

Parcours DataScientist : projet 7

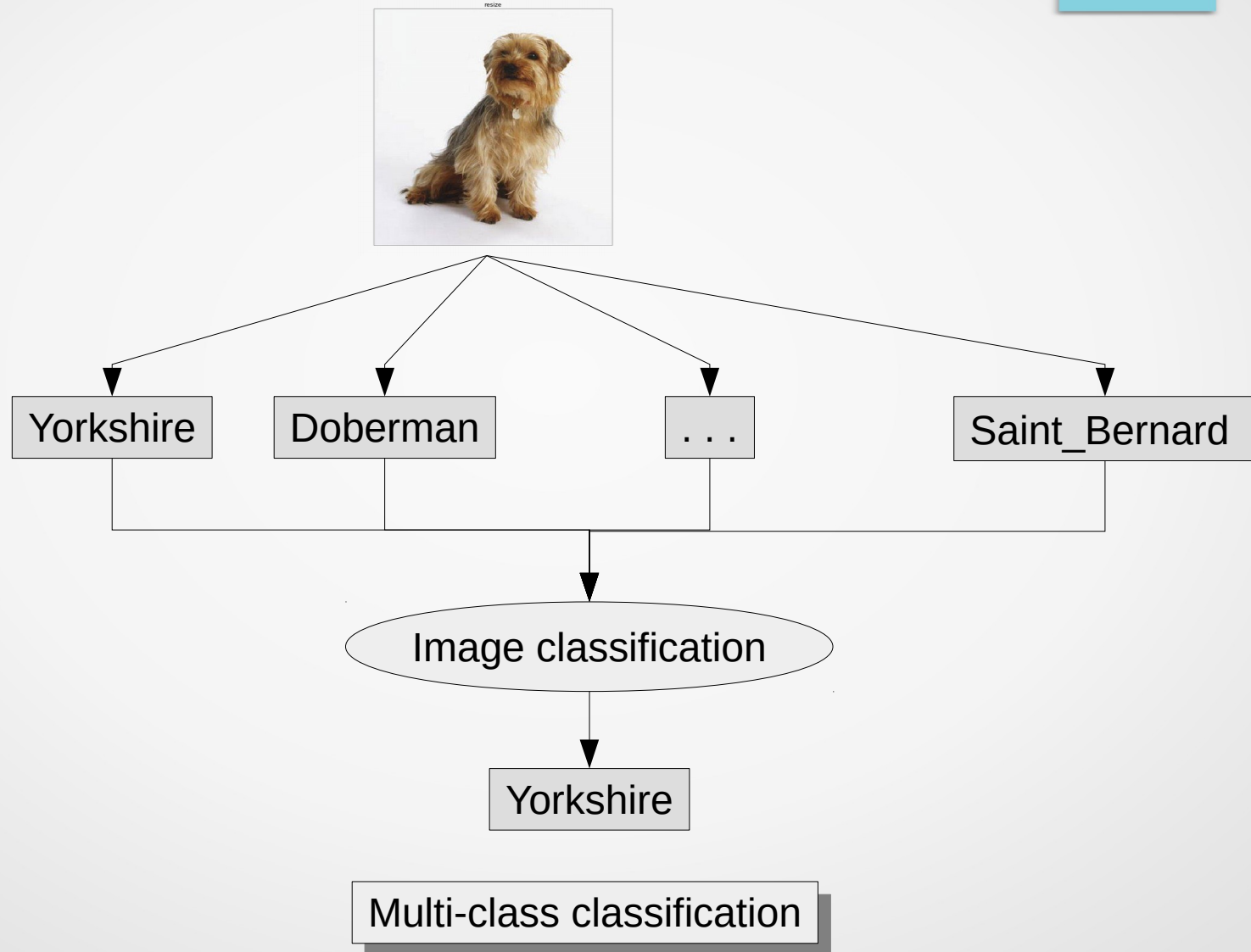


Indexation automatique d'images

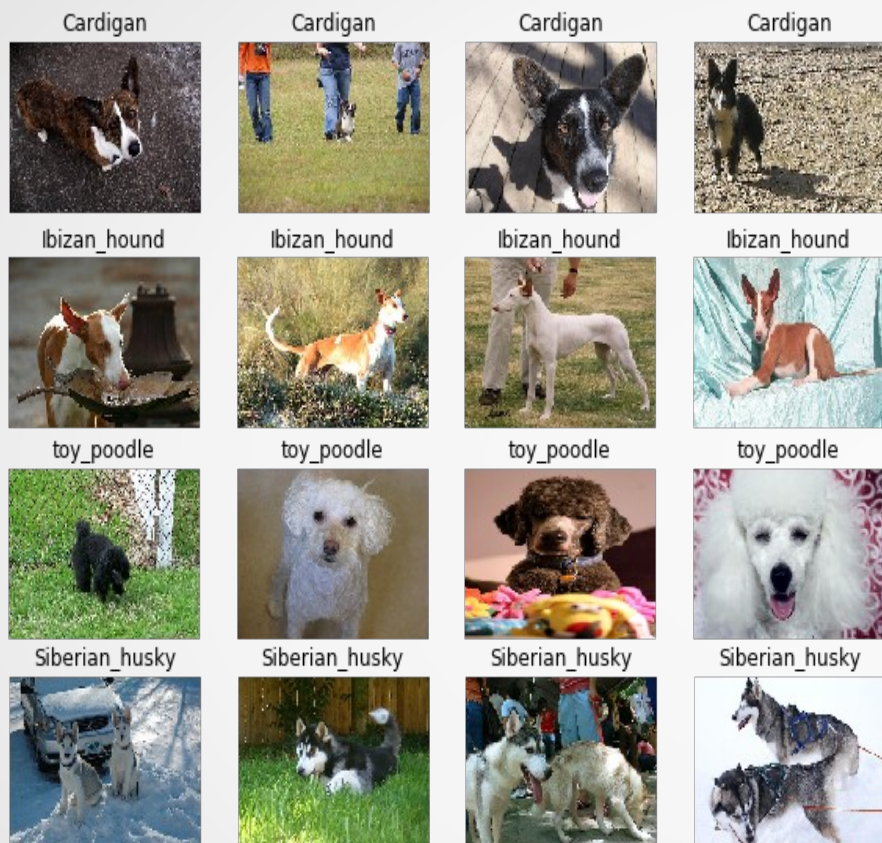
issues de la banque d'images Stanford Dogs Dataset

