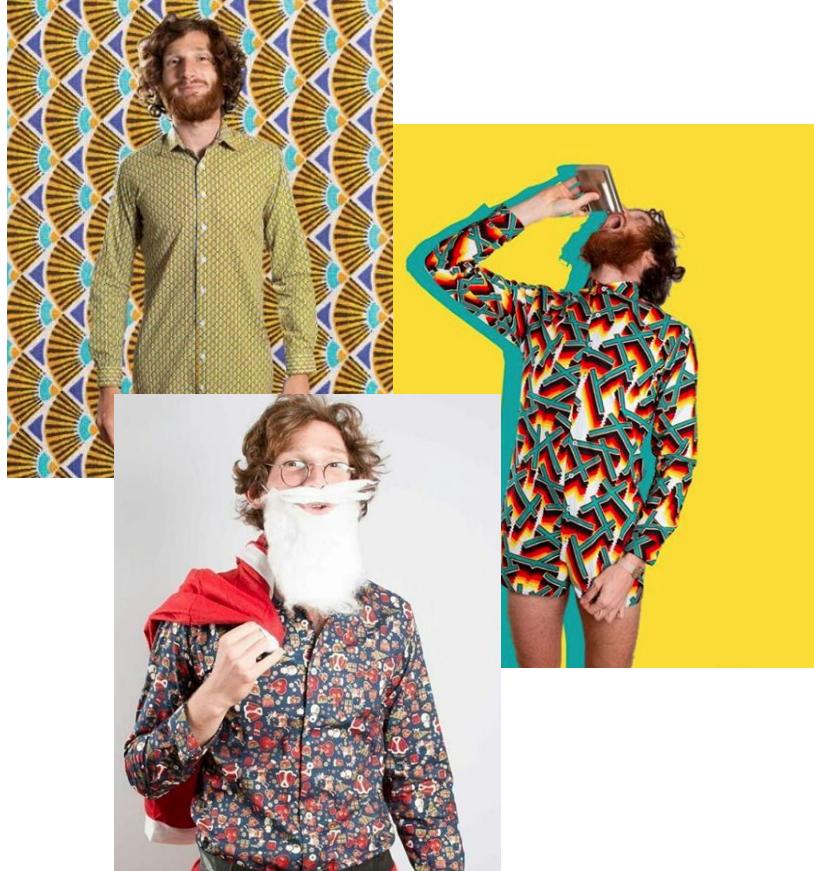


SICARA

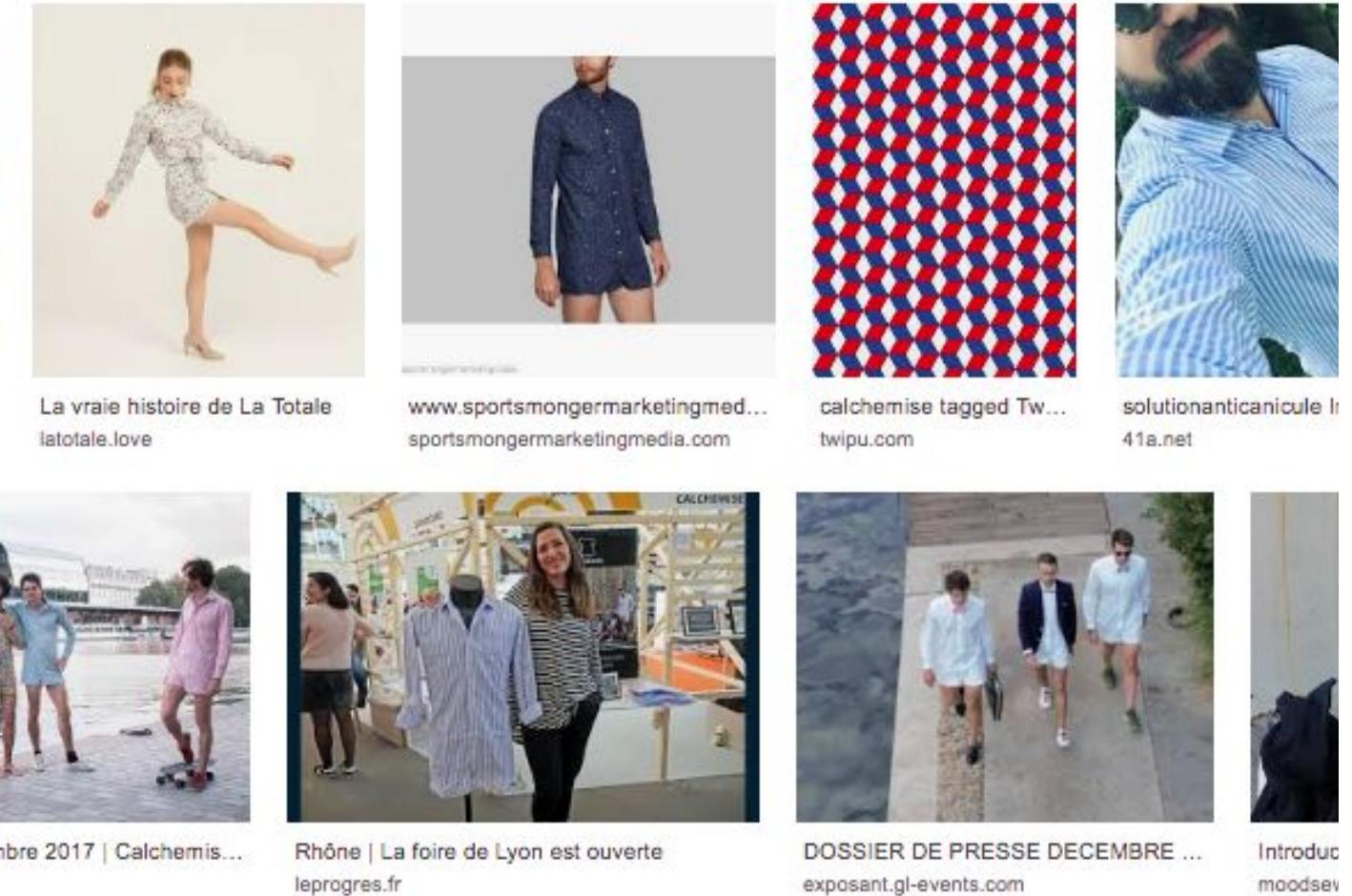
Few shot learning in Production

Few shot learning en production

QUESAKO



k (1~10) reference images per label



n real life images to classify

SOMMAIRE

1. INTRODUCTION

- a. Concept
- b. Literature
- c. Practical considerations

2. SIAMESE NETS

- a. Training
- b. Visualizations

3. CODE

- a. keras_fsl
- b. notebooks

