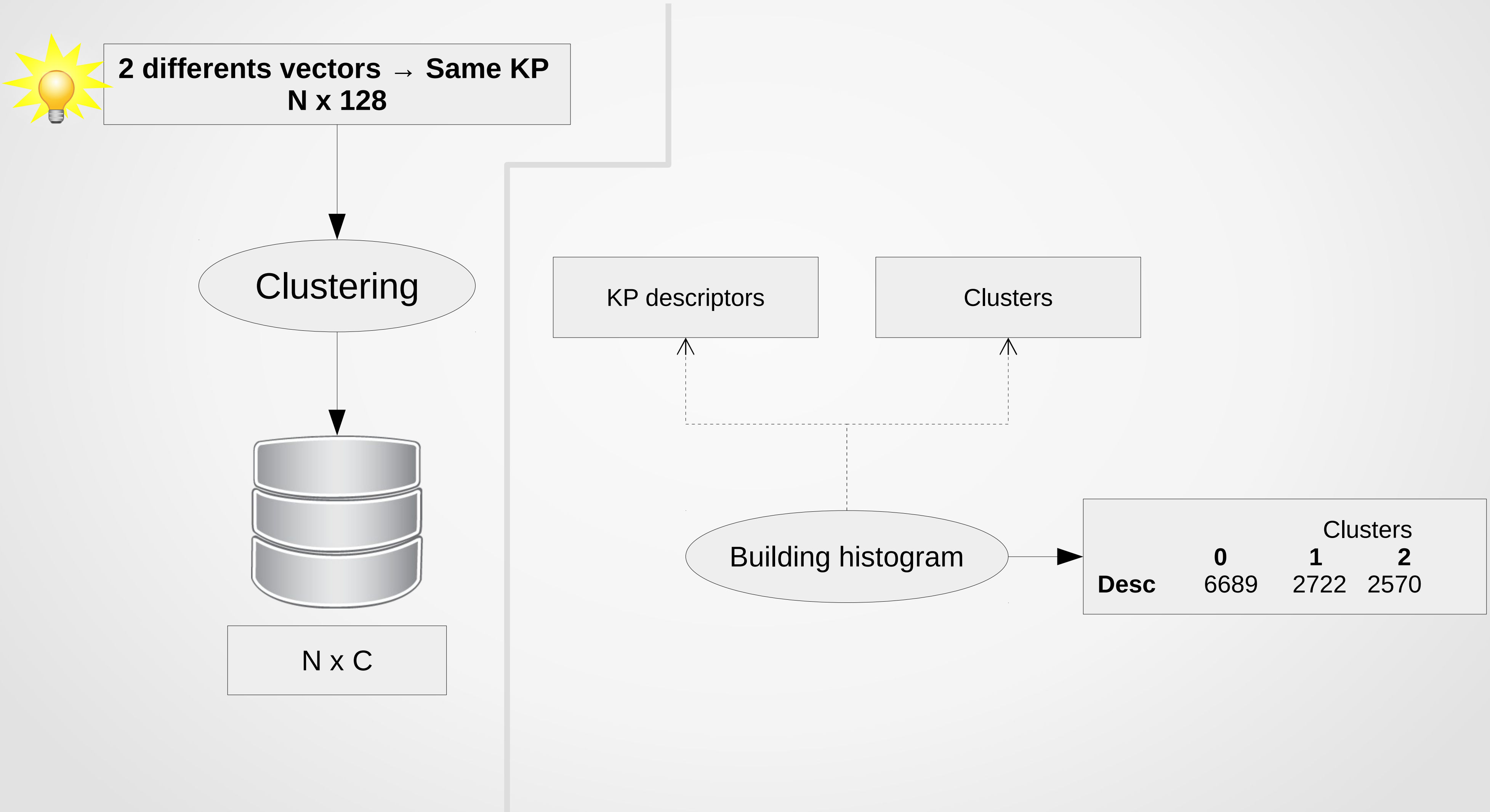


(0,2)