François BANGUI

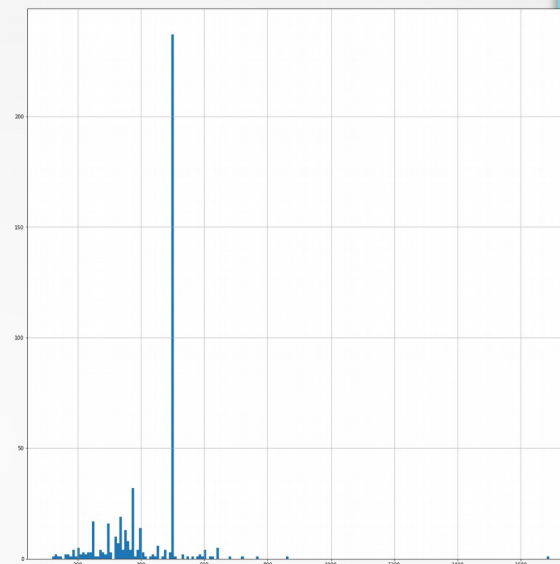
Problem classification type



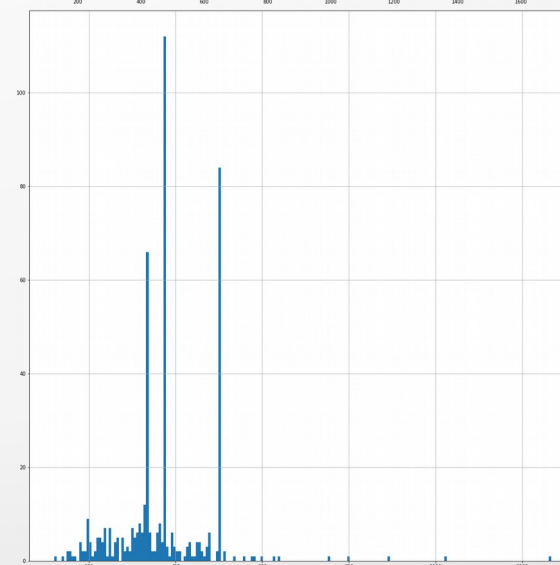
Stanford dogs database



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

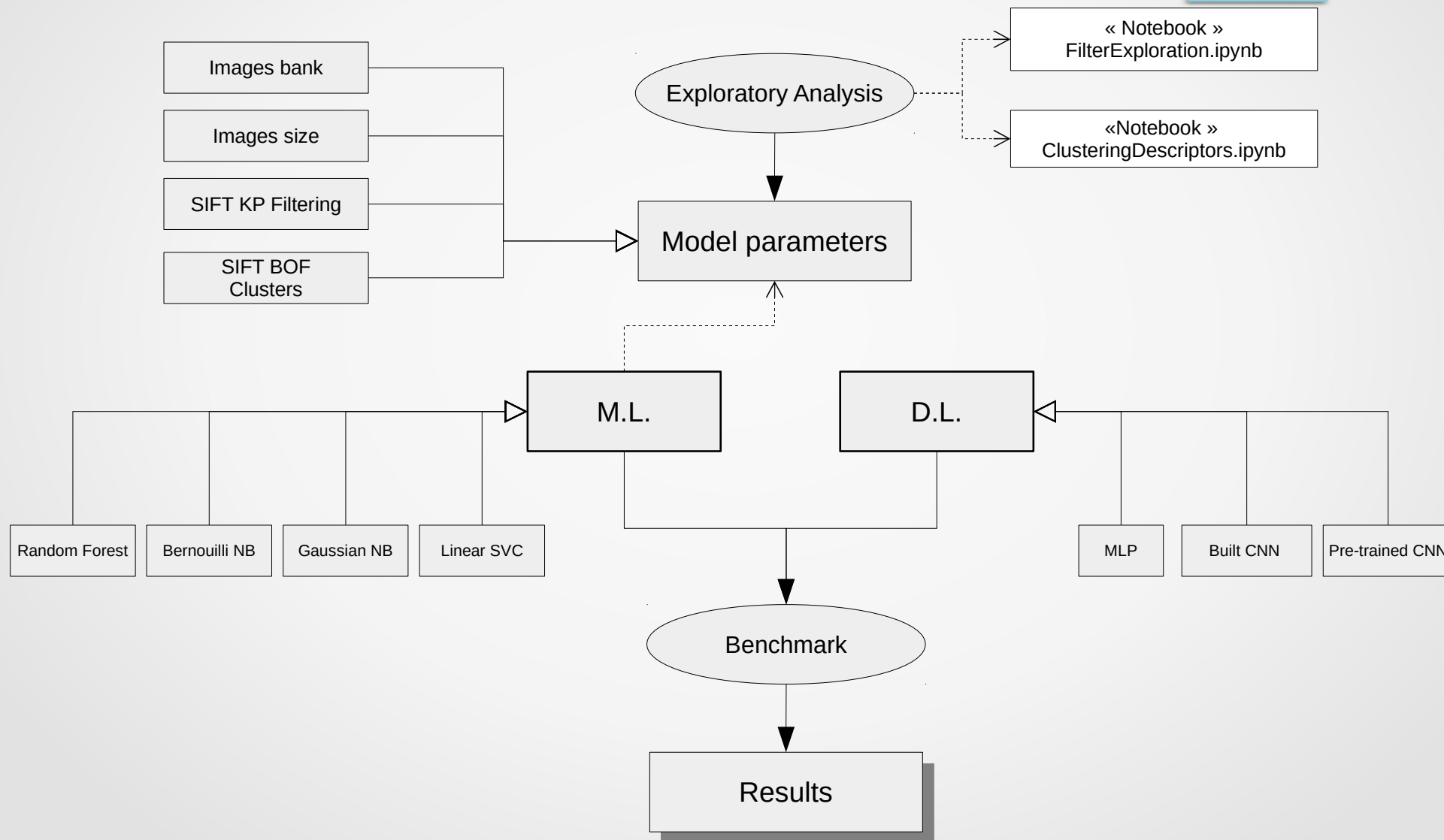


$200 \leq X \leq 1700$



$100 \leq Y \leq 1300$

M.L vs D.L : Global benchmark scheme

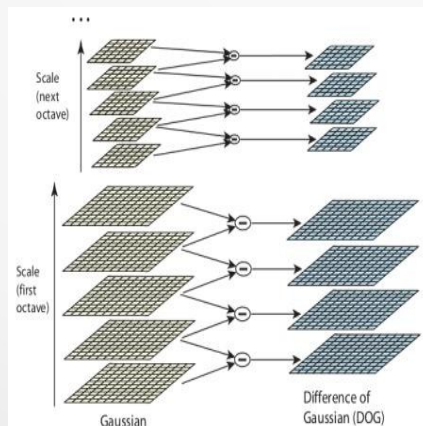


M.L. : Features extraction with SIFT



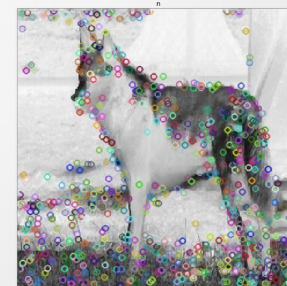
Key Points Extraction

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



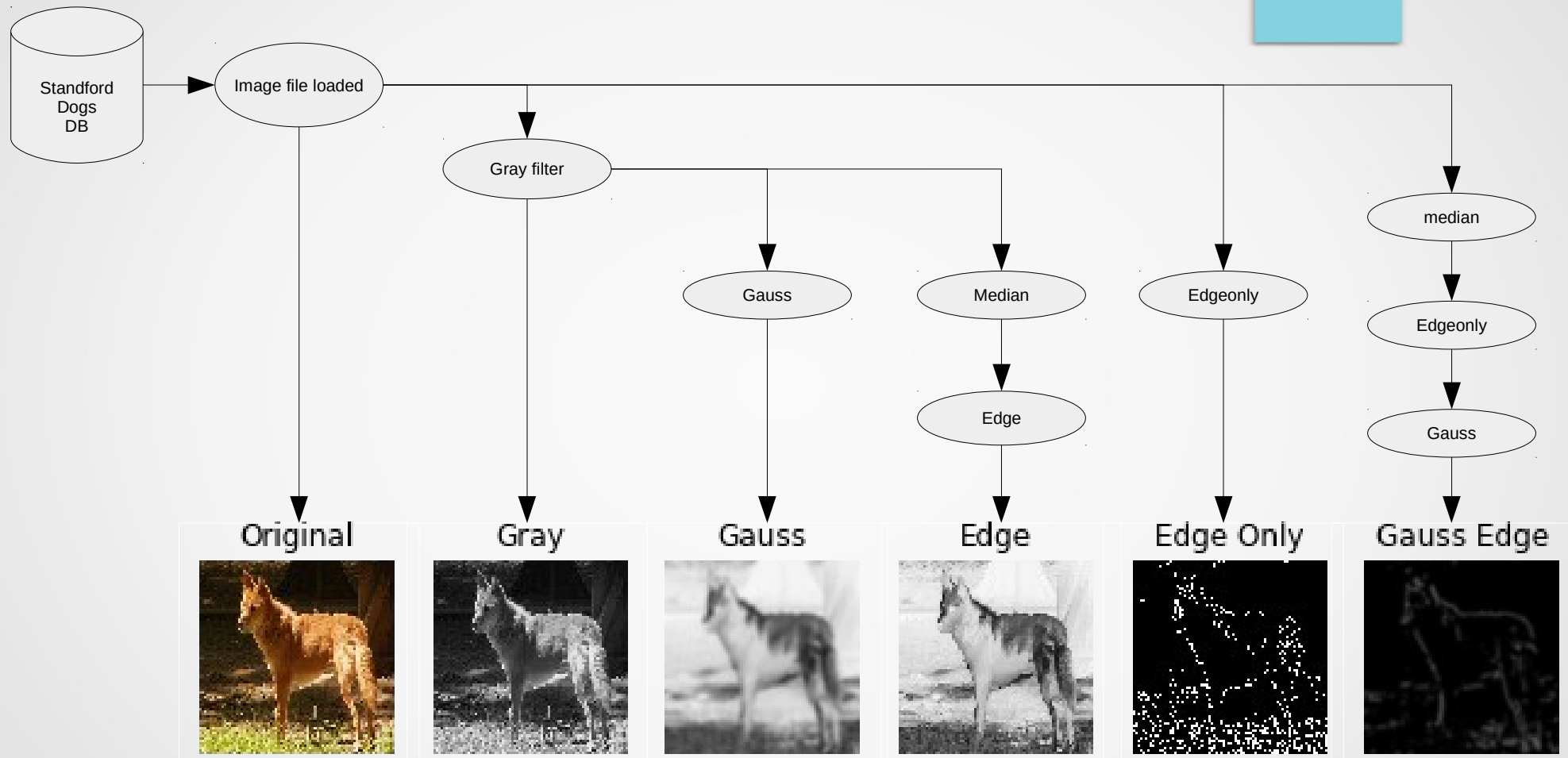
Key Points

Geometric invariant properties
Scaled invariant
Vector 128



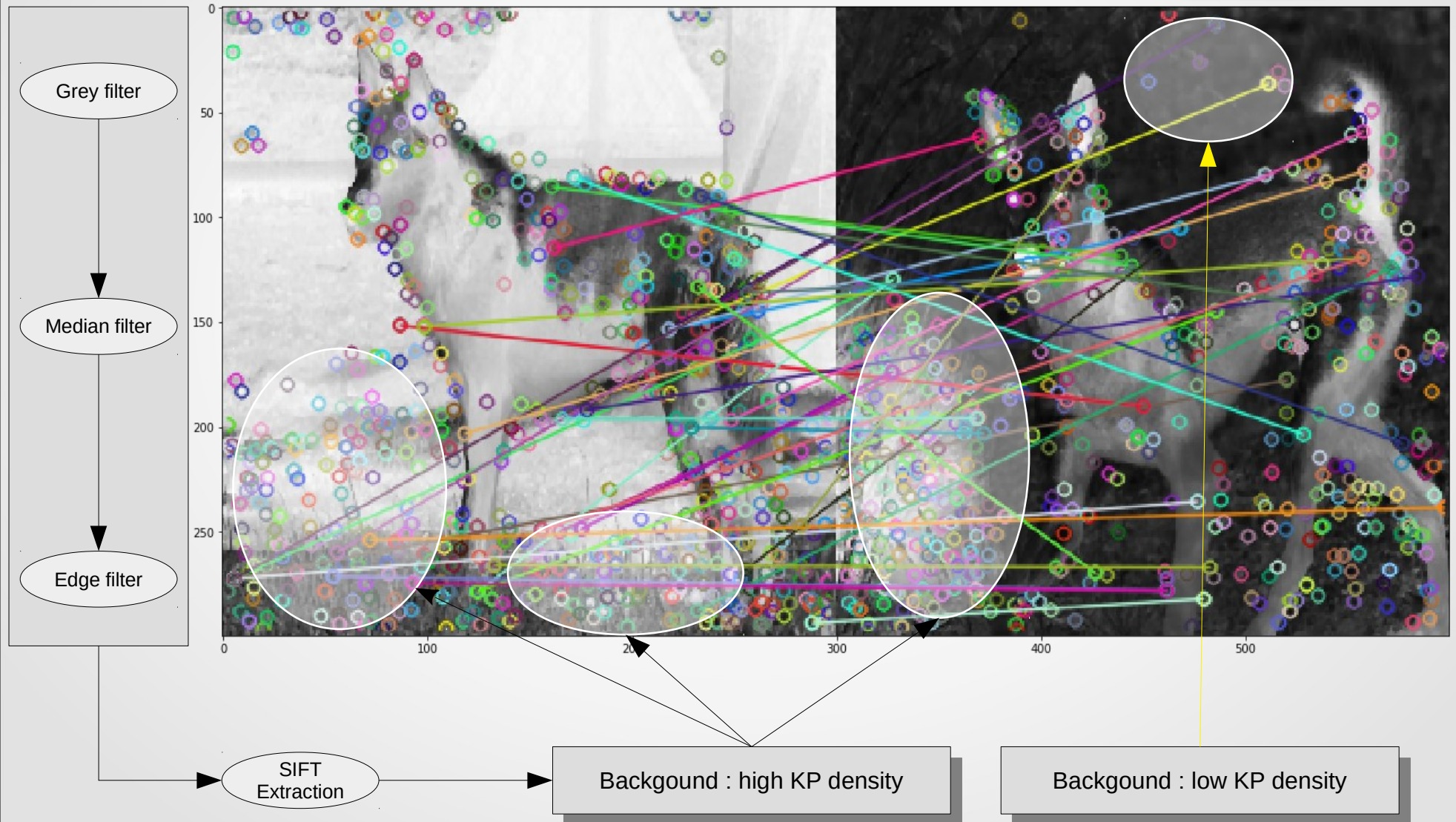
2 different vectors → Same KP

M.L. : Filters exploration



Aim : Most efficient SIFT extraction

M.L. : KP extraction efficiency

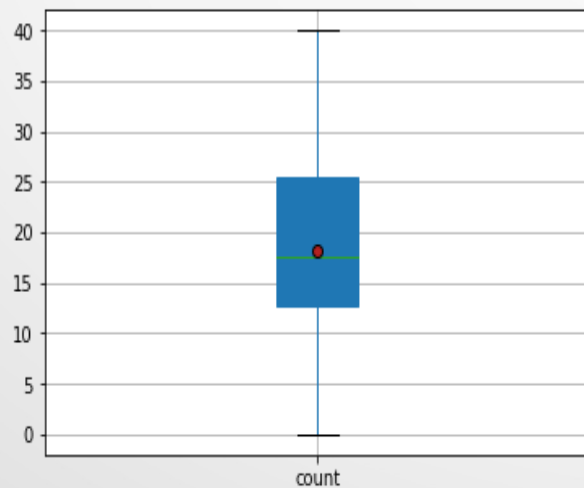
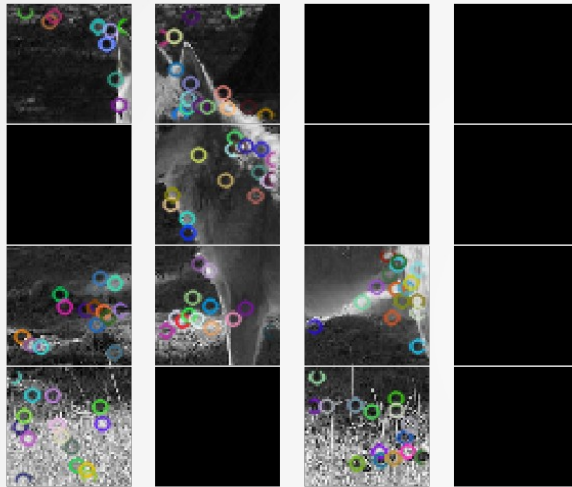


SIFT KP density per splitted image

Gray : KP = 341 \rightarrow 290 / -15 %

Grey filter

Median filter

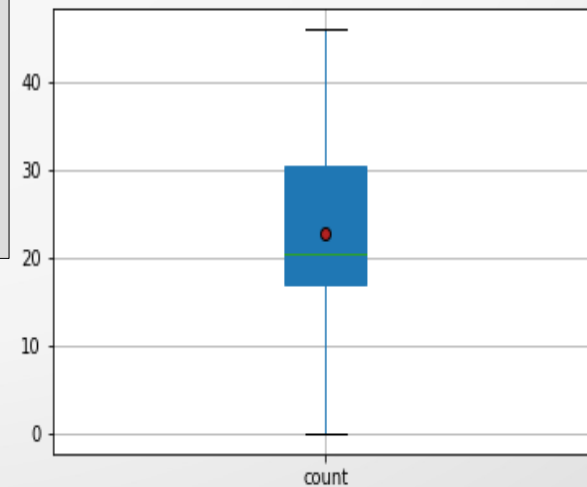
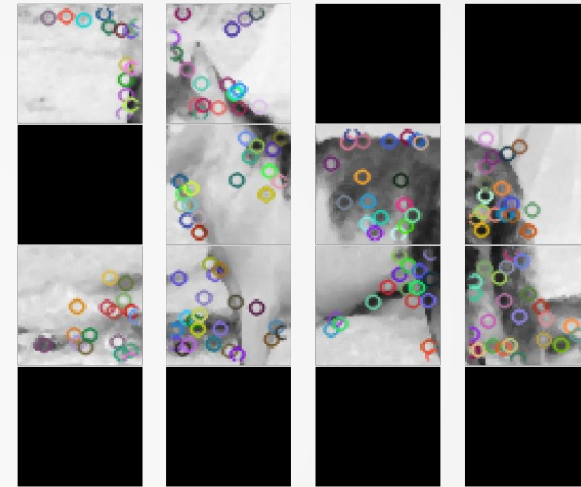