4. CONCLUSION

SOMMAIRE

1. INTRODUCTION

- a. Concept
- b. Literature
- c. Practical considerations

2. SIAMESE NETS

- a. Training
- b. Visualizations

3. CODE

- a. keras_fsl
- b. notebooks

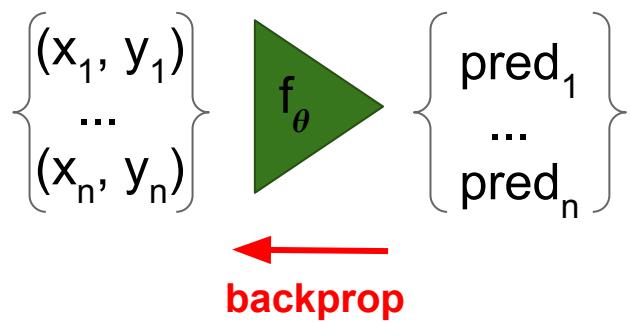
4. CONCLUSION

Concept

LEARNING / META LEARNING

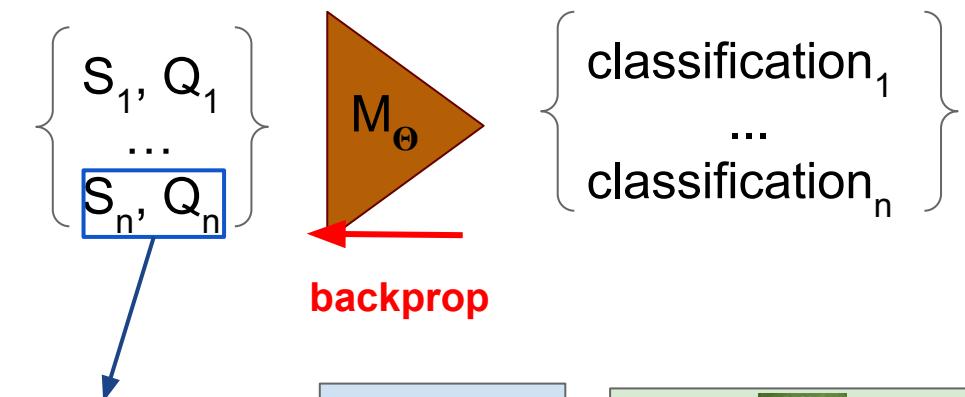
Learning : learn $image \mapsto label$

Training epoch

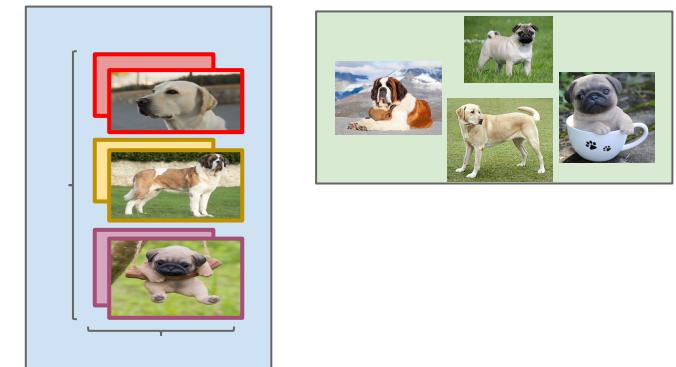