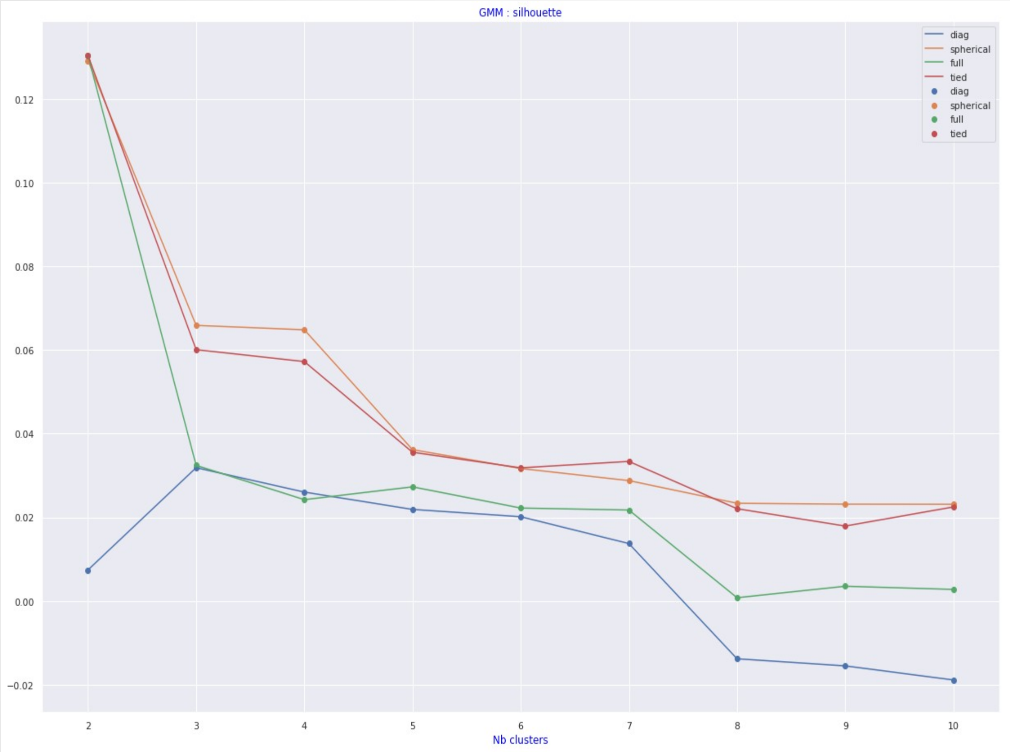
ISOMAP dimension reduction

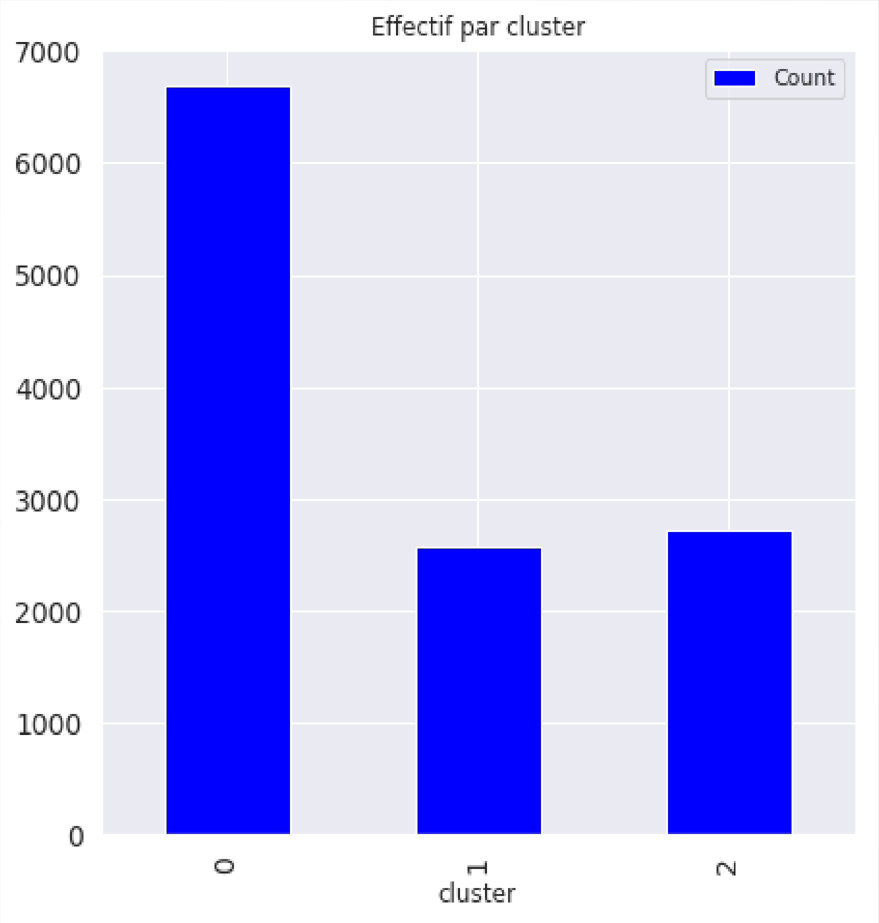


Bag Of Features



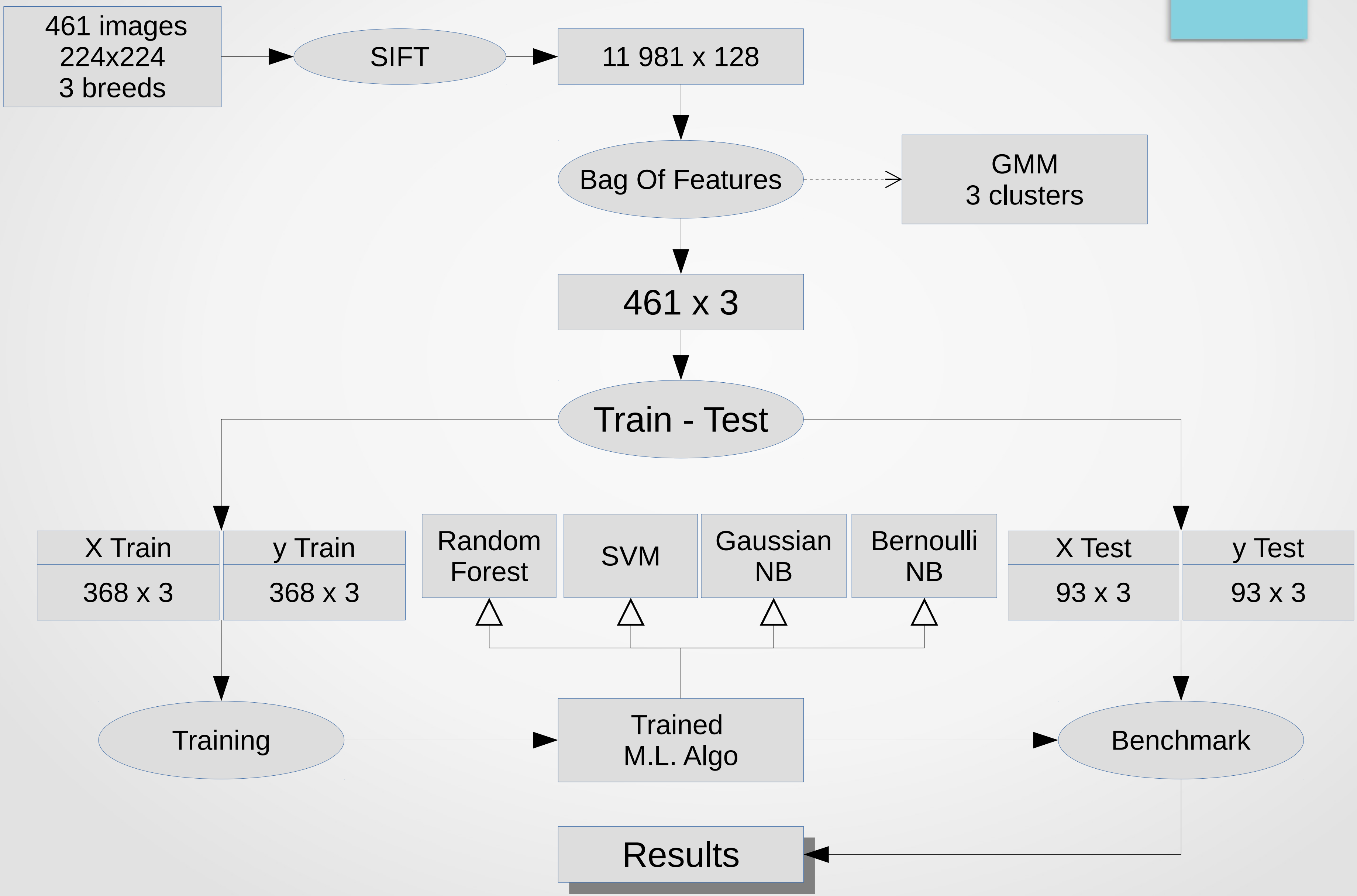
GMM clustering



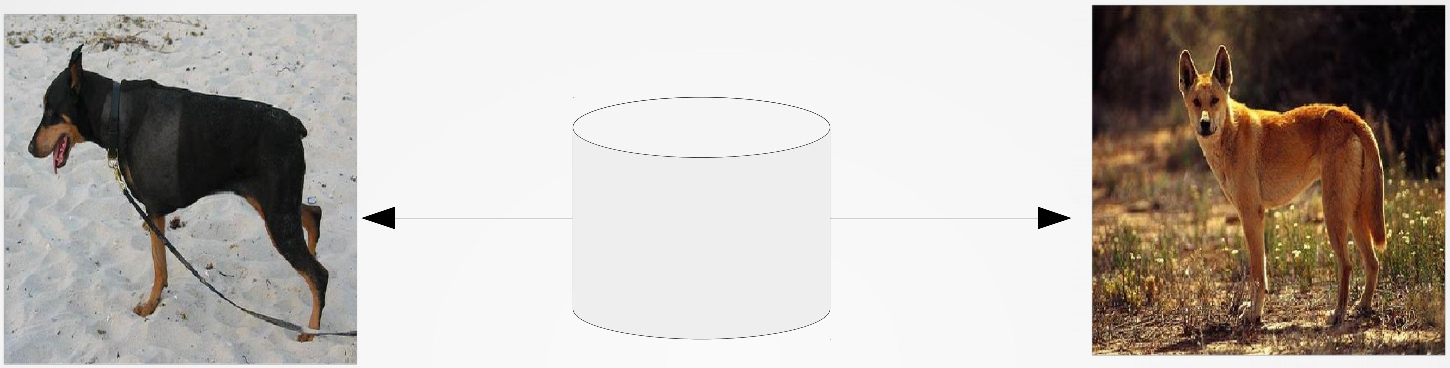


	Count
cluster	
0	6689
2	2722
1	2570

M.L. : model building



M.L. : binary classification

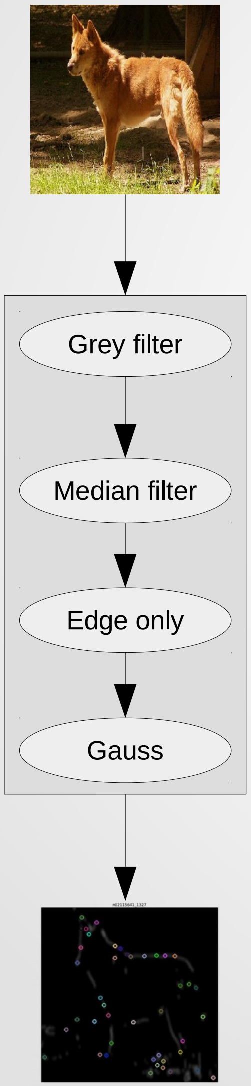


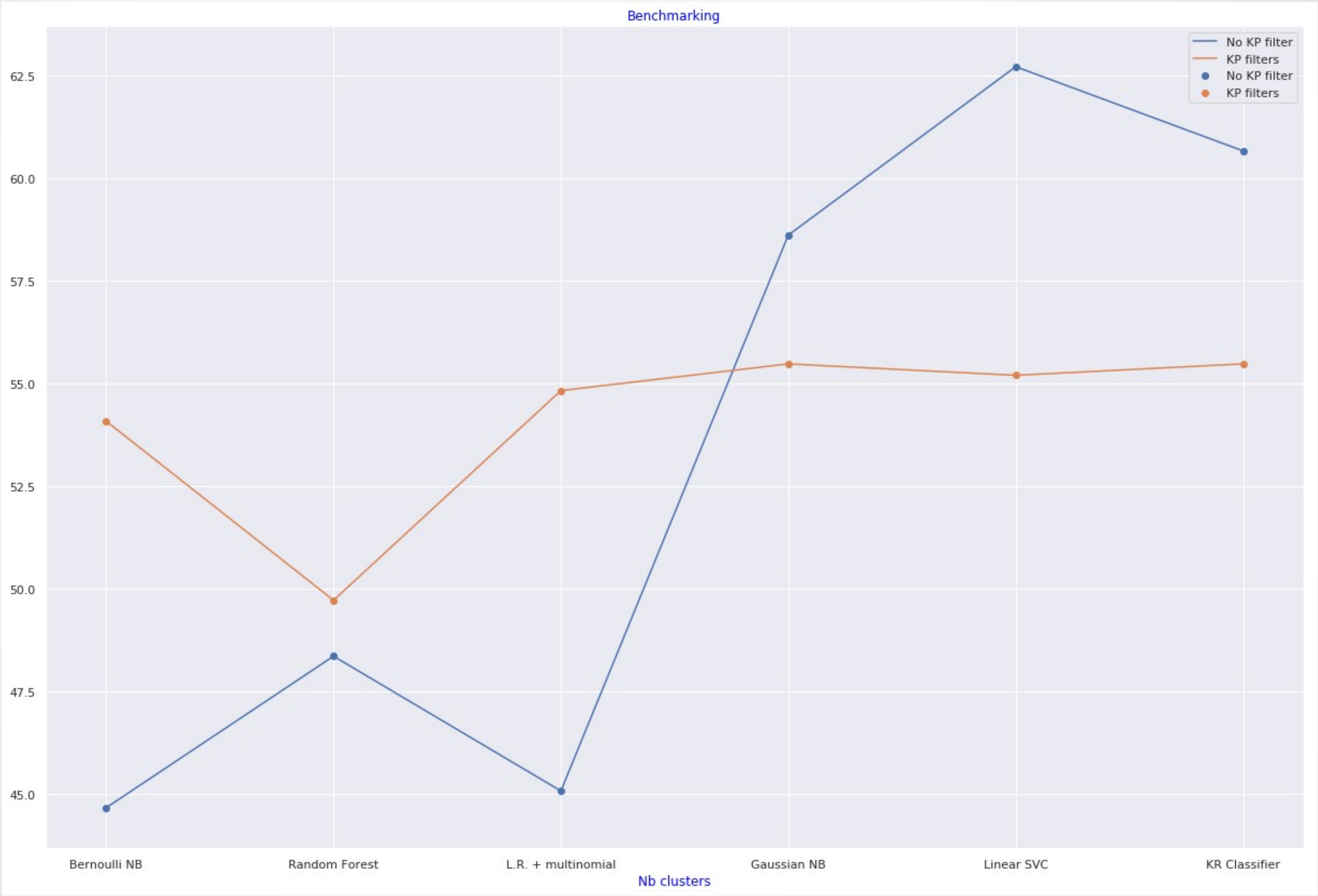
Doberman

dingo

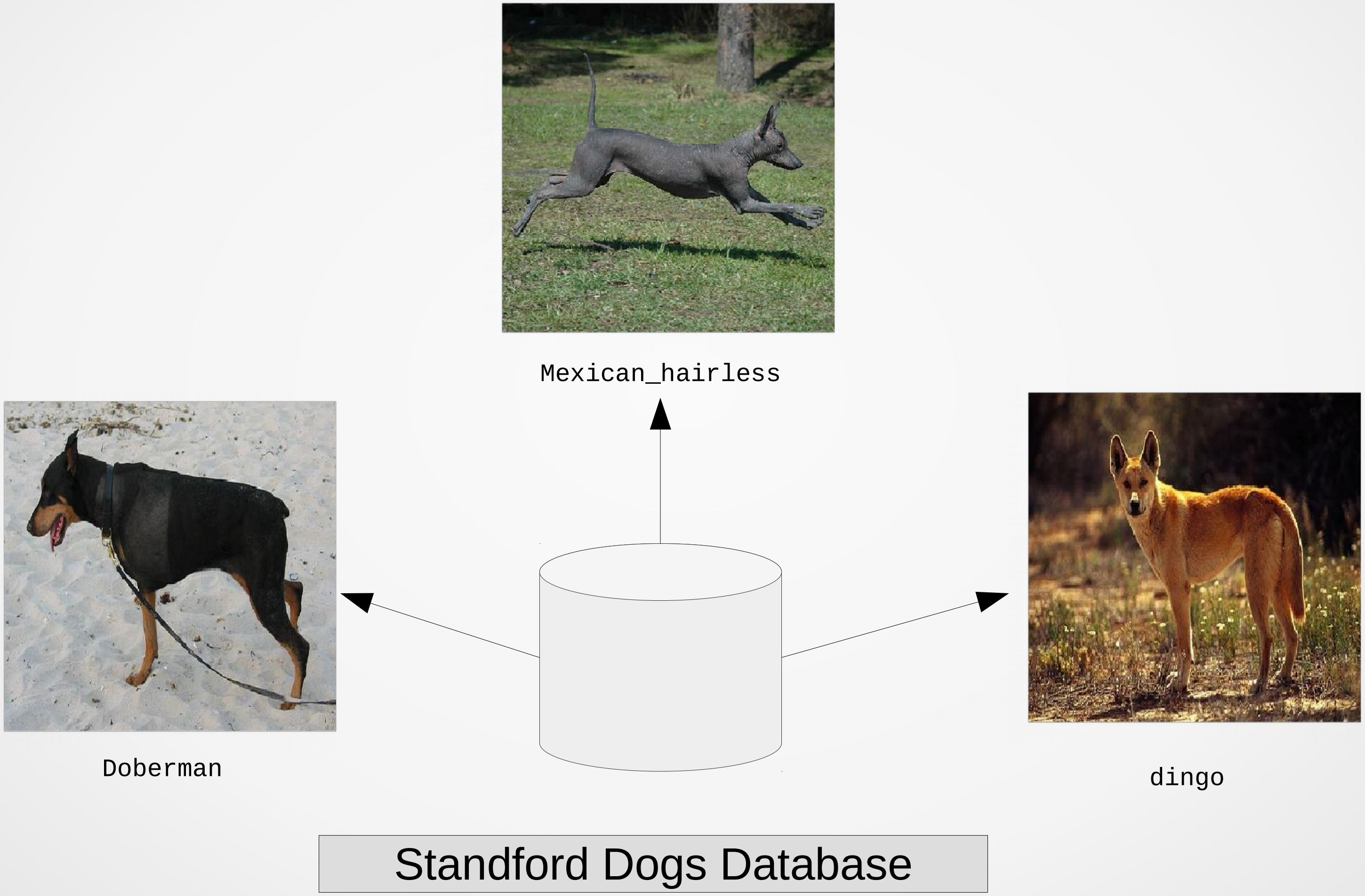
Standford Dogs Database

M.L. : binary classification : no KP filter