Edge: KP = 450 \rightarrow 366 / -20 %

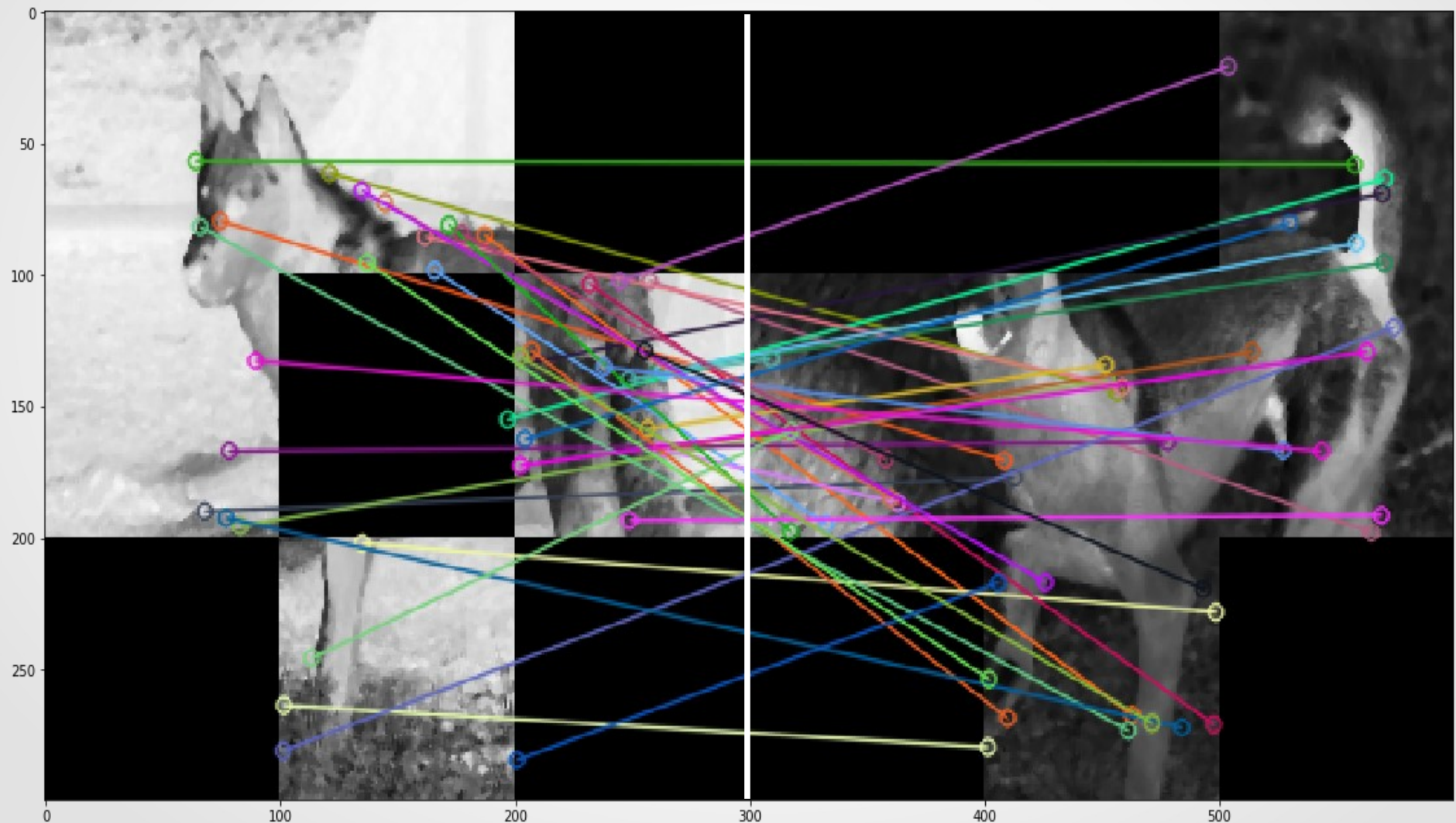
Grey filter

Median filter

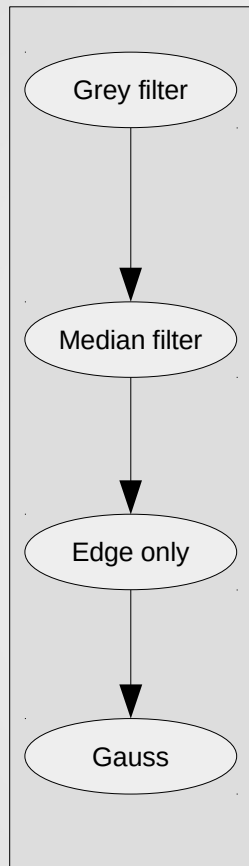
Edge filter



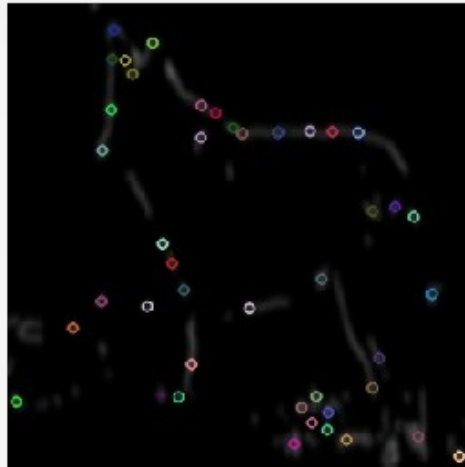
Filtered SIFT key points matching



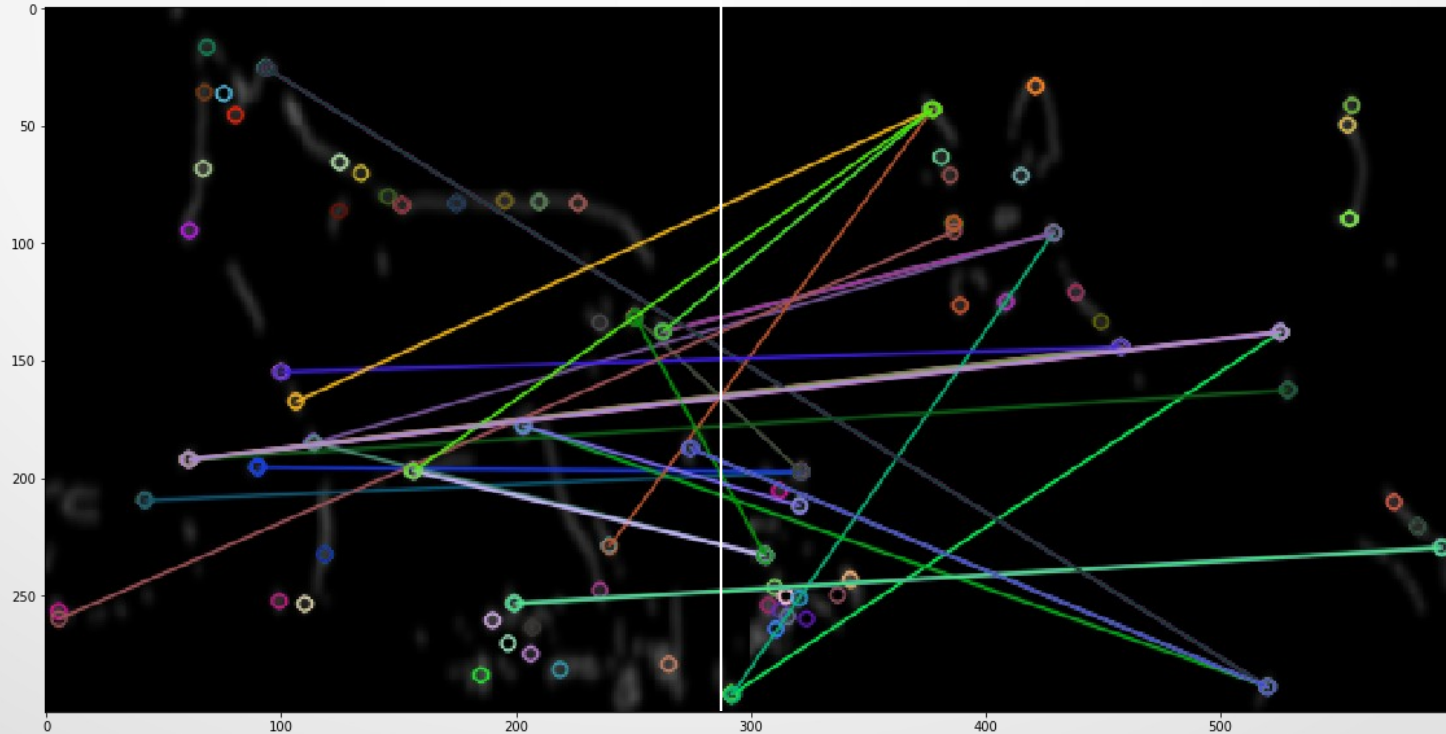
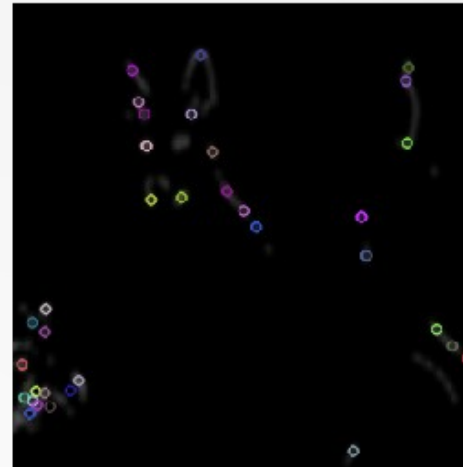
Filtered SIFT key points matching



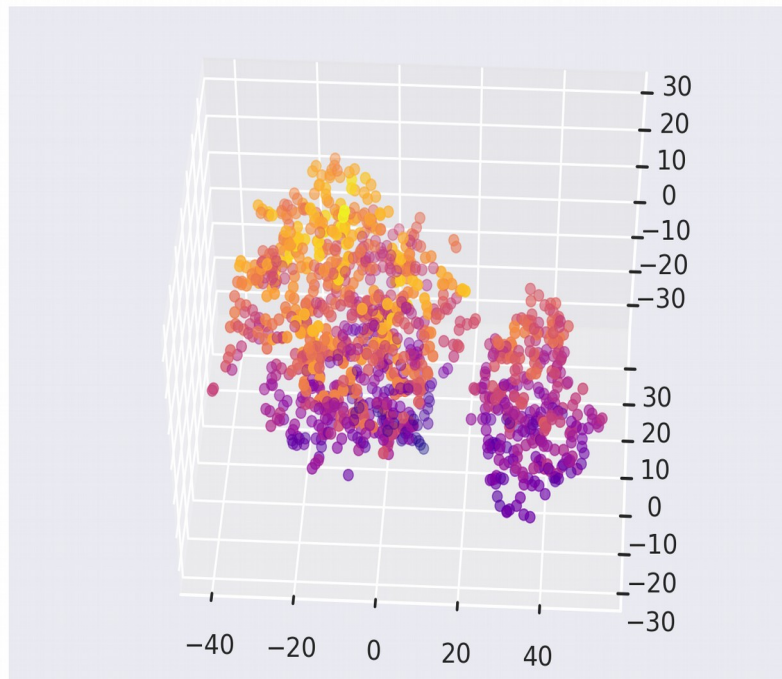
n02094433_3947



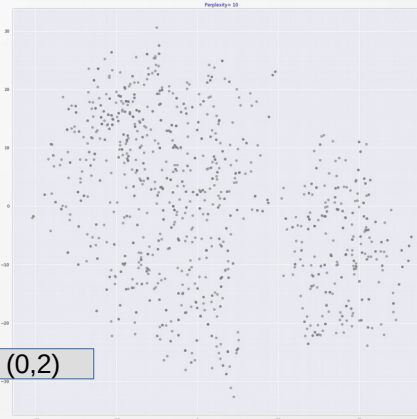
n02094433_8535



SIFT descriptors t-SNE

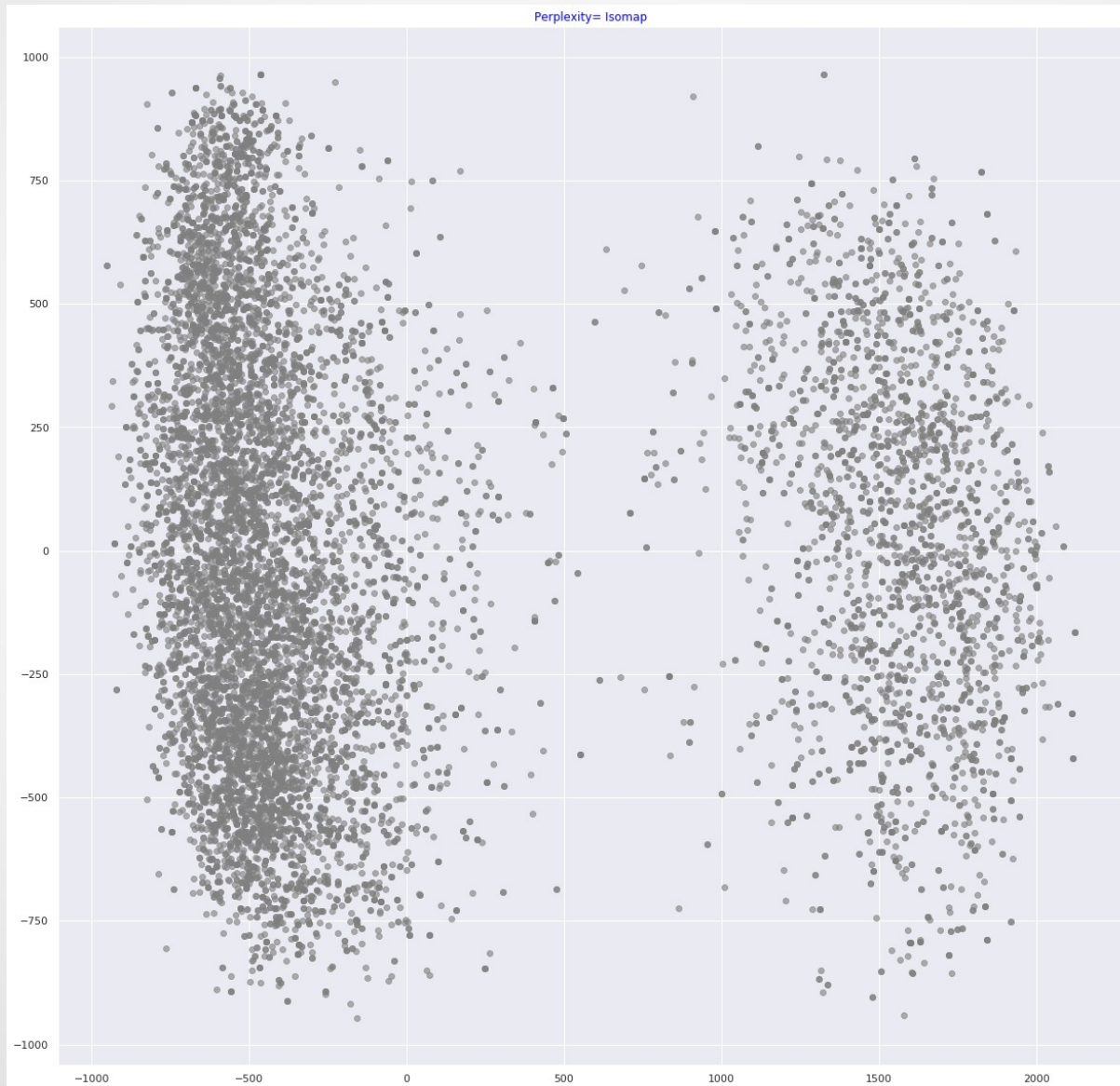


(0,1)



(0,2)