Meta-learning : learn $support-set \mapsto (image \mapsto label)$

Training epoch



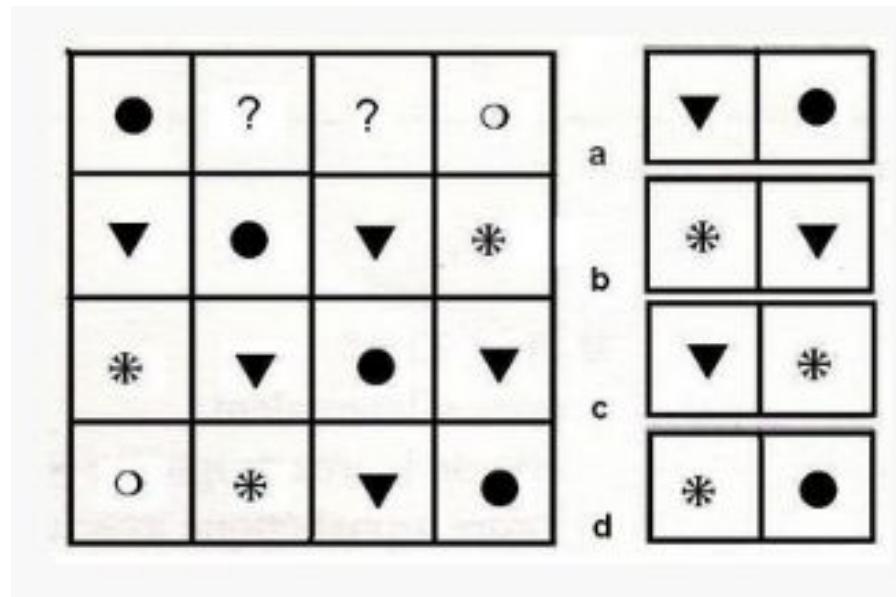
episode =



Concept

META-LEARNING : TWO APPROACHES

Without retraining



With retraining



Concept

META-LEARNING : DEUX POSSIBILITÉS

Without retraining

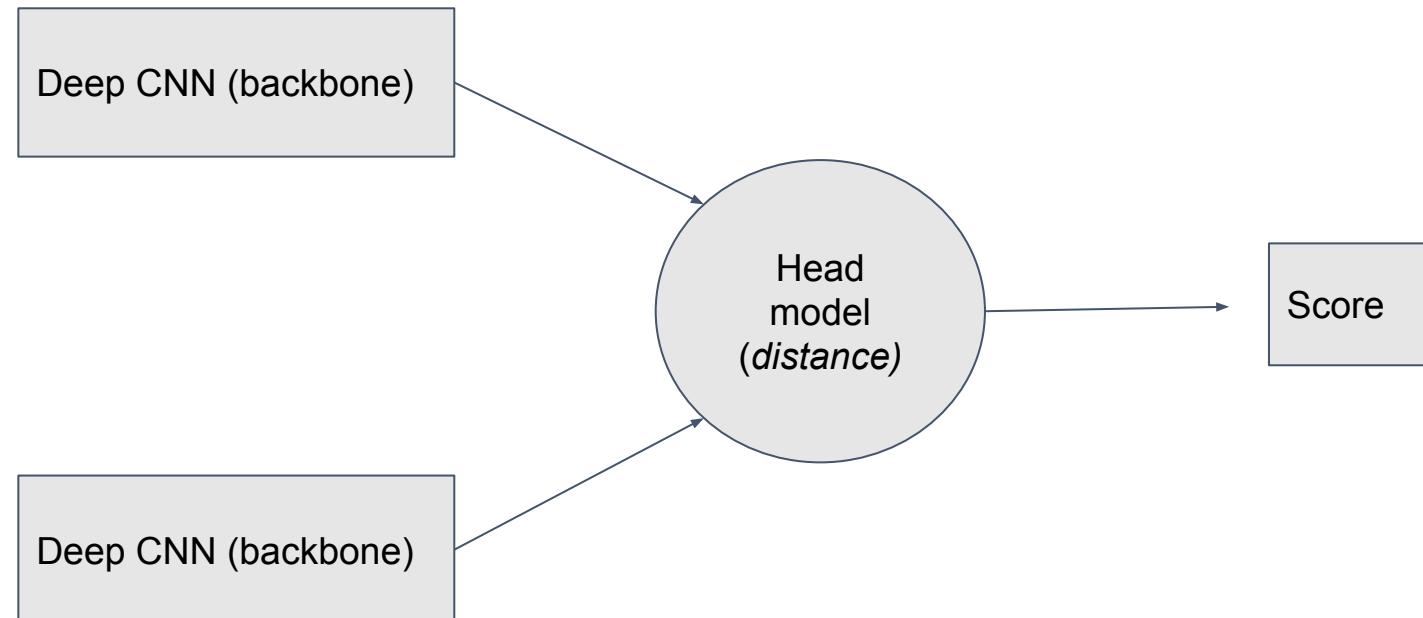
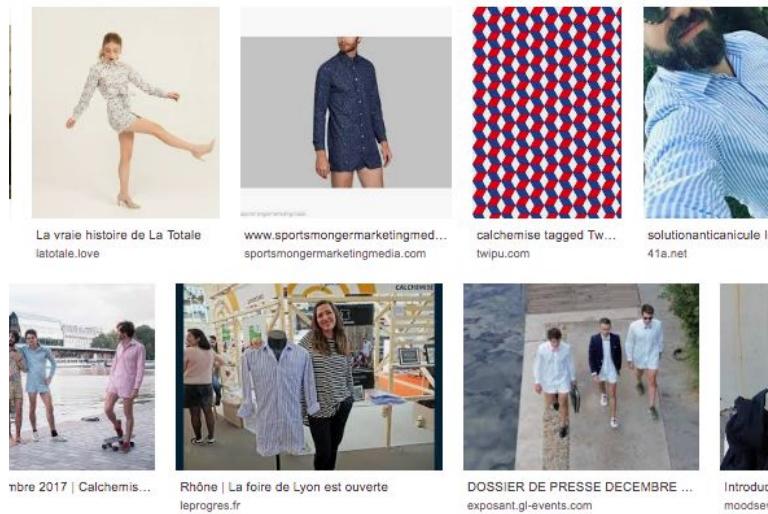
●	?	?	○
▼	●	▼	*
*	▼	●	▼
○	*	▼	●