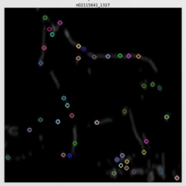


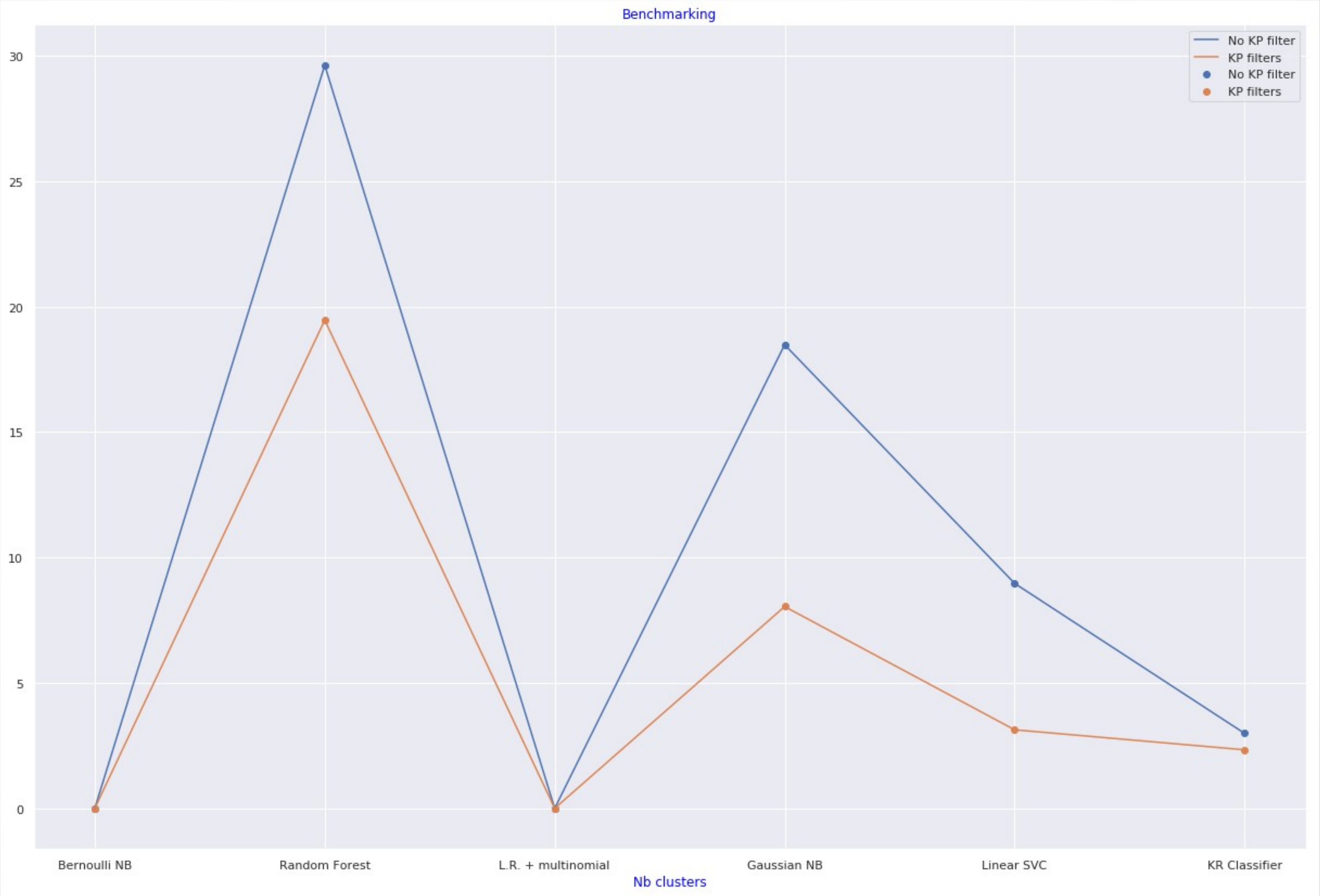
3 breeds from Standford dogs database



M.L. : benchamrk of 3 breeds

Grey filter → Median filter → Edge only → Gauss





M.L. conclusions

- Binary classification : better than a random classifier
- Multi-class classification : weeker then a random classifier
- Feature engineering based on SIFT : need to be improved
- Linear algorithm can’t efficiently deals multi-class problem
- Hypothesis made over breeds lead to a biased solution