ISOMAP dimension reduction



Bag Of Features



2 different vectors → Same KP
 $N \times 128$

Clustering



$N \times C$

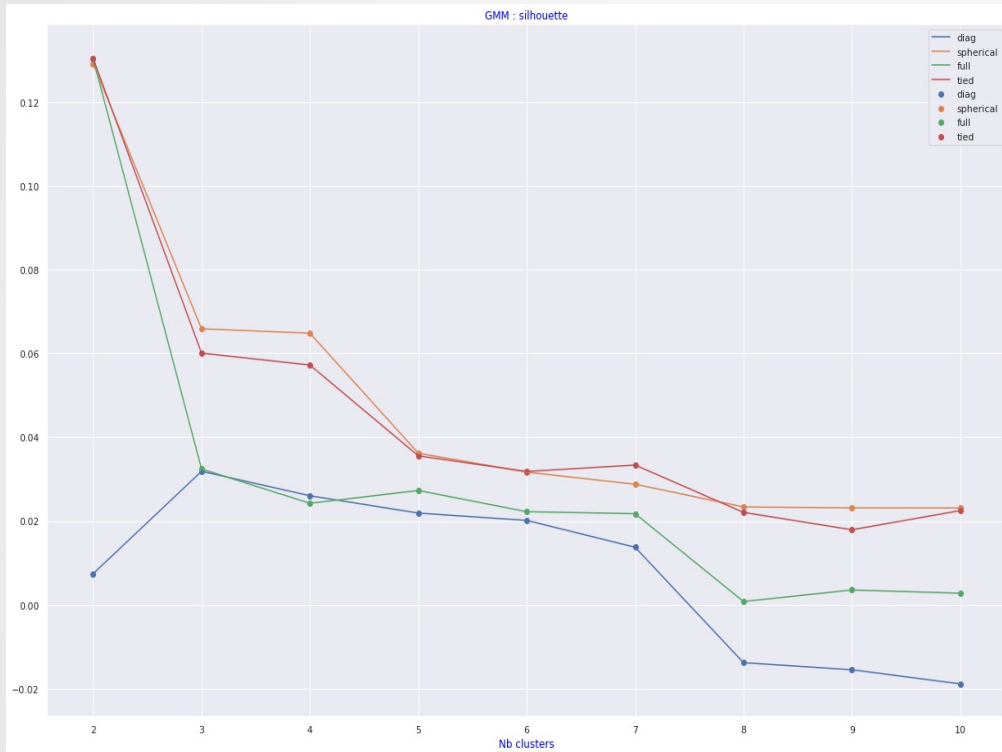
KP descriptors

Clusters

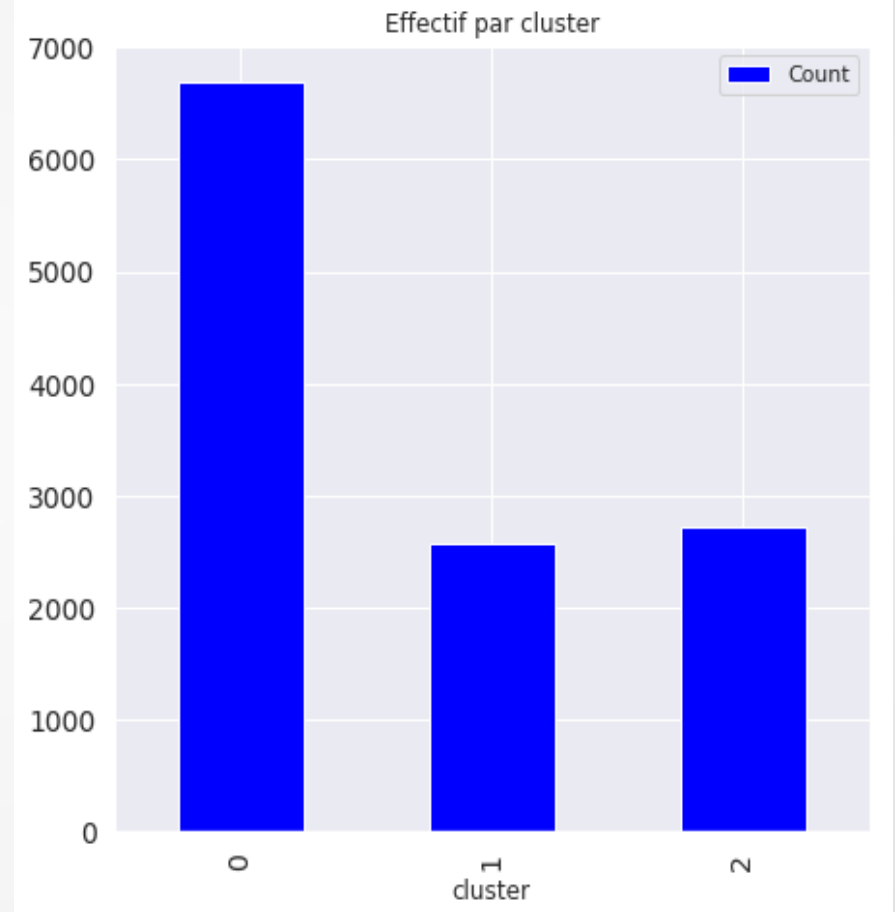
Building histogram

	Clusters		
Desc	0	1	2
	6689	2722	2570

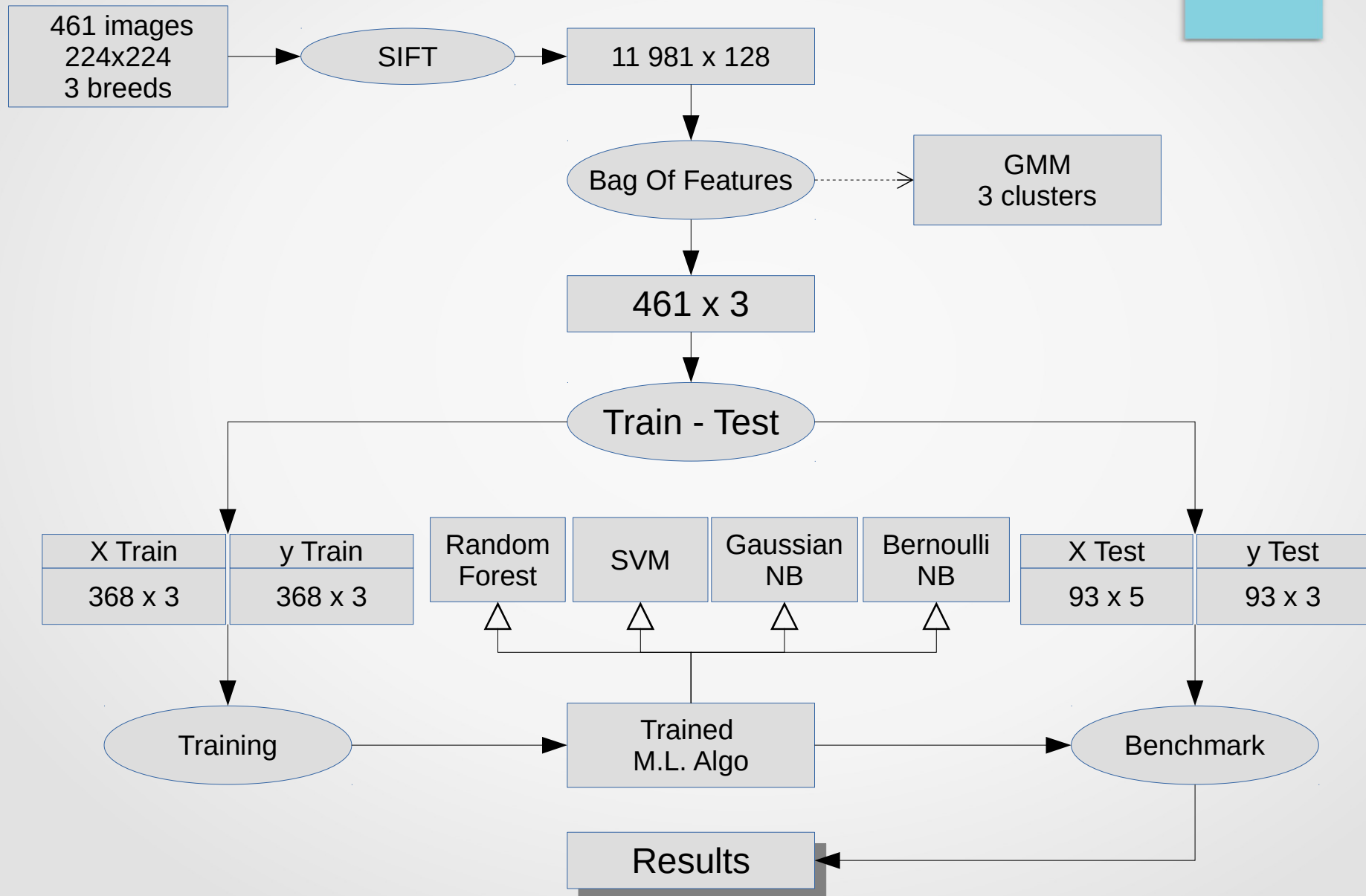
GMM clustering



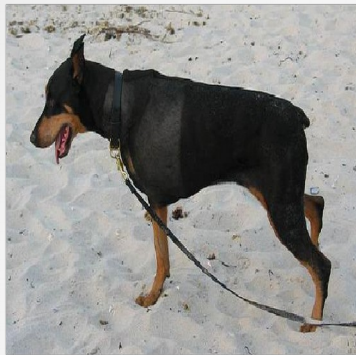
cluster	Count
0	6689
2	2722
1	2570



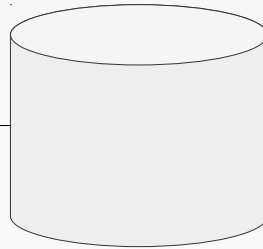
M.L. : model building



M.L. : binary classification



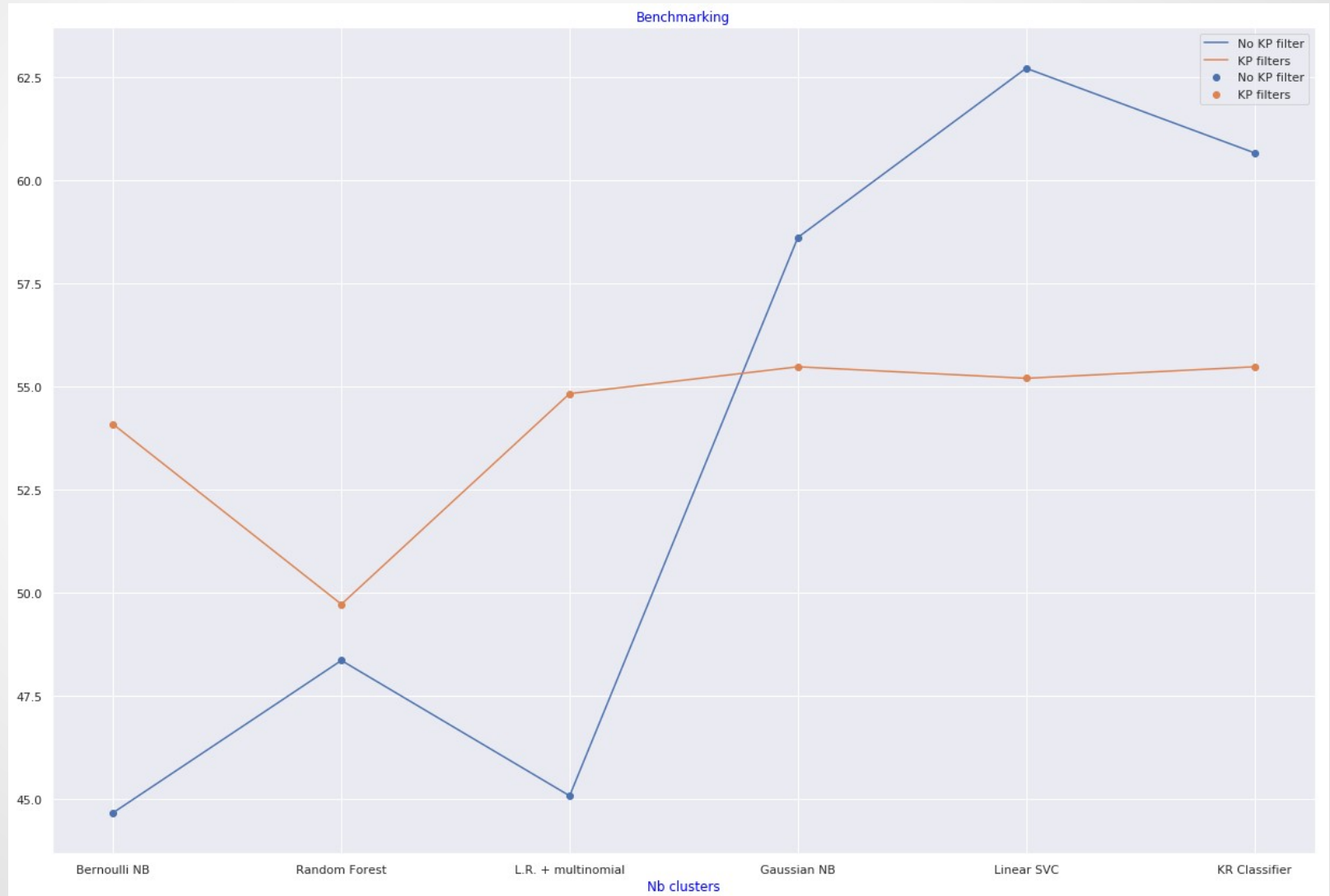
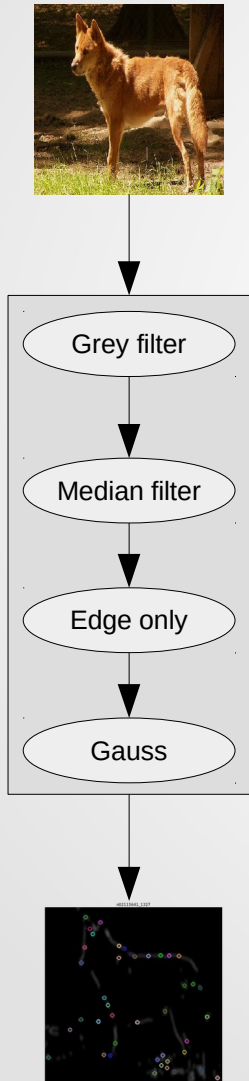
Doberman