a	▼	●
b	*	▼
c	▼	*
d	*	●

With retraining



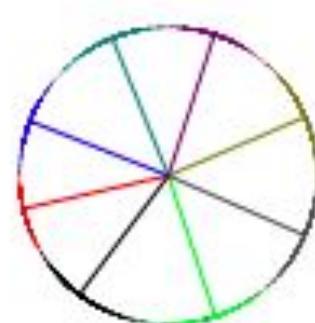
Concept METRIC LEARNING



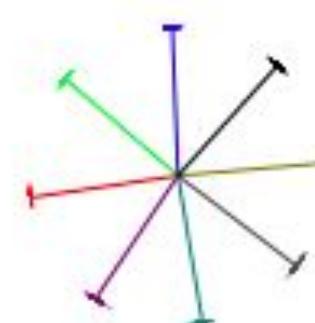
Introduction

LITERATURE

- [A closer look at few shot learning, Chen et al. \(ICLR 19'\)](#)
 - “deeper backbones significantly reduce the performance differences among methods”
- [Few-Shot Learning with Localization in Realistic Settings, Wertheimer & Hariharan, 19'](#)
 - “We show that prior methods designed for few-shot learning do not work out of the box in real world recognition problems”
- Face recognition problem
 - Similar but with a lot of training data
 - SotA : distance learning.
 - [ArcFace: Additive Angular Margin Loss for Deep Face Recognition, Deng et al. 19'](#)



(a) Softmax



(b) ArcFace

Introduction

PRACTICAL CONSIDERATIONS

	Academic research	Production
Dataset	Balanced	Random
Initial weights	Random	Pre-trained (ex. ImageNet)
Dataset size	Constant	Increasing
Support set	Constant	Dynamic
Add of a new class	Theoretical	Instantaneous

SOMMAIRE

1. INTRODUCTION

- a. Concept
- b. Literature
- c. Practical considerations

2. SIAMESE NETS

- a. Training
- b. Visualizations

3. CODE

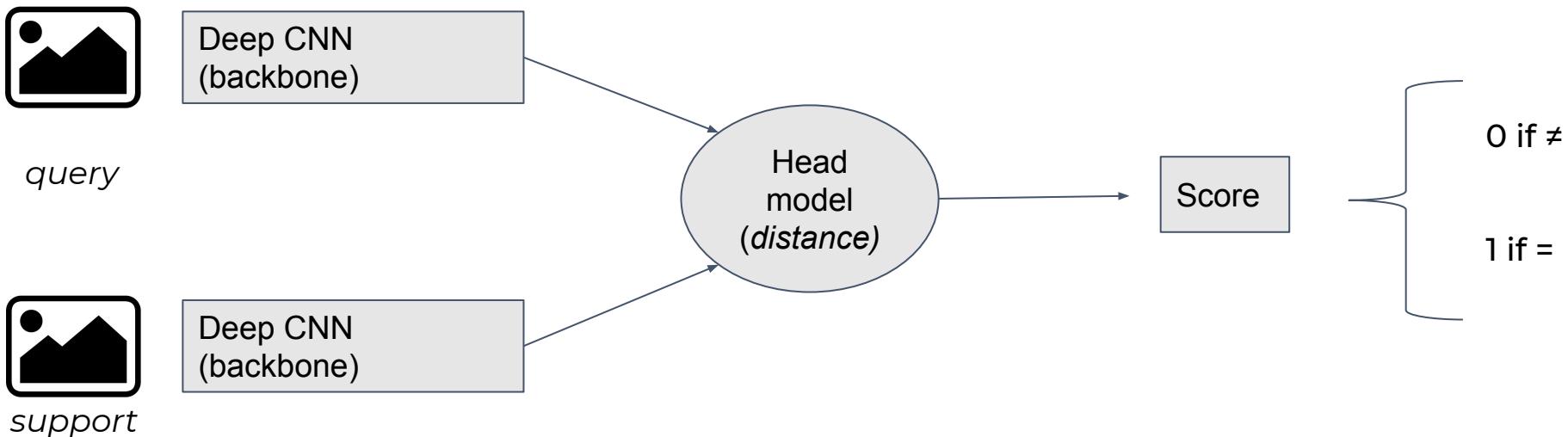
- a. keras_fsl
- b. notebooks

4. CONCLUSION

Siamese Nets

TRAINING

- [Siamese Neural Networks for One-shot Image Recognition, Koch et al. 15'](#)
- Architecture