Neural Network experience

Neural Networks

Multi-layer perceptron

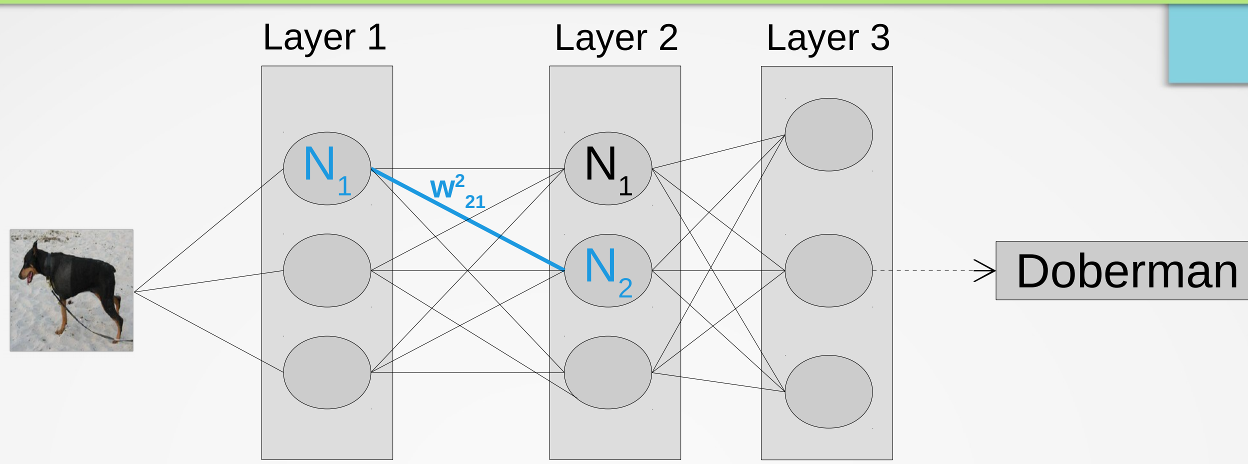
Built CNN

Pre-trained CNN

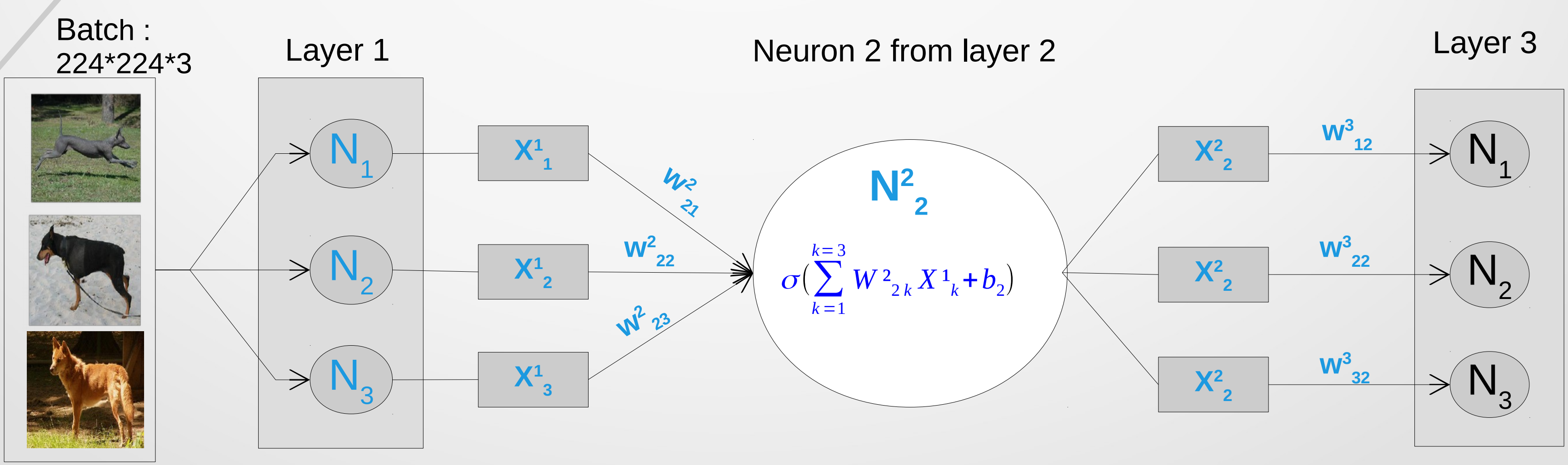
CNN Transfer learning : partial tuning

CNN Transfer learning : fine-learning

M.L.P & Softmax: Architecture feed forward



- More layers ⇒ More sophisticated classification
- W^{2}_{21} : weight belonging to neuron 2 into layer 2 coming from neuron 1 into layer 1
- Training with batch of
- Multi-class classification ⇒ Use of Softmax



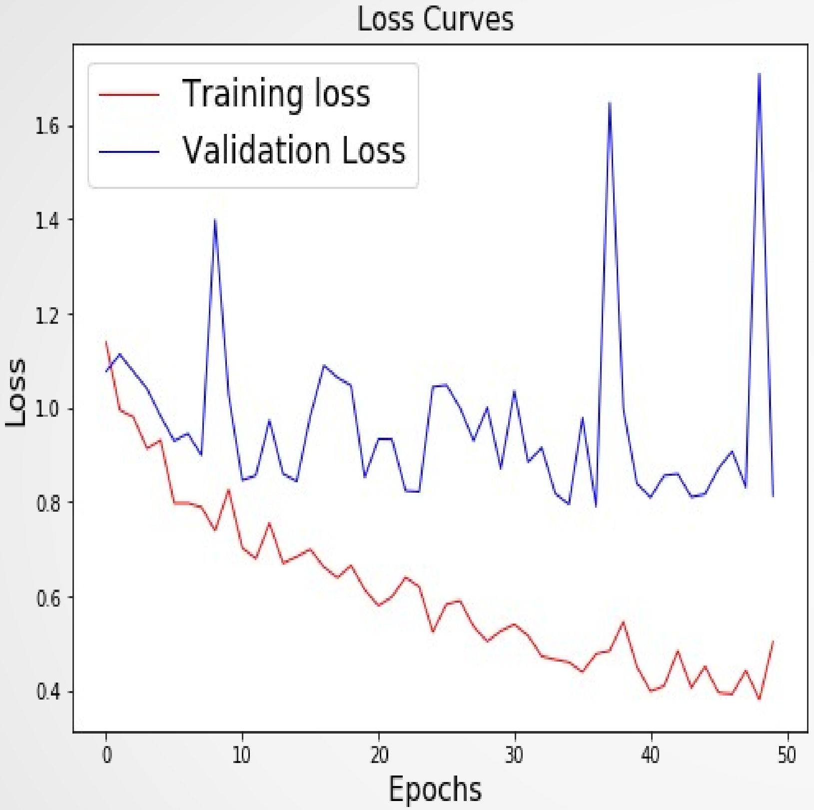
N.N. : background propagation

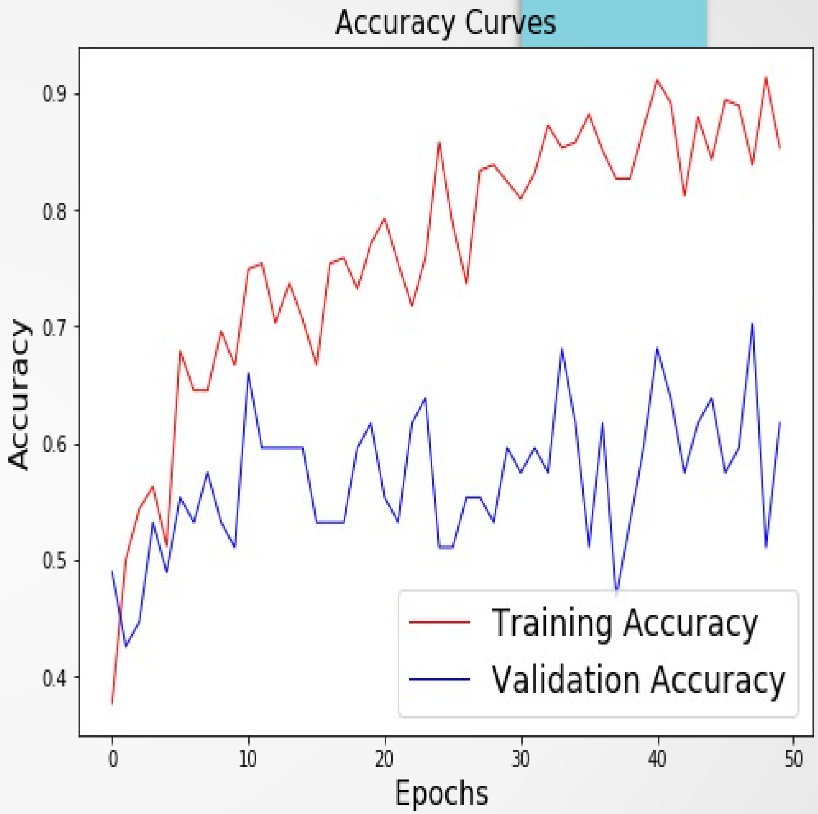
B.P. algorithm : iterative method in order to update Weights and Bias

1. Last layer : Compute error on thanks to Cost function and activation function A :
   - $\varepsilon^{Last} = \frac{\partial Cost}{\partial X^{Last}} * A'((X * \omega + b)^{Last})$ With $Cost(A) = \frac{-1}{node} \sum_{1}^{node} Y \ln(A) = \sum_{1}^{node} p \ln(A)$
2. Compute error on Layer L thanks to error on layer Layer L+1
   - $\varepsilon^{layer} = (\omega^{layer+1})^T * \varepsilon^{layer+1} \odot Act'((X * \omega + b)^{layer})$ $where\ X \odot Y\ is\ Hadamard\ product\ vector\ between\ X\ with\ Y$
3. Compute gradient components for Cost function, for any neuron J into any layer
   - $\frac{\partial Cost}{\partial b_j^{layer}} = \varepsilon_j^{layer}$
   - $For\ any\ neuron\ j \in any\ layer : \frac{\partial Cost}{\partial \omega_{jk}^{layer}} = X_k^{layer-1} \varepsilon_j^{layer} = (\nabla_{\omega} Cost)_{jk}$