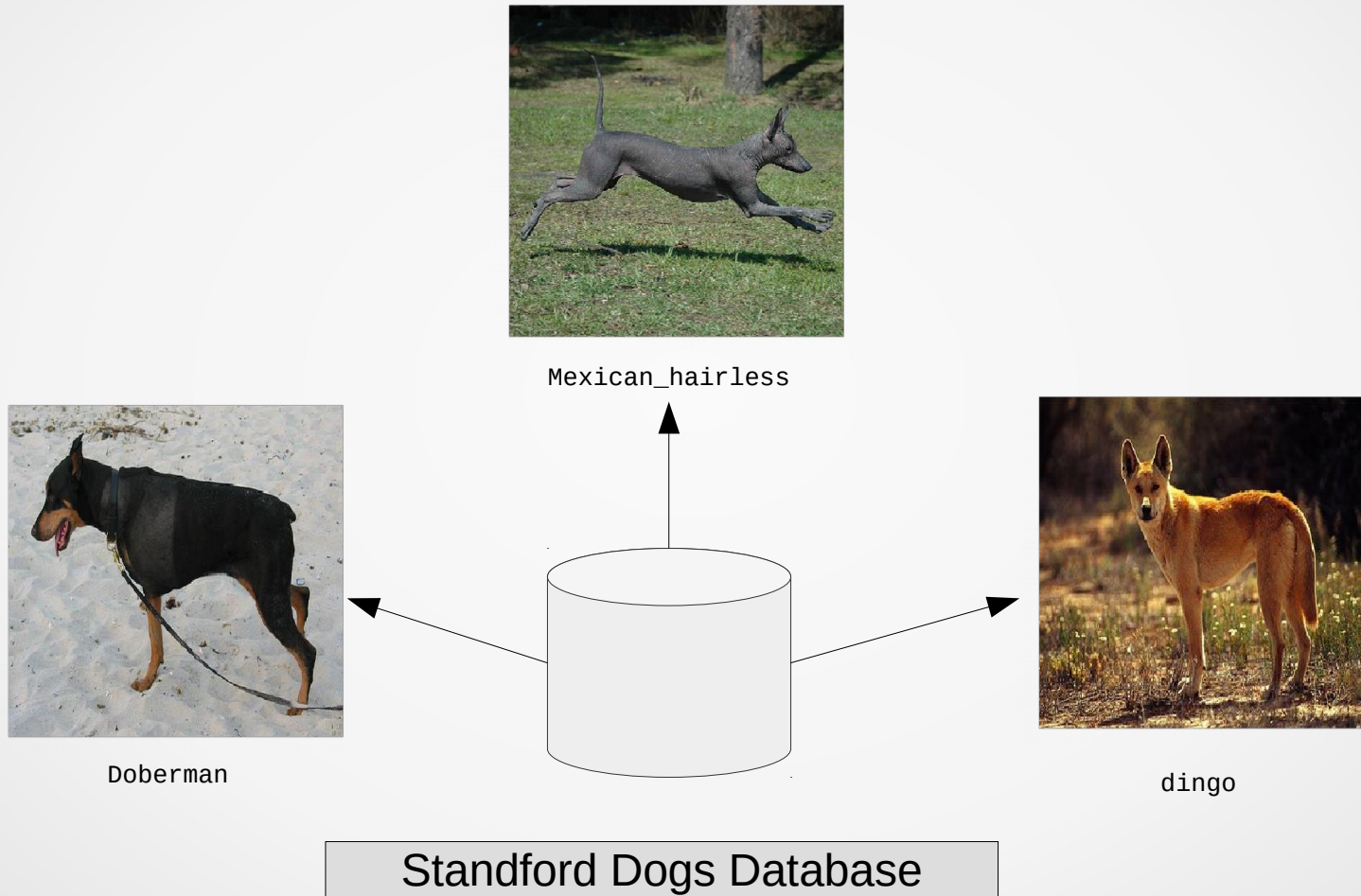
dingo

Stanford Dogs Database

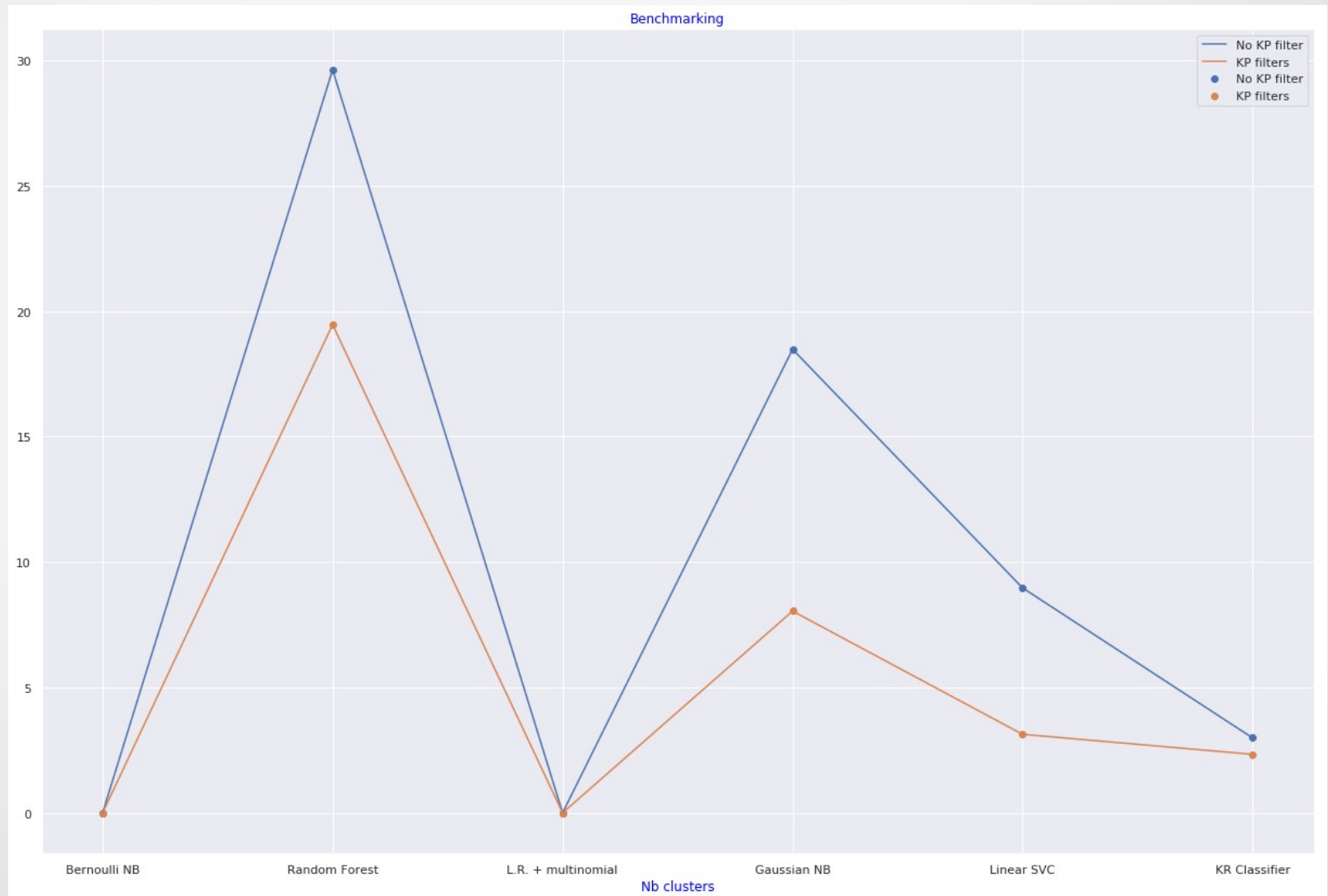
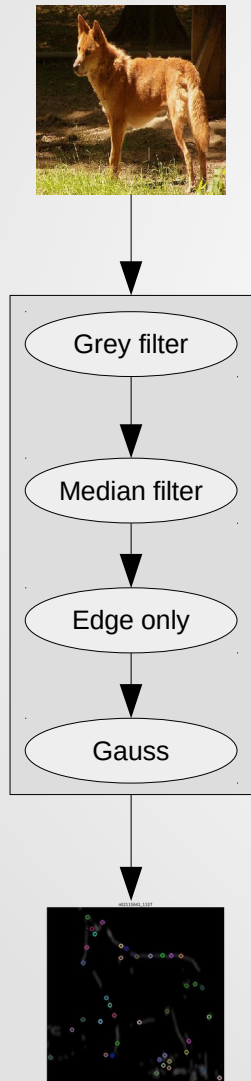
M.L. : binary classification : no KP filter



3 breeds from Stanford dogs database

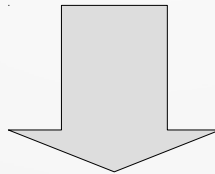


M.L. : benchamrk of 3 breeds



M.L. conclusions

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



Neural Network experience

Neural Networks

Multi-layer perceptron

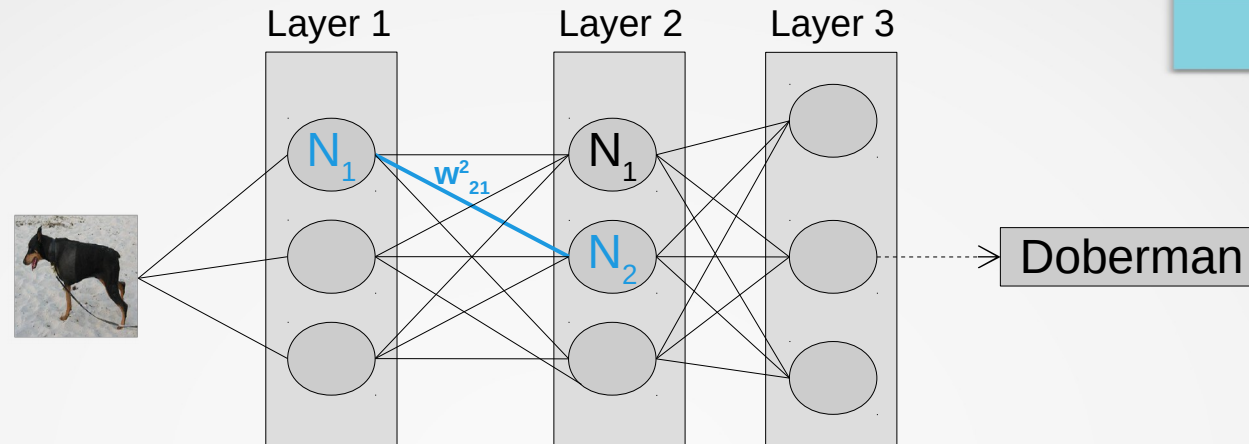
Built CNN

Pre-trained CNN

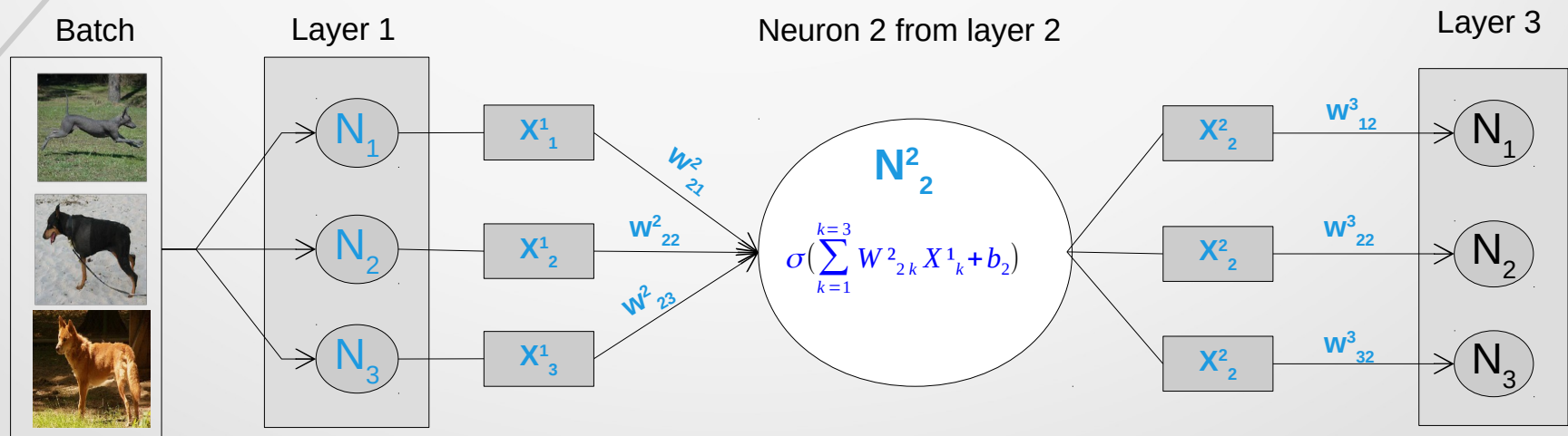
CNN Transfer learning : partial tuning

CNN Transfer learning : fine-learning

M.L.P & Softmax: Architecture feed forward



- More layers \Rightarrow More sophisticated classification
- w_{21}^2 : weight belonging to neuron 2 into layer 2 coming from neuron 1 into layer 1
- Same weights values on all links issued from any N_i
- Training with batch of
- Multi-class classification \Rightarrow Use of Softmax



N.N. : background propagation

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

1. Last layer : Compute error on thanks to Cost function and activation function Act :

- $$\varepsilon^{Last} = \frac{\partial Cost}{\partial X^{Last}} * Act'((X * \omega + b)^{Last}) \quad \text{With} \quad Cost(w, b) = \sum_{node} p \log_2(p) - \sum_{node} q \log_2(p) \text{ where } q = q(w, b)$$

2. Compute error on Layer L thanks to error on layer Layer L+1

- $$\varepsilon^{layer} = (\omega^{layer+1})^T * \varepsilon^{layer+1} \odot Act'((X * \omega + b)^{layer}) \text{ where } X \odot Y \text{ is Hadamard product vector between } X \text{ with } Y$$