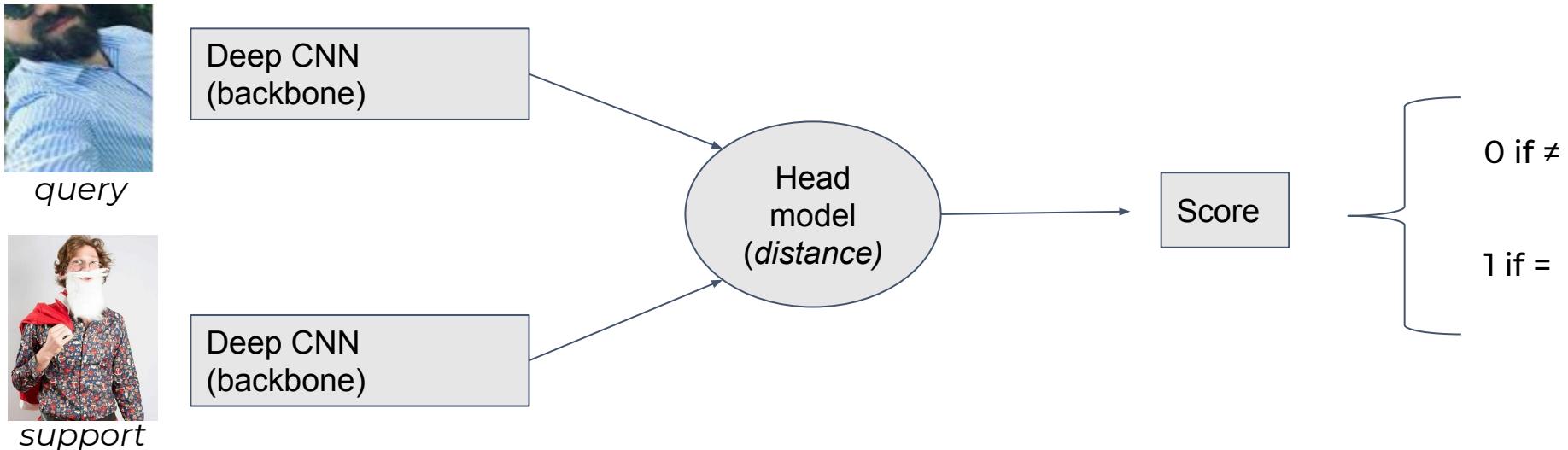


- **Training**
 - generation of pairs of same and different classes
- **Prediction**
 - use the learnt *metric* to in any usual classifier in the feature space
 - e.g: nearest neighbor (usual Siamese nets)
 - nearest prototypes (ProtoNets)