   ⇐ average over all layer nodes
4. Thanks to cost gradient, update weight from neuron J in any layer :
   - $For\ any\ neuron\ j \in any\ layer : \omega_j^{next} = \omega_j^{now} - \eta * \nabla_{\omega_j} Cost$
     $For\ any\ neuron\ j \in any\ layer : b_j^{next} = b_j^{now} - \eta * \nabla_{b_j} Cost$

MLP

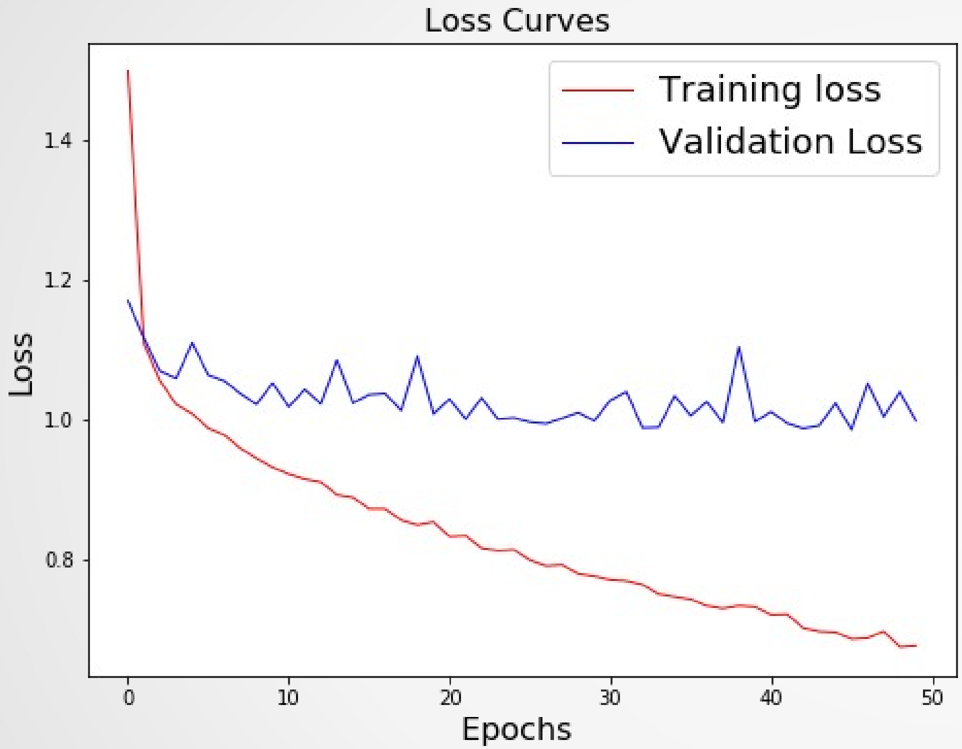


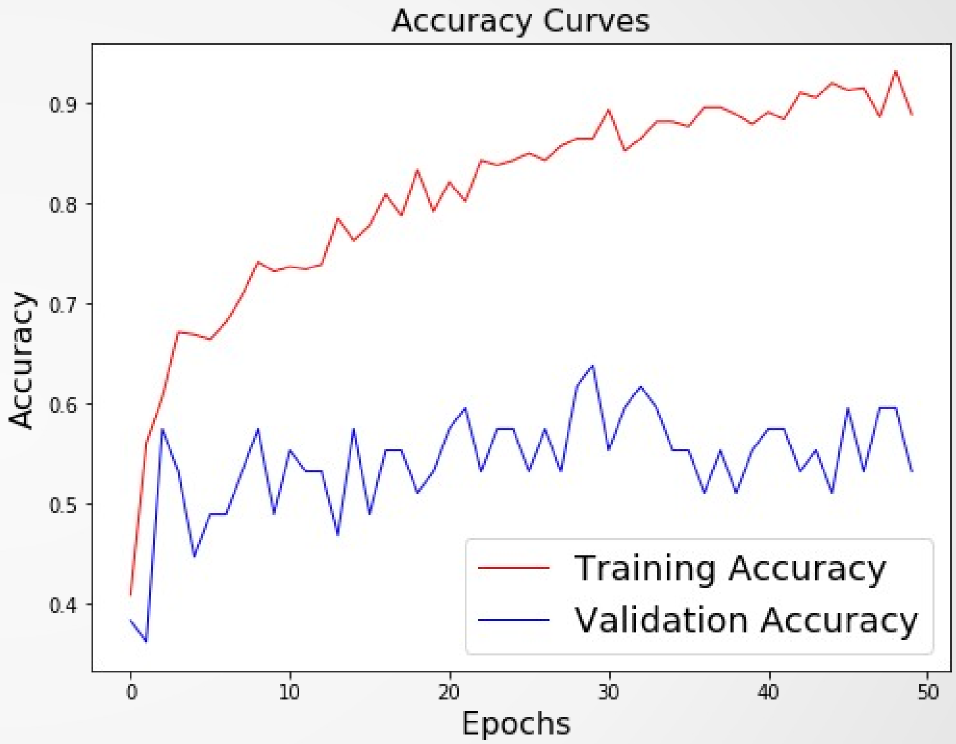


- 3 layers
- 461 Images / 3 breeds / 322 trained images / 139 tested images
- Accuracy : → 60 %

- Over-fitting issue

MLP : overfitting reduction L2 regularization





- 3 layers
- 461 Images / 3 breeds / 322 trained images / 139 tested images
- Accuracy : → 50 %

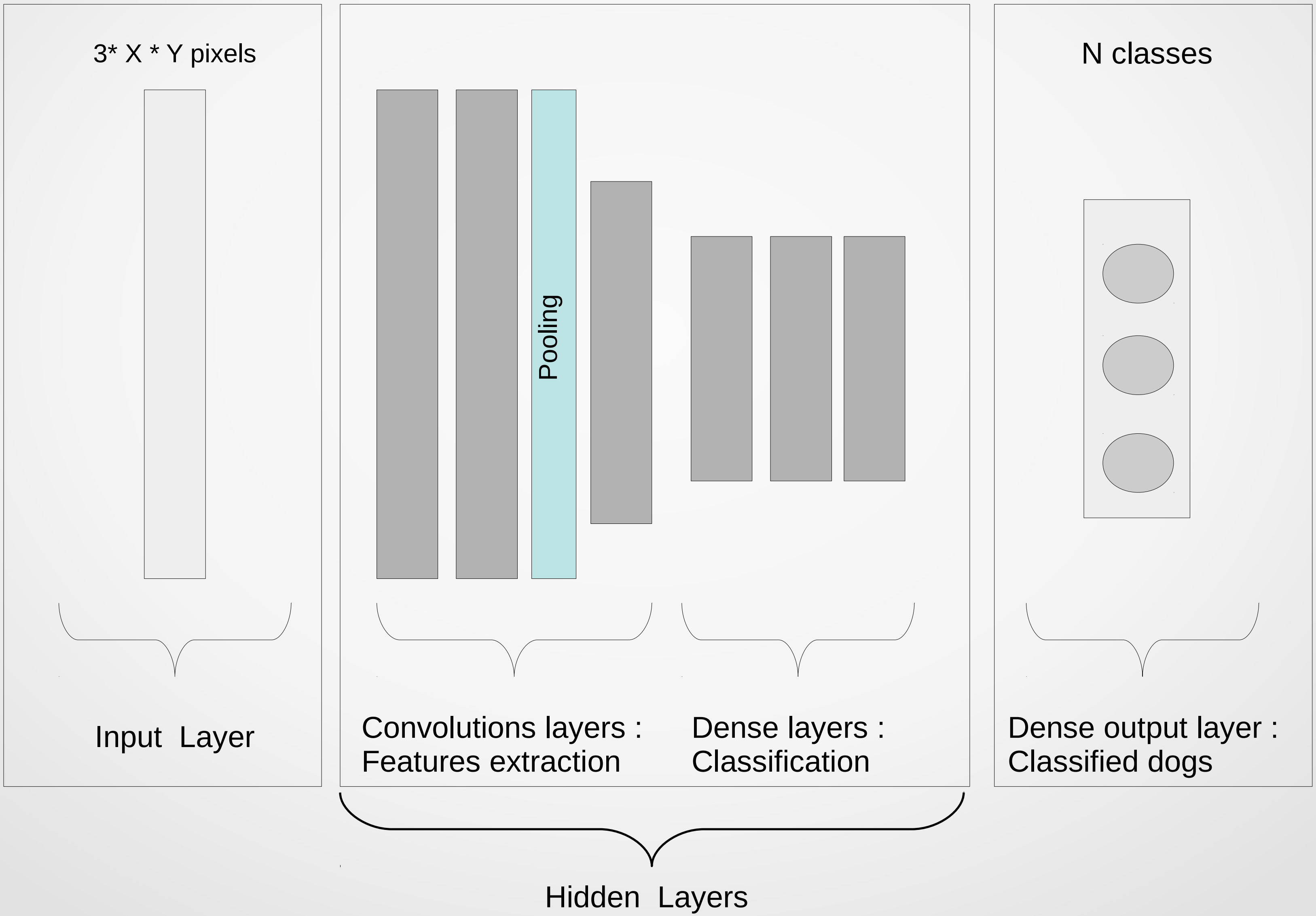
- Over-fitting issue decreased, better decrease of loss function