3. Compute gradient components for Cost function, for any neuron J into any layer

- $$\frac{\partial Cost}{\partial b_j^{layer}} = \varepsilon_j^{layer}$$
- $$\text{For any neuron } j \in \text{any layer} : \frac{\partial Cost}{\partial \omega_{jk}^{layer}} = X_k^{layer-1} \varepsilon_j^{layer} = (\nabla_{\omega} Cost)_{jk}$$

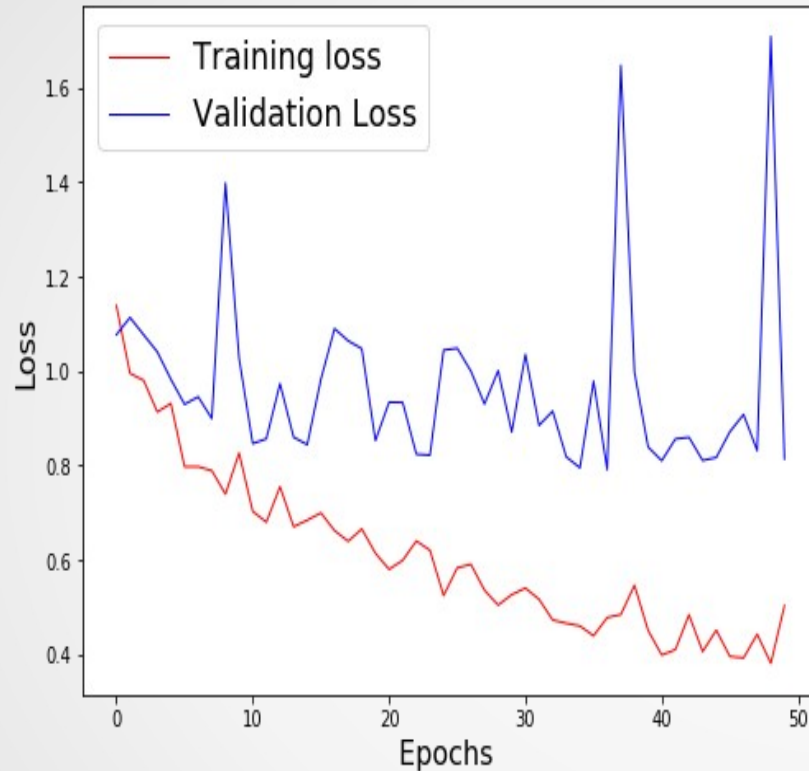
} \Leftarrow average over all layer nodes

4. Thanks to cost gradient, update weight from neuron J in any layer :

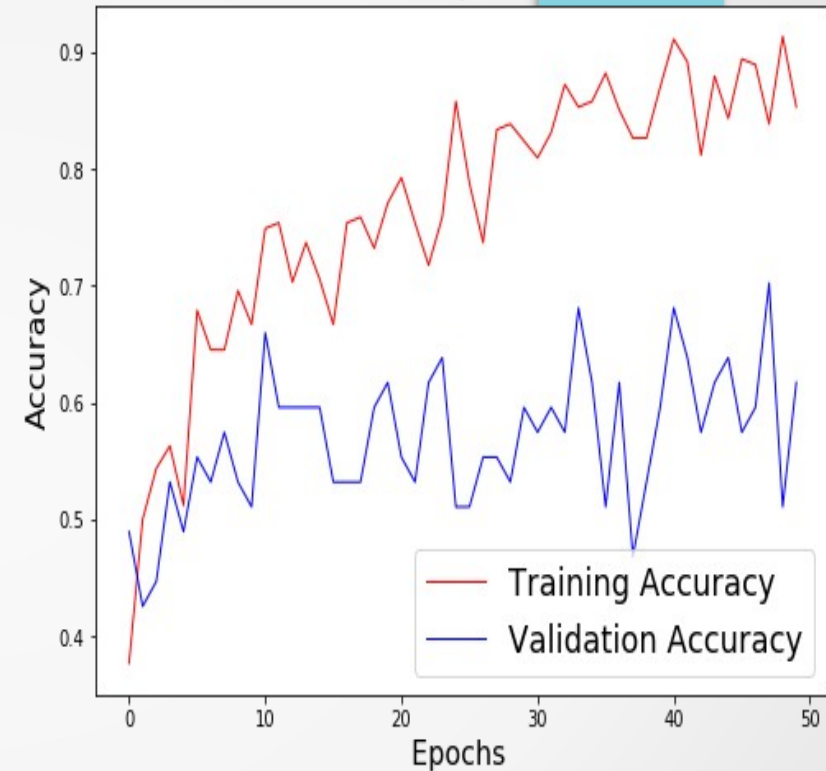
- $$\text{For any neuron } j \in \text{any layer} : \omega_j^{next} = \omega_j^{now} - \eta * \nabla_{\omega_j} Cost$$
$$\text{For any neuron } j \in \text{any layer} : b_j^{next} = b_j^{now} - \eta * \nabla_{b_j} Cost$$

MLP

Loss Curves

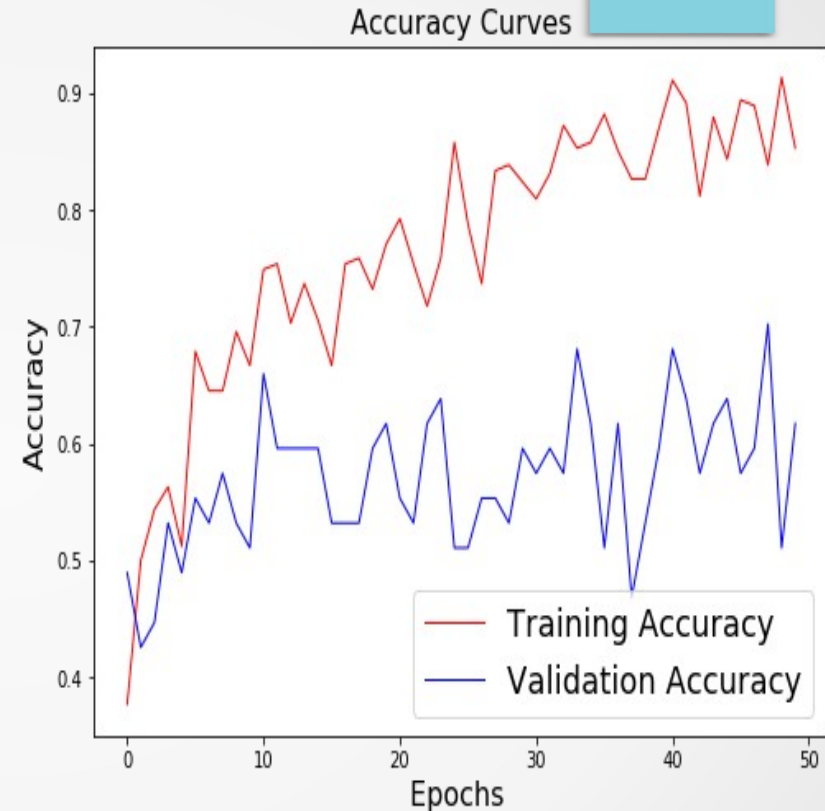
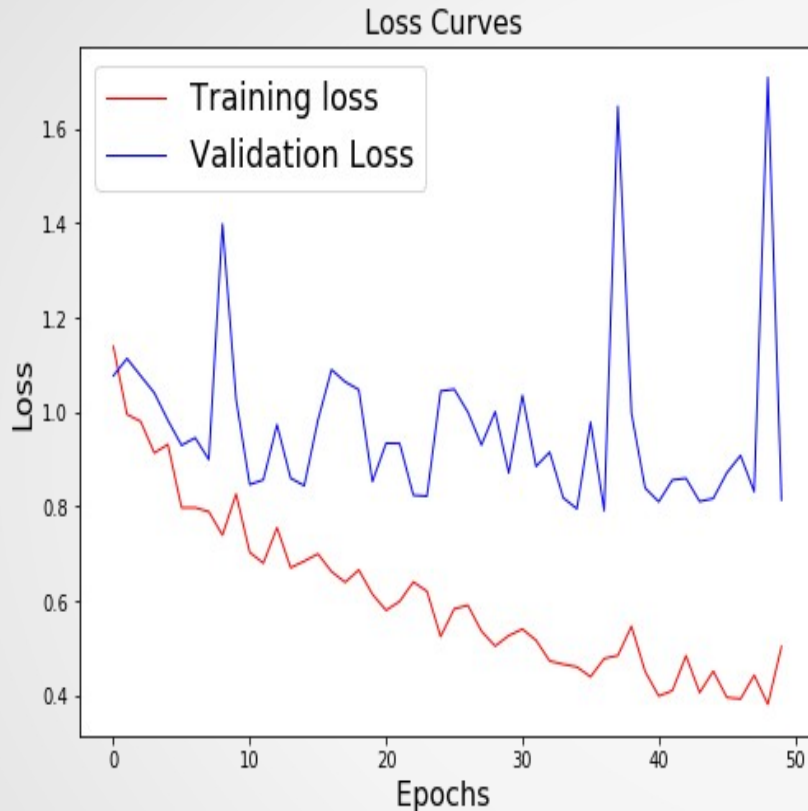


Accuracy Curves



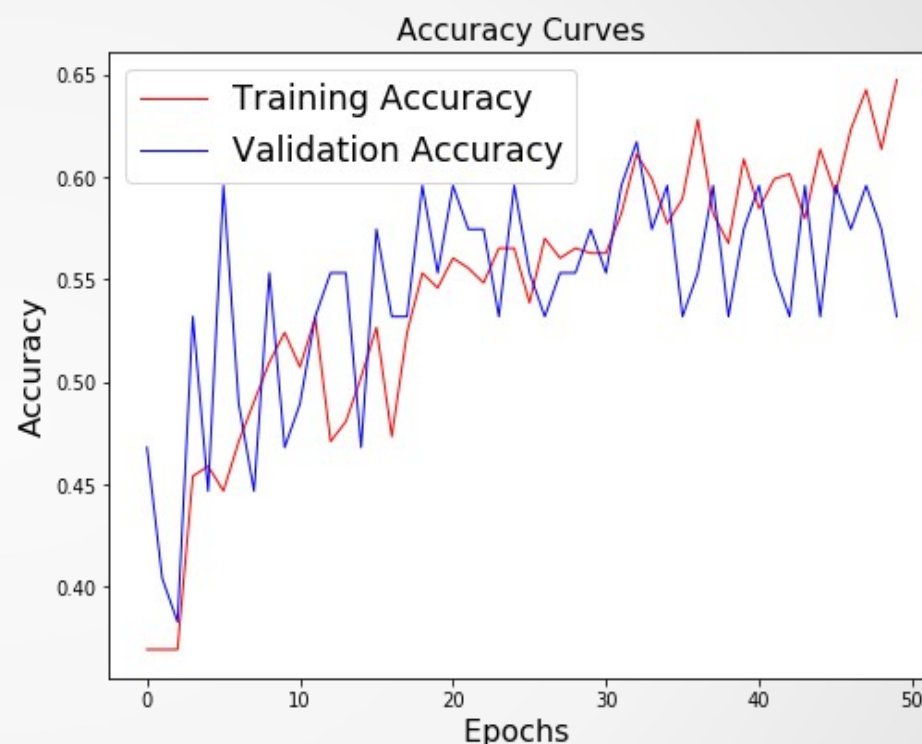
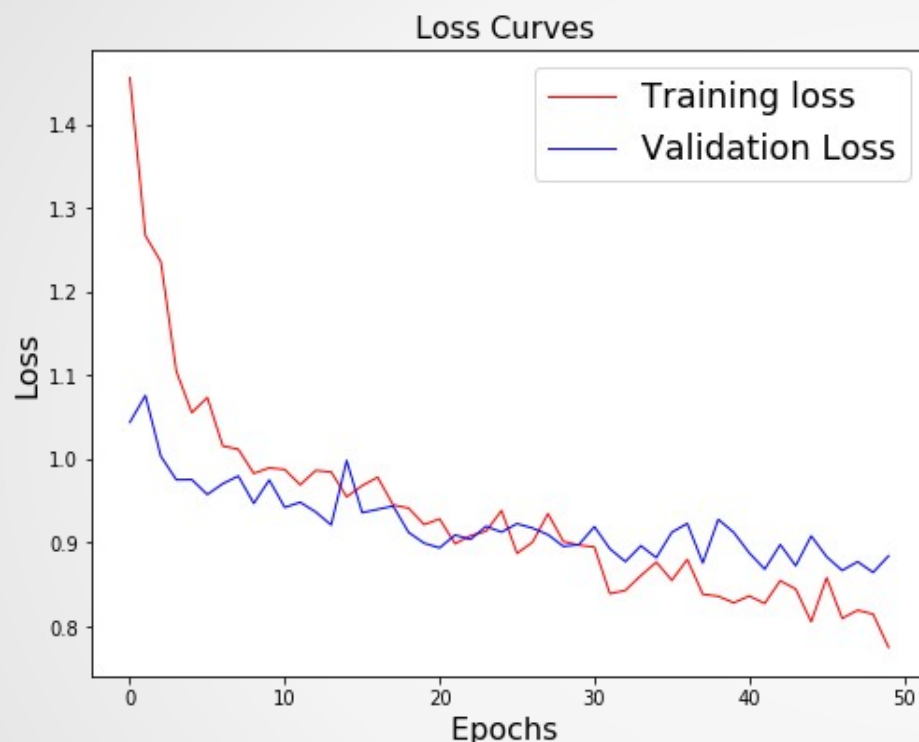
- 3 layers
 - 461 Images / 3 breeds / 322 trained images / 139 tested images
 - Accuracy : → 60 %
-
- Over-fitting issue