Siamese Nets

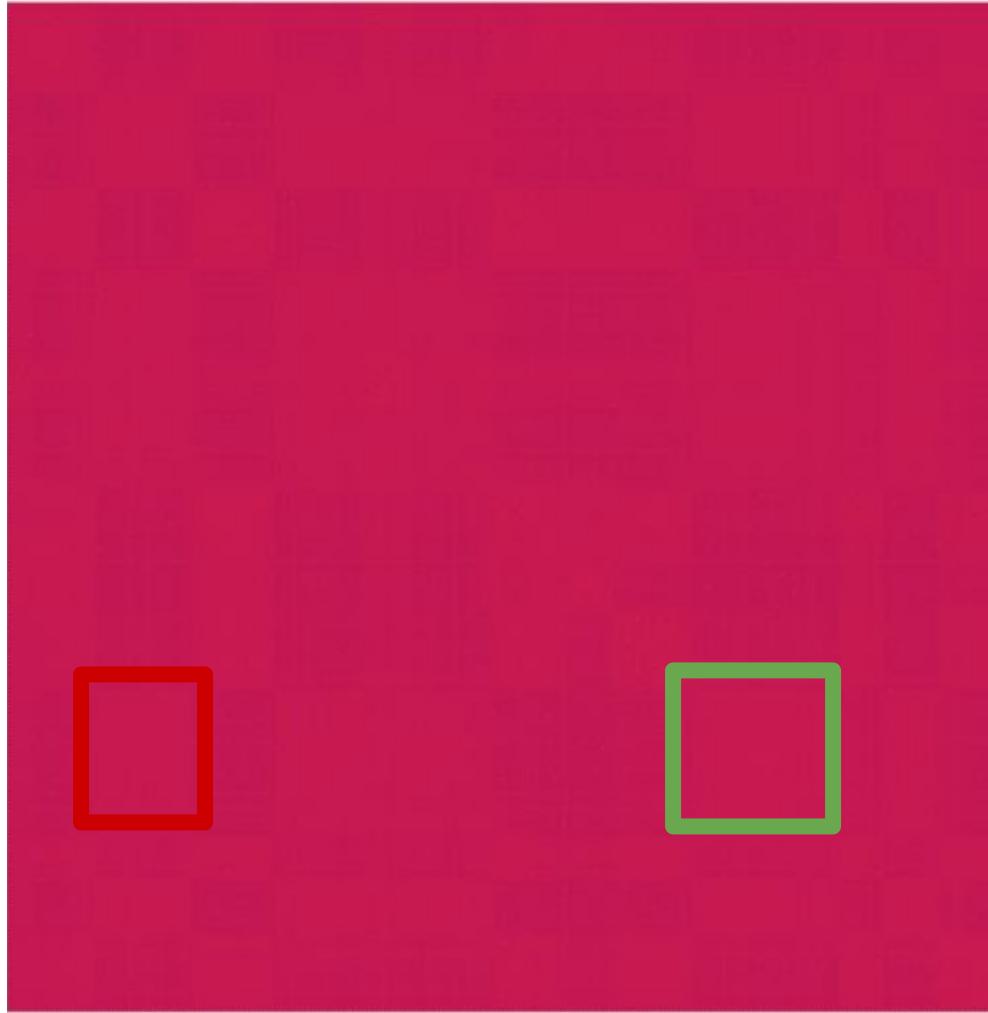
TRAINING

- [Siamese Neural Networks for One-shot Image Recognition, Koch et al. 15'](#)
- Architecture



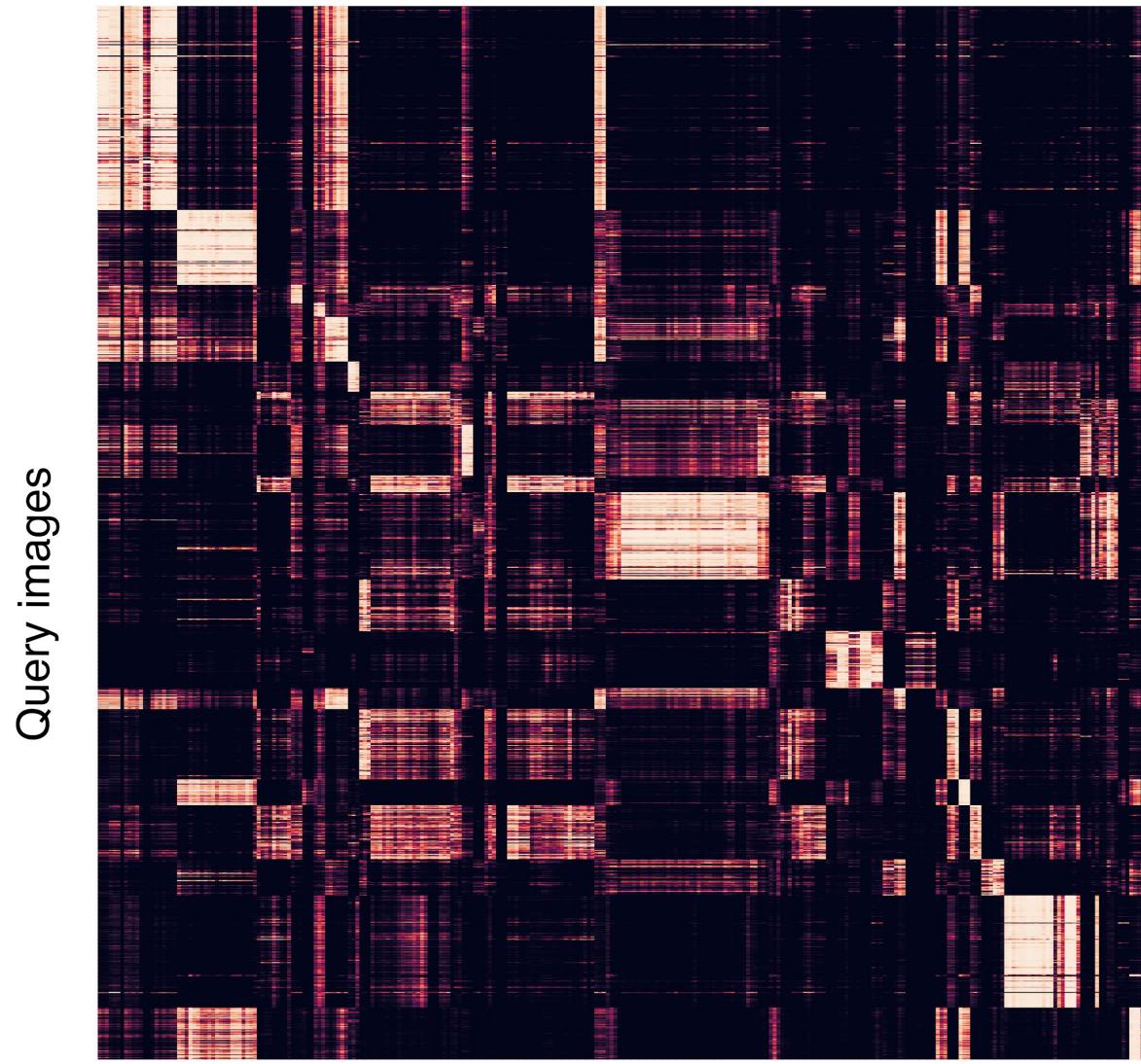
- Training
 - generation of pairs of same and different classes
- Prediction
 - use the learnt *metric* to in any usual classifier in the feature space
 - e.g: nearest neighbor (usual Siamese nets)
 - nearest prototypes (ProtoNets)

Siamese Nets VISUALISATION



pair-wise distance matrix over the train set during training; classes are visibles as blocks