MLP : overfitting reduction droping neurons

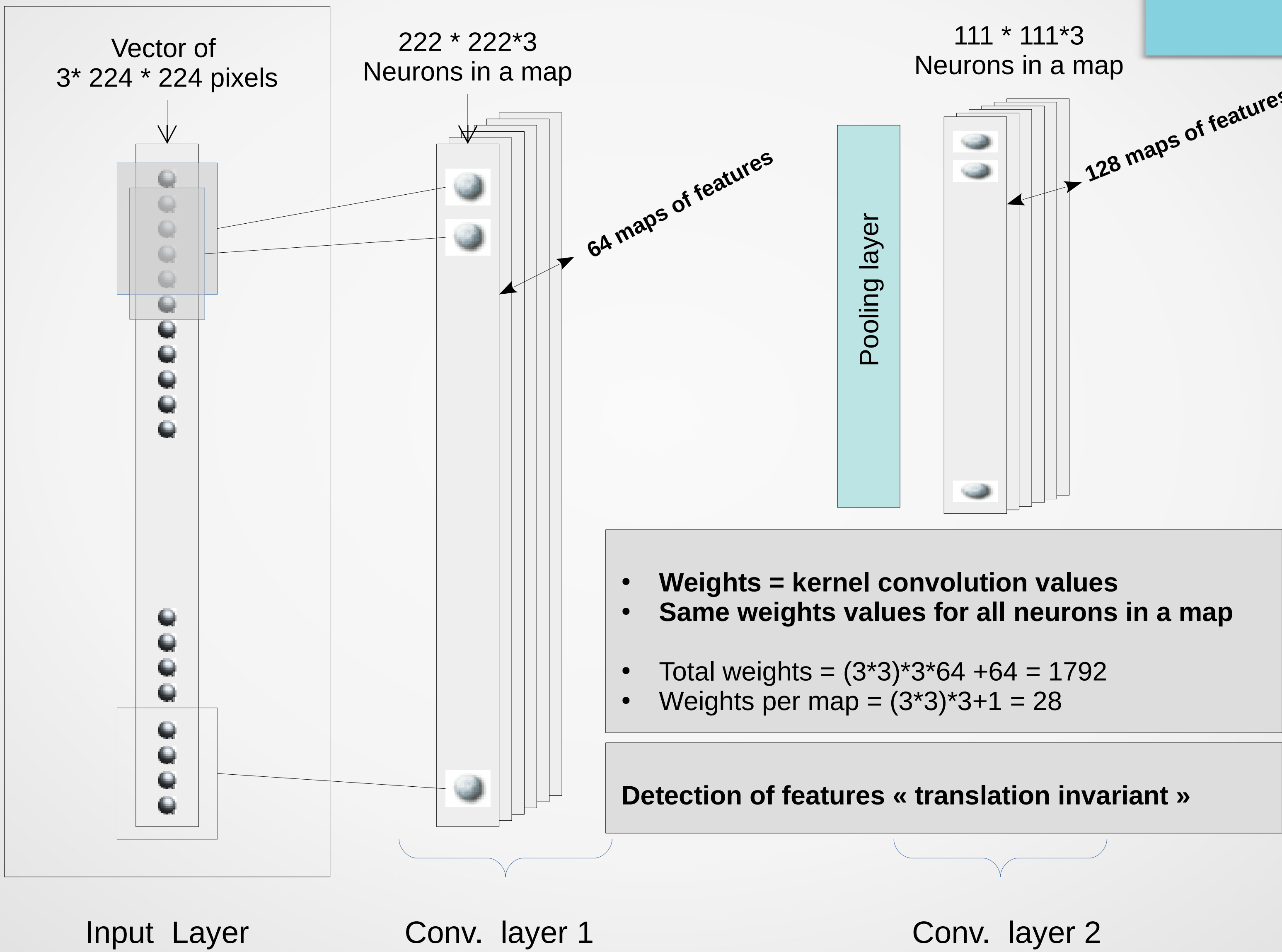
- 461 Images / 3 breeds / 414 trained images / 47 tested images
- Accuracy : → Asymptoticaly 55 %
- Test loss: sticked on training less

- Learning issue ⇒ more iterations

CNN architecture



Convolution layers



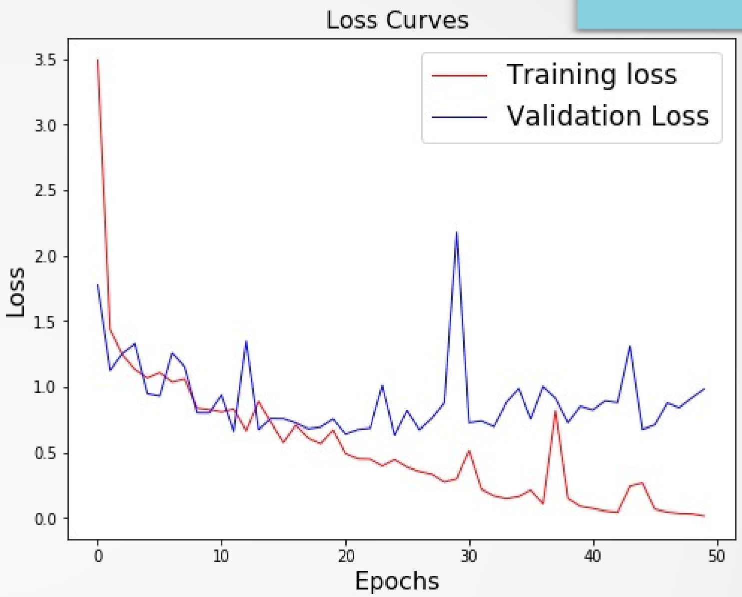
- **Weights = kernel convolution values**
- **Same weights values for all neurons in a map**

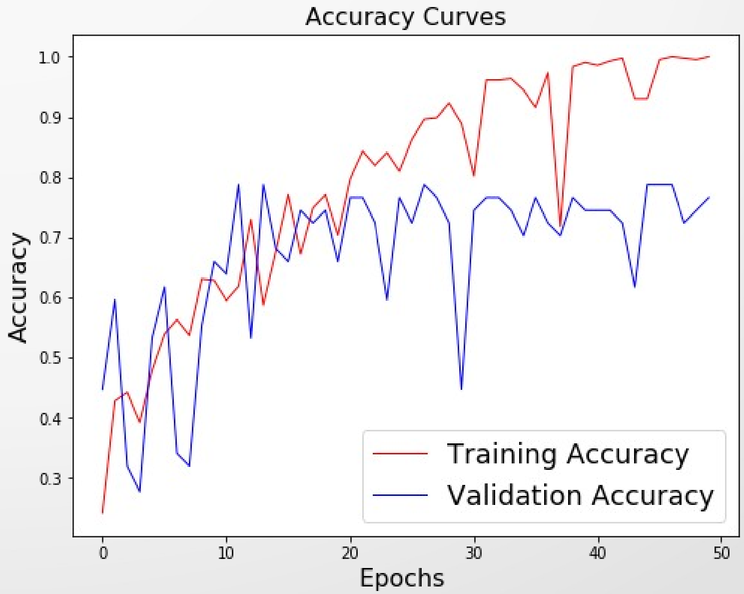
- Total weights = (3*3)*3*64 +64 = 1792
- Weights per map = (3*3)*3+1 = 28

Detection of features « translation invariant »

Built CNN using Keras

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 224, 224, 64)      1792
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 222, 222, 64)      36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 111, 111, 64)      0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 111, 111, 64)      36928
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 109, 109, 64)      36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 54, 54, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 54, 54, 64)        36928
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 52, 52, 64)        36928
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 26, 26, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 43264)             0
_________________________________________________________________
dense_3 (Dense)              (None, 414)               17911710
_________________________________________________________________
dense_4 (Dense)              (None, 414)               171810
=================================================================
Total params: 18,269,952
Trainable params: 18,269,952
Non-trainable params: 0
```





CNN : 8 layers

- 6 convolution layers
- 2 classification layers

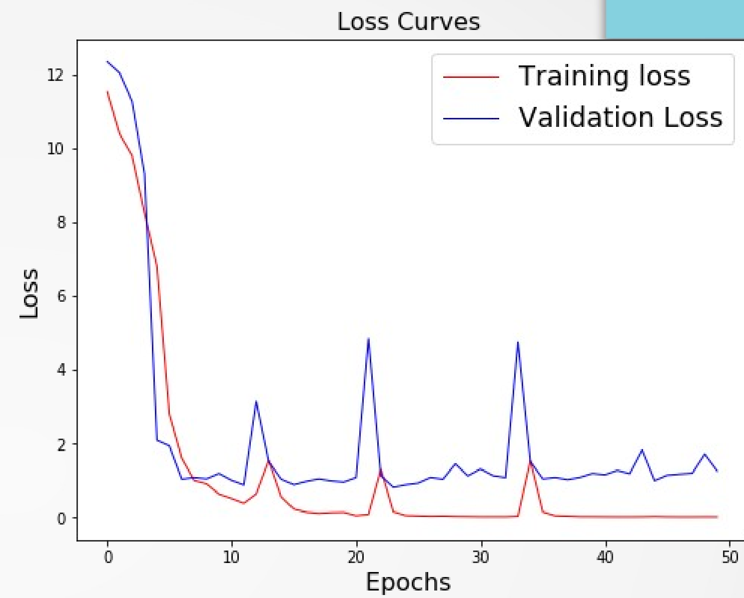
Accuracy : → 70 %

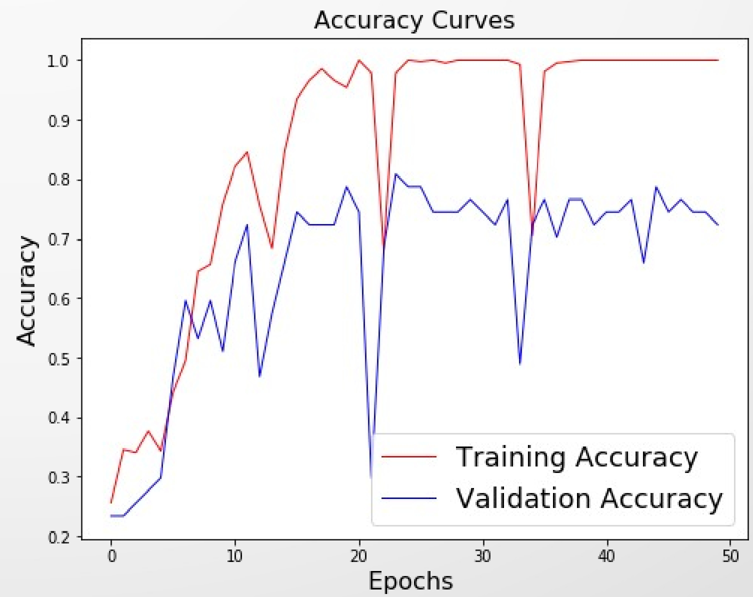
Overfitting issue

Built CNN using Keras adressing overfitting issue

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 64)      1792
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 222, 222, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 111, 111, 64)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 111, 111, 64)      36928
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 109, 109, 64)      36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 54, 54, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 54, 54, 64)        36928
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 52, 52, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 26, 26, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 43264)             0
_________________________________________________________________
dense_1 (Dense)              (None, 414)               17911710
_________________________________________________________________
dropout_1 (Dropout)          (None, 414)               0
_________________________________________________________________
dense_2 (Dense)              (None, 414)               171810
=================================================================
Total params: 18,269,952
Trainable params: 18,269,952
Non-trainable params: 0
```