MLP : overfitting reduction L2 regularization



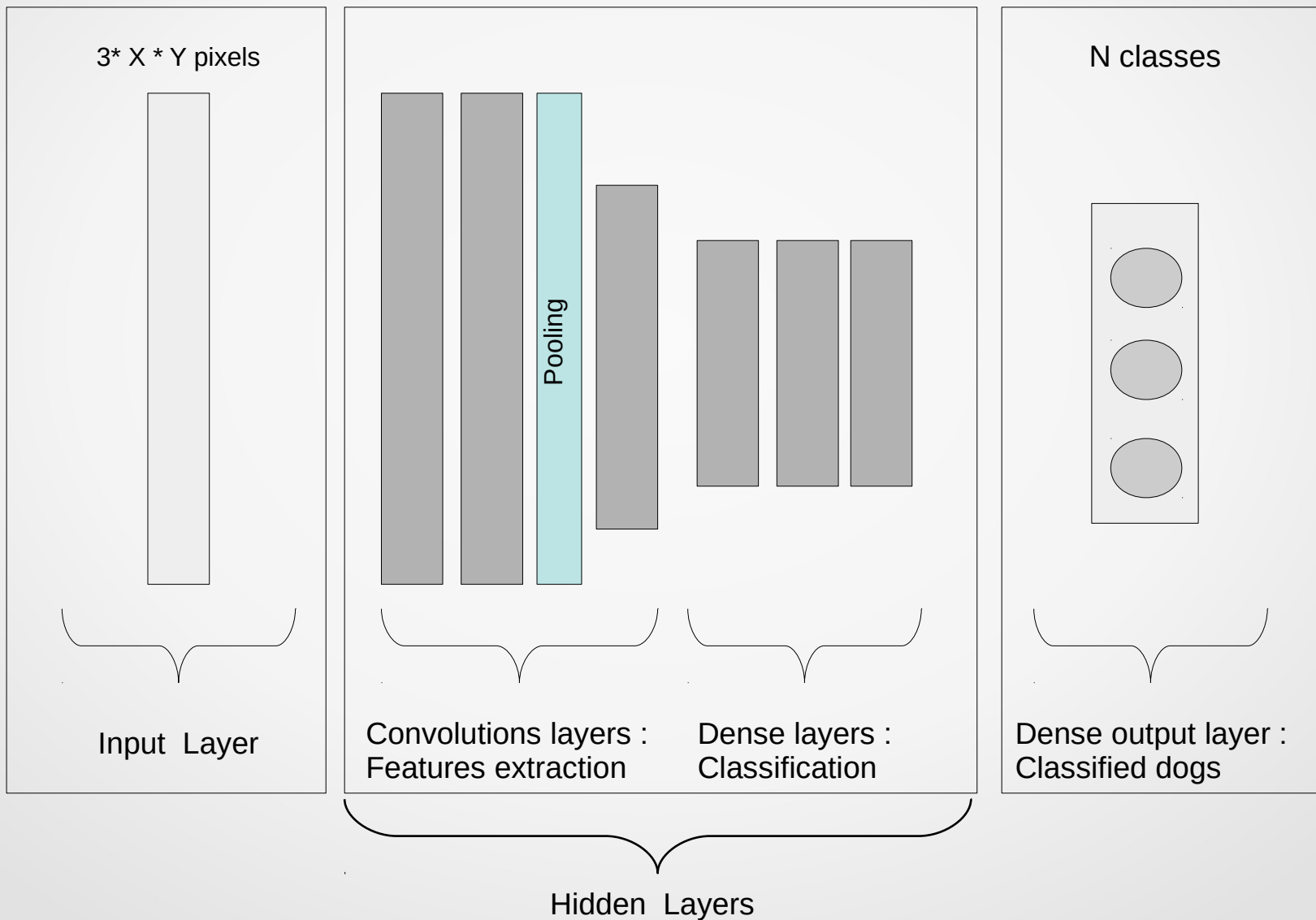
- 3 layers
 - 461 Images / 3 breeds / 322 trained images / 139 tested images
 - Accuracy : → 70 %
-
- Over-fitting issue decreased, better decrease of loss function

MLP : overfitting reduction dropping neurons



- 461 Images / 3 breeds / 414 trained images / 47 tested images
- Accuracy : 53 % → Asymptotically 60 %
- Test loss: stuck on training less
- Learning issue ⇒ more iterations

CNN architecture



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 (MaxPooling2D)	(None, 111, 111, 64)	0
conv2d_9 (Conv2D)	(None, 111, 111, 64)	36928
conv2d_10 (Conv2D)	(None, 109, 109, 64)	36928
max_pooling2d_5 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_11 (Conv2D)	(None, 54, 54, 64)	36928
conv2d_12 (Conv2D)	(None, 52, 52, 64)	36928
max_pooling2d_6 (MaxPooling2D)	(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

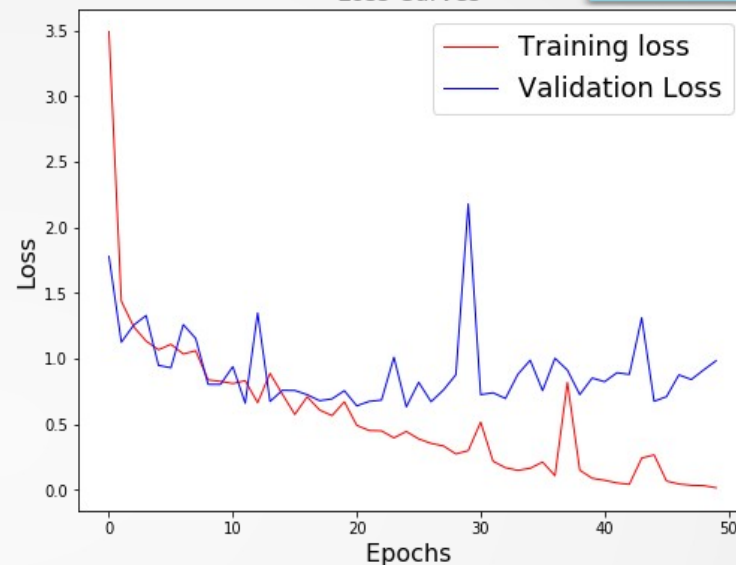
CNN : 8 layers

- 6 convolution layers
- 2 classification layers

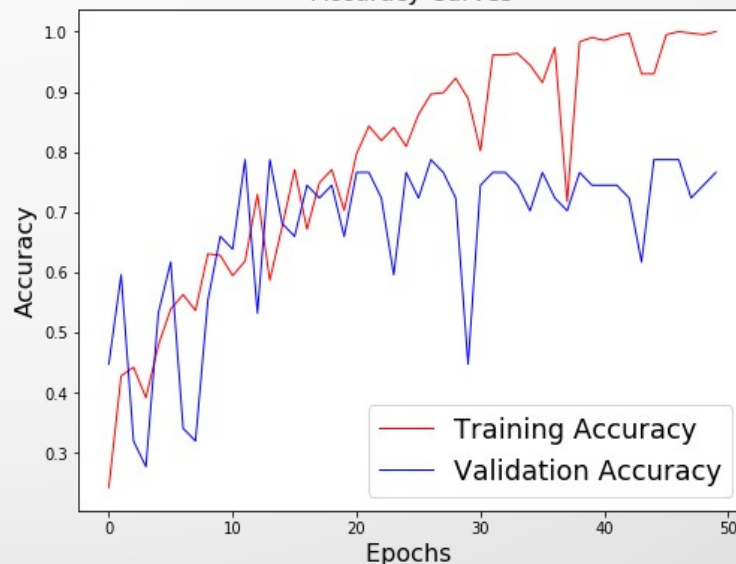
Acc : 76 %

Overfitting issue

Loss Curves



Accuracy Curves



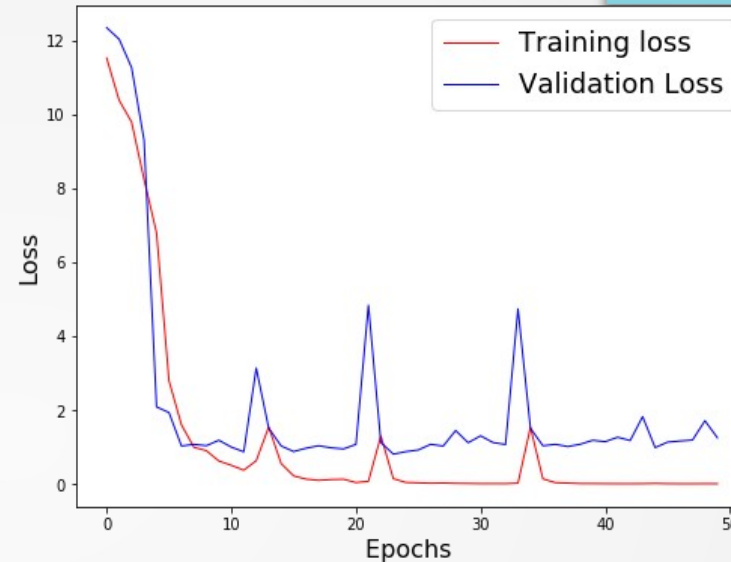
Built CNN using Keras addressing overfitting issue

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 224, 224, 64)	1792
conv2d_2 (Conv2D)	(None, 222, 222, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 111, 111, 64)	0
conv2d_3 (Conv2D)	(None, 111, 111, 64)	36928
conv2d_4 (Conv2D)	(None, 109, 109, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 54, 54, 64)	0
conv2d_5 (Conv2D)	(None, 54, 54, 64)	36928
conv2d_6 (Conv2D)	(None, 52, 52, 64)	36928
max_pooling2d_3 (MaxPooling2D)	(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

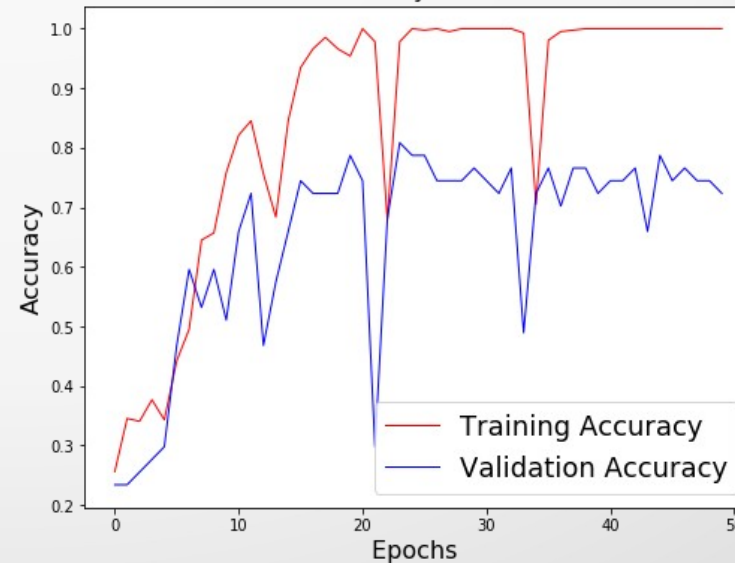
=====
Total params: 18,269,952
Trainable params: 18,269,952
Non-trainable params: 0