Siamese Nets VISUALISATION



pair-wise distance matrix test set against support set; unsorted query images

SOMMAIRE

1. INTRODUCTION

- a. Concept
- b. Literature
- c. Practical considerations

2. SIAMESE NETS

- a. Training
- b. Visualizations

3. CODE

- a. keras_fsl
- b. notebooks

4. CONCLUSION

Code keras_fsl

The screenshot shows a code editor interface with the following details:

- Title Bar:** Keras-FewShotLearning [/Volumes/TimeMachine/Documents/Code/Keras-FewShotLearning] - .../keras_fsl/models/siamese_nets.py
- Toolbar:** Includes standard icons for file operations (New, Open, Save, Print, Find, Replace, etc.), Git status (Commit, Push, Pull, etc.), and other development tools.
- Project Explorer:** Shows the project structure under "Keras-FewShotLearning". The "models" folder contains several subfolders and files, with "siamese_nets.py" currently selected and highlighted in blue.
- Code Editor:** Displays the Python code for "siamese_nets.py". The code defines a class "SiameseNets" that takes parameters for branch and head models, handles their initialization, and creates a final model by stacking them. It uses the Keras library's Model class and Input layer.

```
5
6
7     def SiameseNets(
8         branch_model='KochNet',
9         head_model='DenseSigmoid',
10        *args,
11        weights=None,
12        **kwargs,
13    ):
14        if not isinstance(branch_model, Model):
15            if isinstance(branch_model, str):
16                branch_model = {'name': branch_model}
17                branch_model_name = branch_model['name']
18                branch_model = getattr(branch_models, branch_model_name)(**branch_model.get('init', {}))
19                branch_model = Model(branch_model.inputs, branch_model.outputs, name='branch_model')
20
21        if not isinstance(head_model, Model):
22            if isinstance(head_model, str):
23                head_model = {'name': head_model}
24                head_model_name = head_model['name']
25                head_model_init = {
26                    **head_model.get('init', {}),
27                    'input_shape': branch_model.output.shape[1:],
28                }
29                head_model = getattr(head_models, head_model_name)(**head_model_init)
30                head_model = Model(head_model.inputs, head_model.outputs, name='head_model')
31
32        inputs = [Input(shape=branch_model.input_shape[1:], name=f'input_{i}') for i in range(len(head_model.inputs))]
33        embeddings = [branch_model(input_) for input_ in inputs]
34        output = head_model(embeddings)
35
36        model = Model(inputs, output, *args, **kwargs)
37        if weights is not None:
38            model.load_weights(weights)
39
40        return model
41
```

- Status Bar:** Shows the current file is "41:1 LF UTF-8 4 spaces" and the Git status is "Git: master KerasFSL @pct".

Code keras_fsl

Keras-FewShotLearning [/Volumes/TimeMachine/Documents/Code/Keras-FewShotLearning] - .../keras_fsl/models/siamese_nets.py

encoder

```
5
6
7     def SiameseNets(
8         branch_model='KochNet',
9         head_model='DenseSigmoid',
10        *args,
11        weights=None,
12        **kwargs,
13    ):
14        if not isinstance(branch_model, Model):
15            if isinstance(branch_model, str):
16                branch_model = {'name': branch_model}
17                branch_model_name = branch_model['name']
18                branch_model = getattr(branch_models, branch_model_name)(**branch_model.get('init', {}))
19                branch_model = Model(branch_model.inputs, branch_model.outputs, name='branch_model')
20
21        if not isinstance(head_model, Model):
22            if isinstance(head_model, str):
23                head_model = {'name': head_model}
24                head_model_name = head_model['name']
25                head_model_init = {
26                    **head_model.get('init', {}),
27                    'input_shape': branch_model.output.shape[1:],
28                }
29                head_model = getattr(head_models, head_model_name)(**head_model_init)
30                head_model = Model(head_model.inputs, head_model.outputs, name='head_model')
31
32        inputs = [Input(shape=branch_model.input_shape[1:], name=f'input_{i}') for i in range(len(head_model.inputs))]
33        embeddings = [branch_model(input_) for input_ in inputs]
34        output = head_model(embeddings)
35
36        model = Model(inputs, output, *args, **kwargs)
37        if weights is not None:
38            model.load_weights(weights)
39
40        return model
41
```

41:1 LF UTF-8 4 spaces Git: master KerasFSL @pct

Code keras_fsl

```
5
6
7     def SiameseNets(
8         branch_model='KochNet',
9         head_model='DenseSigmoid',
10        *args,
11        weights=None,
12        **kwargs,
13    ):
14        if not isinstance(branch_model, Model):
15            if isinstance(branch_model, str):
16                branch_model = {'name': branch_model}
17                branch_model_name = branch_model['name']
18                branch_model = getattr(branch_models, branch_model_name)(**branch_model.get('init', {}))
19                branch_model = Model(branch_model.inputs, branch_model.outputs, name='branch_model')
20
21        if not isinstance(head_model, Model):
22            if isinstance(head_model, str):
23                head_model = {'name': head_model}
24                head_model_name = head_model['name']
25                head_model_init = {
26                    **head_model.get('init', {}),
27                    'input_shape': branch_model.output.shape[1:],
28                }
29                head_model = getattr(head_models, head_model_name)(**head_model_init)
30                head_model = Model(head_model.inputs, head_model.outputs, name='head_model')
31
32        inputs = [Input(shape=branch_model.input_shape[1:], name=f'input_{i}') for i in range(len(head_model.inputs))]
33        embeddings = [branch_model(input_) for input_ in inputs]
34        output = head_model(embeddings)
35
36        model = Model(inputs, output, *args, **kwargs)
37        if weights is not None:
38            model.load_weights(weights)
39
40        return model
41
```

encoder

pair-wise metric

Code keras_fsl

Keras-FewShotLearning [/Volumes/TimeMachine/Documents/Code/Keras-FewShotLearning] - .../keras_fsl/models/siamese_nets.py