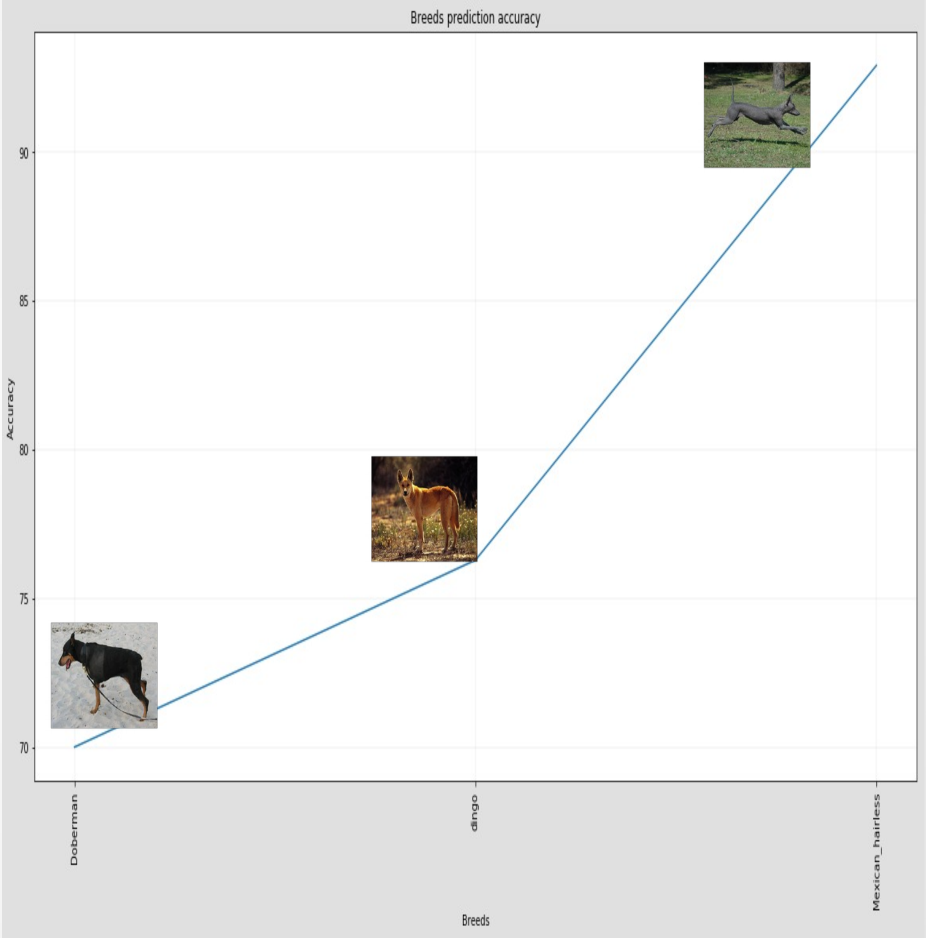
CNN 5 layers + drop layer
Accuracy : → 70 %
Overfitting issue





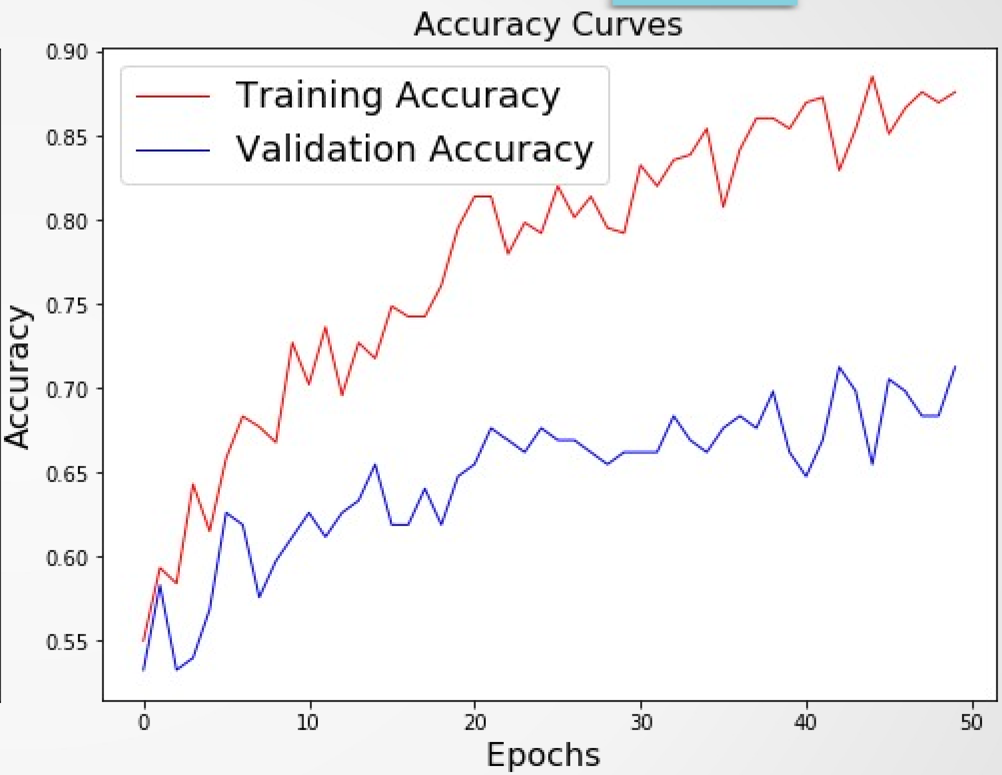
Use of pre-trained VGG16

```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```



Use of VGG16 « off the shelf »
Average Acc : 80 % /Min=70 % / Max : 93 %

Transfer learning : partial tuning of VGG16



- Frozen convolution layers
- 461 Images / 3 breeds / 322 trained images / 139 tested images
- Accuracy : 71 %

Pre-trained VGG16 over Imagenet weights

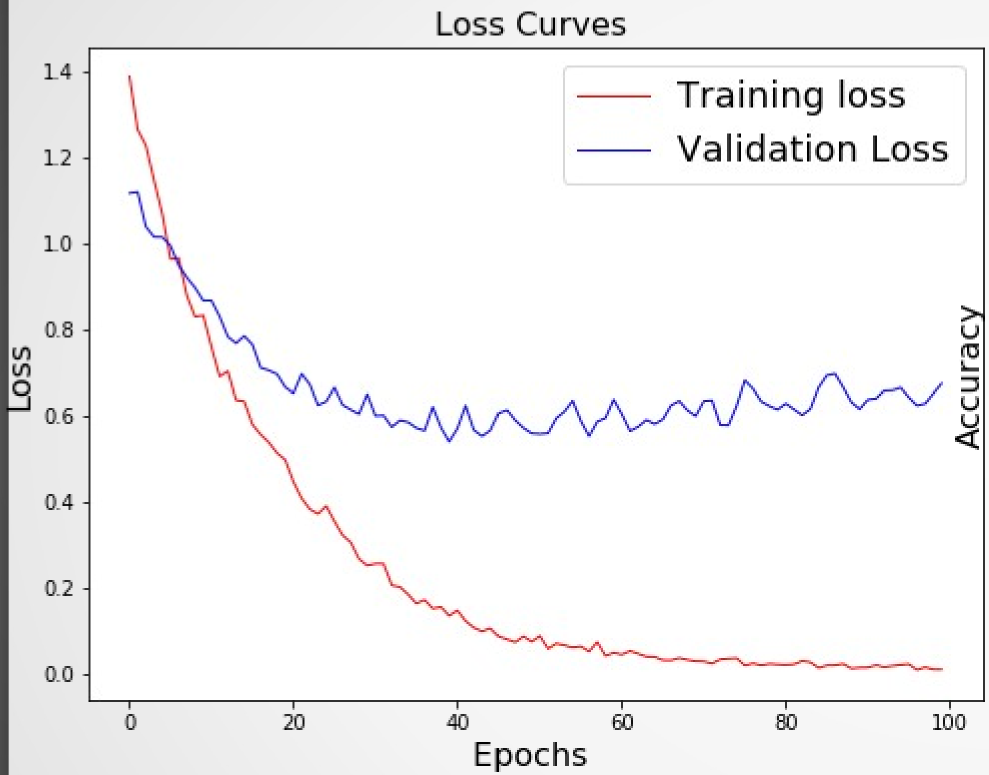
Convolutional layers : 16 layers

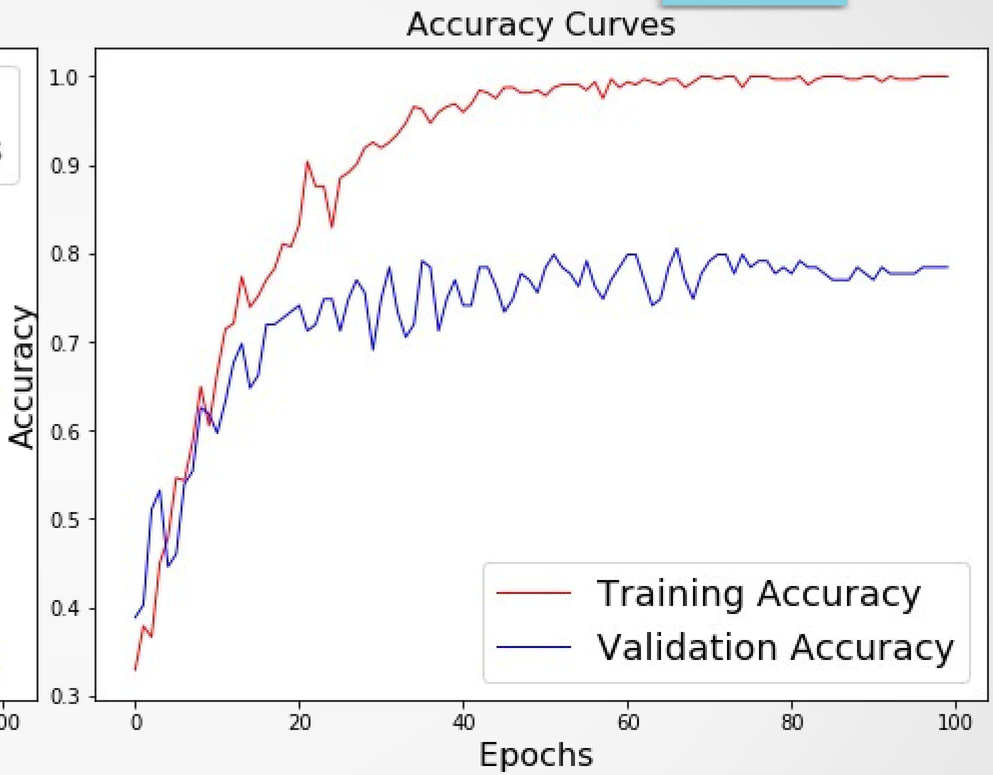
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

Fully connected layers : 2 layers