CNN 5 layers + drop layer
Acc : 75 %
Overfitting issue

Loss Curves

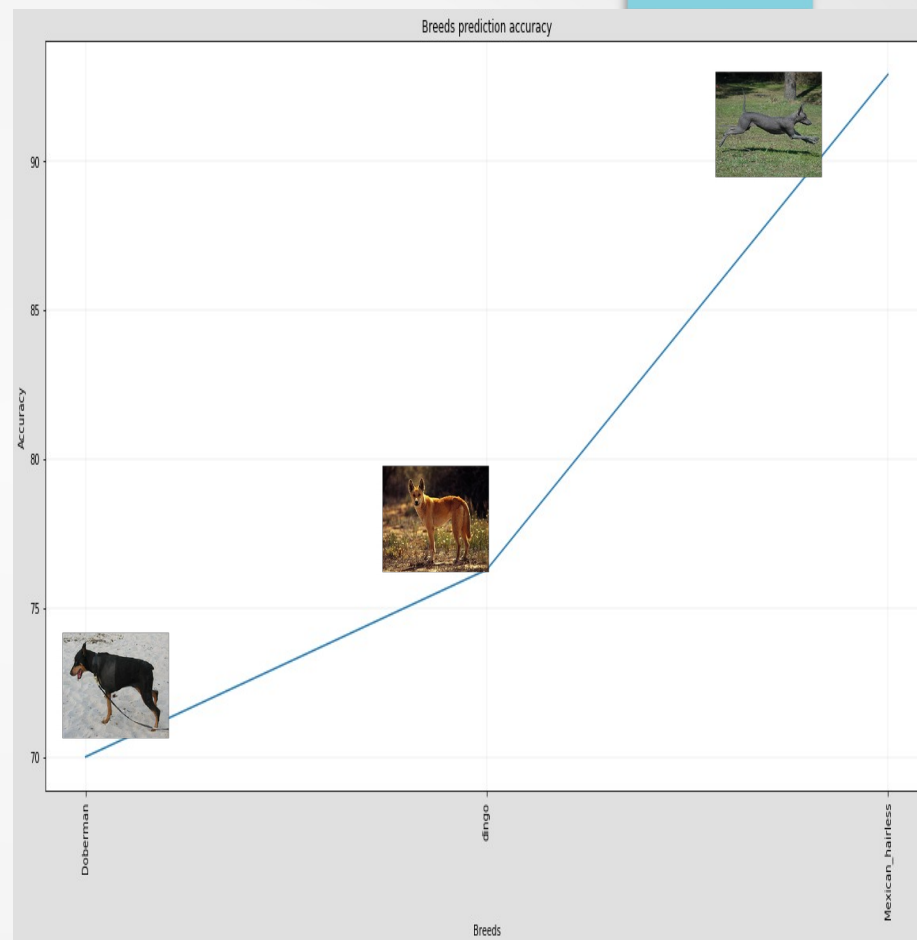


Accuracy Curves



Use of pre-trained VGG16

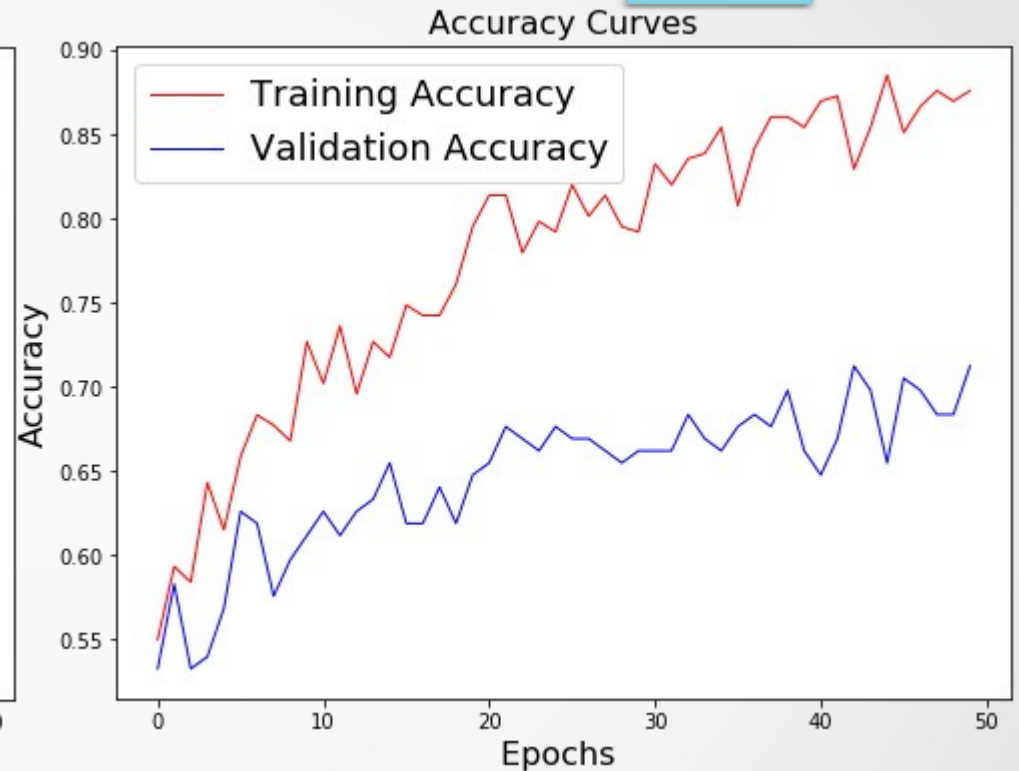
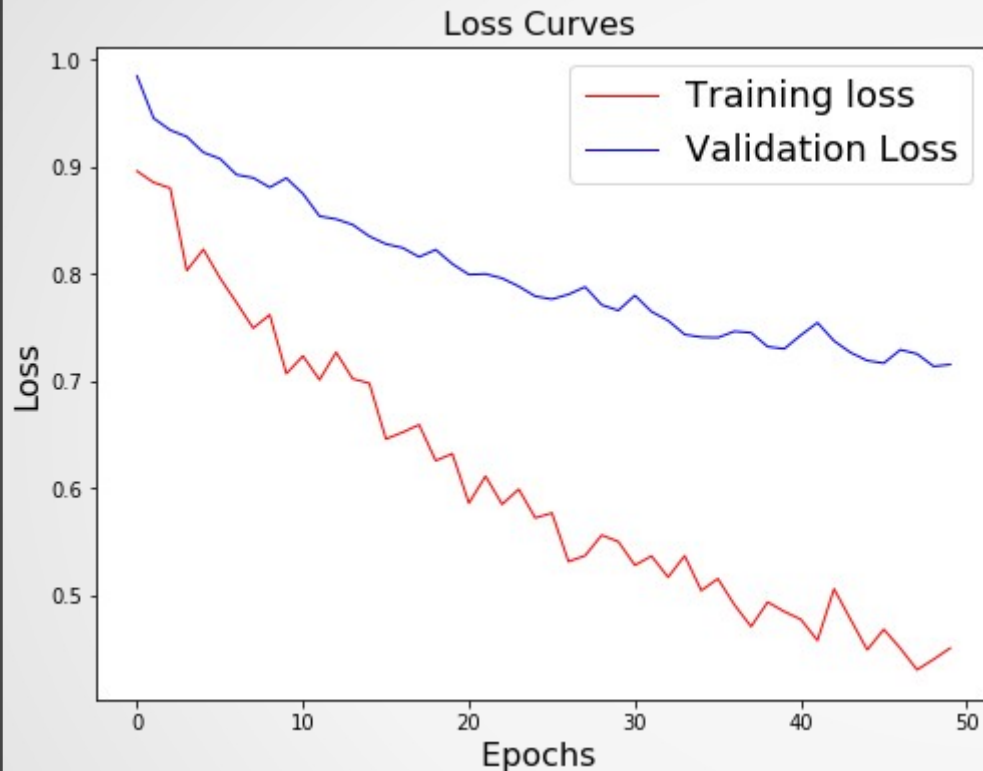
Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 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

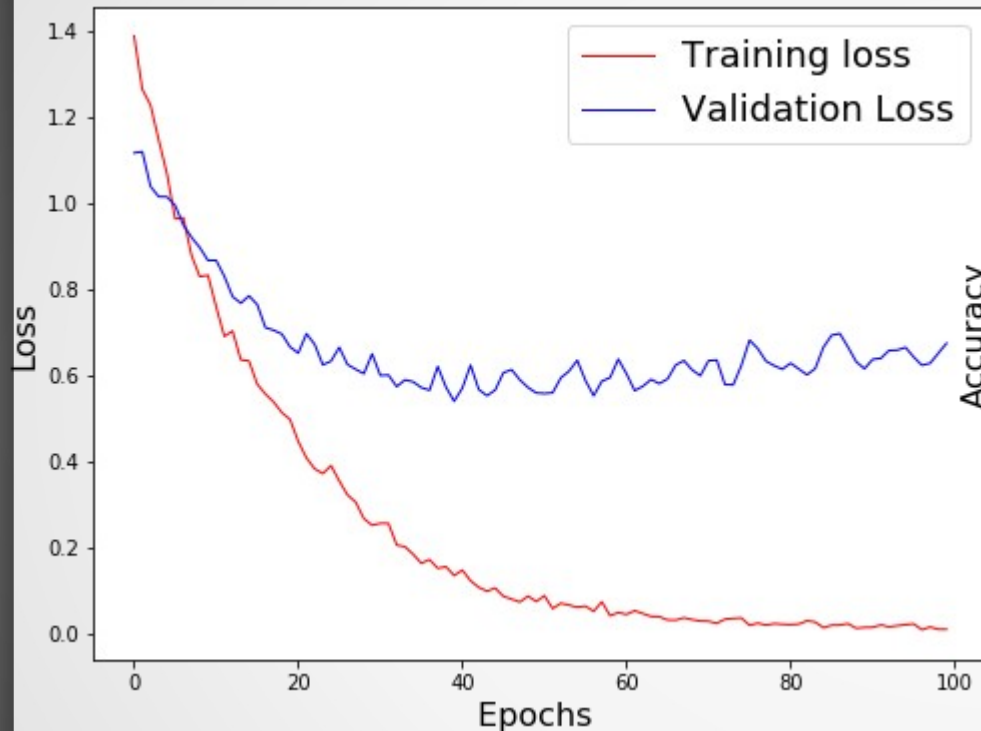
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 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

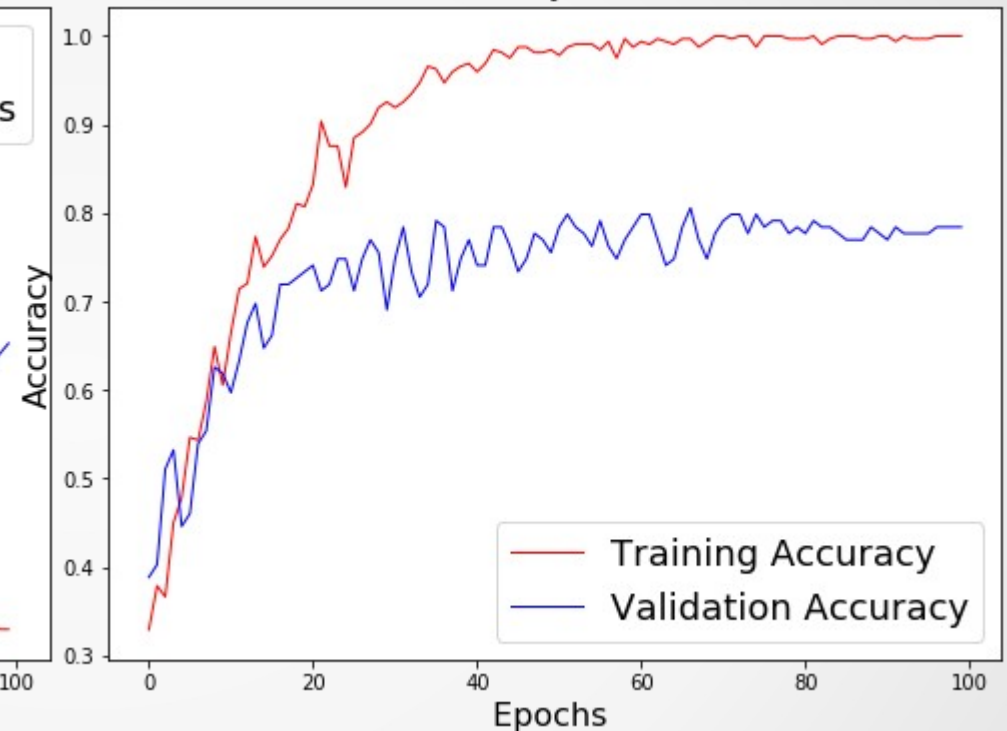
Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, 7, 7, 512)	14714688
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 512)	12845568
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 3)	1539
=====		
Total params: 27,561,795		
Trainable params: 27,561,795		
Non-trainable params: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

Transfer learning : fine tuning of VGG16

Loss Curves

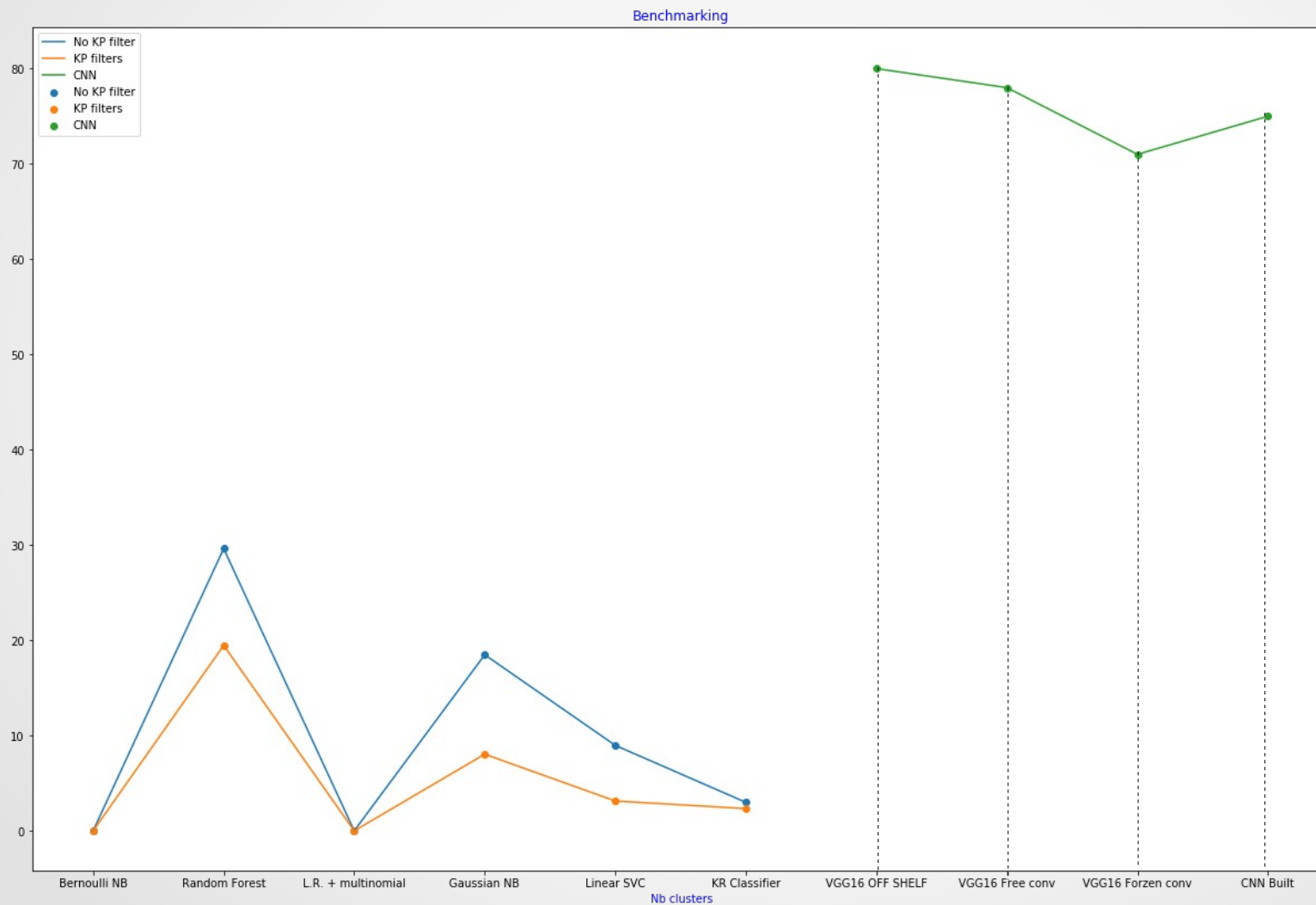


Accuracy Curves



- Free convolution layers
 - 461 Images / 3 breeds / 322 trained images / 139 tested images
 - Accuracy : 78 %
-
- Over-fitting issue

Benchmark results over 3 breeds / 461 dogs



Conclusions & perspectives

- M.L. algorithms:
 - SIFT features extraction may be limited
 - Grey scale \Rightarrow 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 \Rightarrow 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 build 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