```
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               12845568
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 1539
=================================================================
Total params: 27,561,795
Trainable params: 27,561,795
Non-trainable params: 0
_________________________________________________________________
```

Transfer learning : fine tuning of VGG16

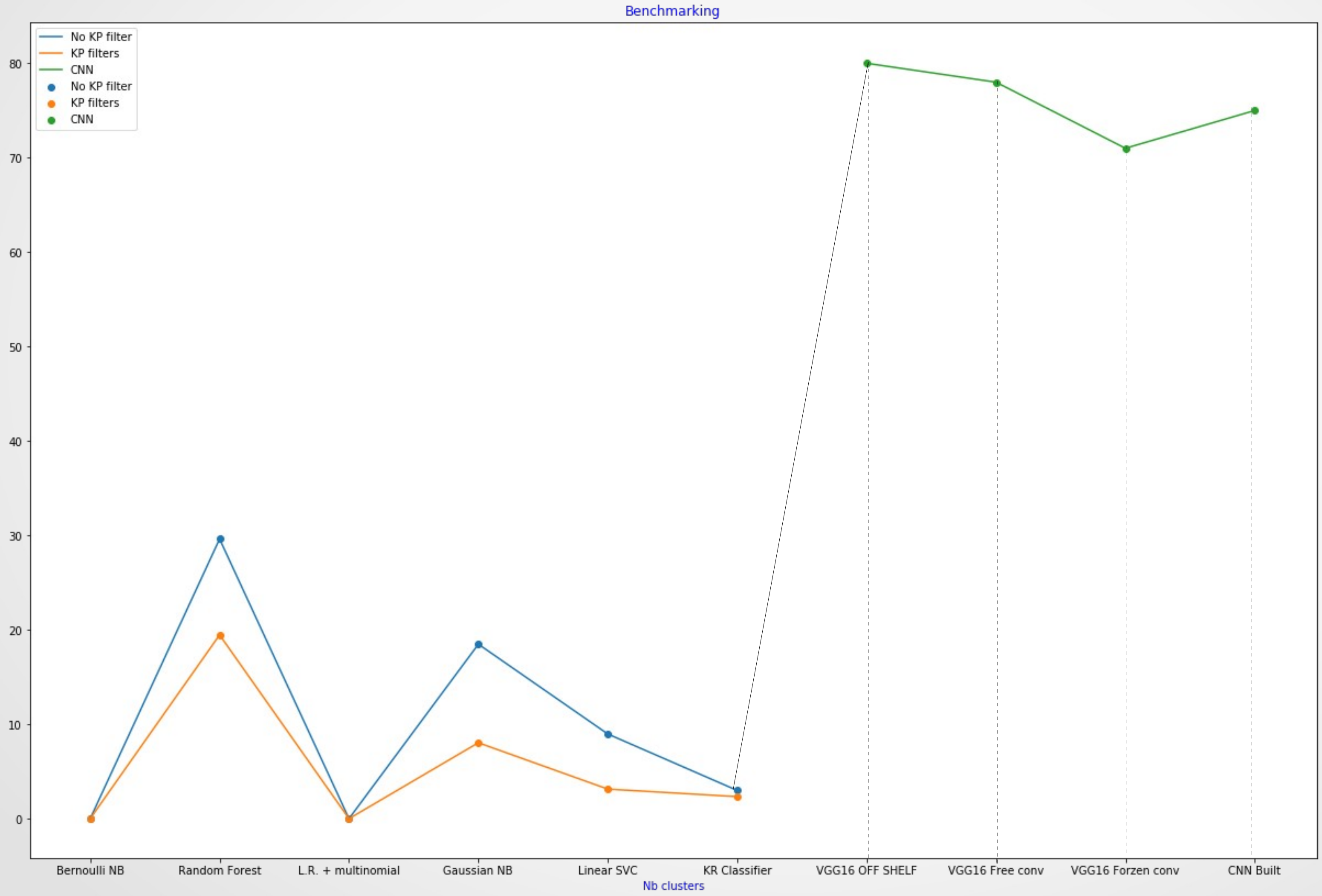




- Free convolution layers
- 461 Images / 3 breeds / 322 trained images / 139 tested images
- Accuracy : 75 %

- Over-fitting issue

Benchmark results over 3 breeds / 461 dogs



Conclusions & perspectives

- M.L. algorithms:
  - SIFT features extraction may be limited
  - Grey scale ⇒ loss of information
  - Failed for multi-class problem : require non linear objective functions for separation
- D.L. :
  - M.L. Perceptron provides better results than any M.L. algorithm
  - CNN provide better results then M.L. perceptron
  - Building efficient CNN requires huges ressources for RAM and computation
  - Using pre-trained CNN require computation resources
  - Cloud computing along with pre-trained C.N. may provide accuracy > 90 %
  - More training data ⇒ increase train accuracy & increase or not test accuracy.
  - Miss of training data : use of data augmentation
  - Apply noise filters as input of CNN

Annexe 1 : source files organization

Notebooks :

- P7_DataAnalysis.ipynb : data analysis
- P7_DataBreed.ipynb : used for building oDataBreed component
- P7_ClusteringDescriptors.ipynb : used to study clustering of descriptors
- P7_FilterExploration.ipynb : used to explore filters
- P7_KerasCNN.ipynb : used to build CNN with Keras
- P7_KerasMLP.ipynb : used to build multi-layers « perceptron »
- P7_Keras_Pretrained.ipynb : used to build pre-trained CNN
- P7_KerasPretrained_FullTuning.ipynb : used to buid pre-trained CNN
- P7_TestDataBreed.ipynb : used to test oDataBreed with multiple classifiers

Python source files: src

- P7_DataBreed.py : implementation for using multiple classifiers
- p7_util.py : functions for P7 project
- p6_util.py : functions issued from P6 project
- p6_util_plot.py : functions for plot issued from P6 project
- p5_util.py : functions issued from P5 project
- p3_util.py : functions issued from P3 project

Report: report

- Openclassrooms_ParcoursDatascientist_P7-V1.pdf : project slides