Project ▾ siamese_nets.py

```
5
6
7     def SiameseNets(
8         branch_model='KochNet',
9         head_model='DenseSigmoid',
10        *args,
11        weights=None,
12        **kwargs,
13    ):
14        if not isinstance(branch_model, Model):
15            if isinstance(branch_model, str):
16                branch_model = {'name': branch_model}
17                branch_model_name = branch_model['name']
18                branch_model = getattr(branch_models, branch_model_name)(**branch_model.get('init', {}))
19                branch_model = Model(branch_model.inputs, branch_model.outputs, name='branch_model')
20
21        if not isinstance(head_model, Model):
22            if isinstance(head_model, str):
23                head_model = {'name': head_model}
24                head_model_name = head_model['name']
25                head_model_init = {
26                    **head_model.get('init', {}),
27                    'input_shape': branch_model.output.shape[1:],
28                }
29                head_model = getattr(head_models, head_model_name)(**head_model_init)
30                head_model = Model(head_model.inputs, head_model.outputs, name='head_model')
31
32        inputs = [Input(shape=branch_model.input_shape[1:], name=f'input_{i}') for i in range(len(head_model.inputs))]
33        embeddings = [branch_model(input_) for input_ in inputs]
34        output = head_model(embeddings)
35
36        model = Model(inputs, output, *args, **kwargs)
37        if weights is not None:
38            model.load_weights(weights)
39
40        return model
41
```

encoder

pair-wise metric

full model

41:1 LF UTF-8 4 spaces Git: master KerasFSL @pct

Code notebook

```
%#% Init data
#%# Init model
branch_model_name = 'ResNet50'

preprocessing = iaa.Sequential([
    iaa.Fliplr(0.5),
    iaa.Flipud(0.5),
    iaa.Affine(rotate=(-180, 180)),
    iaa.CropToFixedSize(224, 224, position='center'),
    iaa.PadToFixedSize(224, 224, position='center'),
    iaa.AssertShape((None, 224, 224, 3)),
    iaa.Lambda(lambda images_list, *_: (
        getattr(keras_applications, branch_model_name.lower())(
            preprocess_input(np.stack(images_list), data_format='channels_last')
        )
    )),
])

siamese_nets = SiameseNets(
    branch_model={
        'name': branch_model_name,
        'init': {'include_top': False, 'input_shape': (224, 224, 3), 'pooling': 'avg'}
    },
    head_model={
        'name': 'MixedNorms',
        'init': {
            'norms': [
                lambda x: 1 - tf.nn.l2_normalize(x[0]) * tf.nn.l2_normalize(x[1]),
                lambda x: tf.math.abs(x[0] - x[1]),
                lambda x: tf.nn.softmax(tf.math.abs(x[0] - x[1])),
                lambda x: tf.square(x[0] - x[1]),
            ]
        }
    }
)
branch_depth = len(siamese_nets.get_layer('branch_model').layers)

#%# Train model with Sequences
callbacks = [
    TensorBoard(output_folder),
    ModelCheckpoint(
        str(output_folder / 'best_model.h5'),
    )
]
```

preprocessing + data augmentation

Siamese model

22:1 LF UTF-8 4 spaces Git: master KerasFSL @pct

Code notebook

Keras-FewShotLearning [~/Volumes/TimeMachine/Documents/Code/Keras-FewShotLearning] - .../notebooks/random_balanced_training.py

Project

siamese_nets.py random_balanced_training.py

```
77
78 #%% Train model with Sequences
79 callbacks = [
80     TensorBoard(output_folder),
81     ModelCheckpoint(
82         str(output_folder / 'best_model.h5'),
83         save_best_only=True,
84     ),
85     ReduceLROnPlateau(),
86
87 train_sequence = training.pairs.RandomBalancedPairsSequence(train_set, preprocessor=preprocessing, batch_size=16)
88 val_sequence = training.pairs.RandomBalancedPairsSequence(val_set, preprocessor=preprocessing, batch_size=16)
89
90 siamese_nets.get_layer('branch_model').trainable = False
91 optimizer = Adam(lr=1e-4)
92 siamese_nets.compile(optimizer=optimizer, loss='binary_crossentropy')
93 siamese_nets.fit_generator(
94     train_sequence,
95     validation_data=val_sequence,
96     callbacks=callbacks,
97     initial_epoch=0,
98     epochs=3,
99     use_multiprocessing=True,
100    workers=2,
101 )
102
103 siamese_nets.get_layer('branch_model').trainable = True
104 for layer in siamese_nets.get_layer('branch_model').layers[:int(branch_depth * 0.8)]:
105     layer.trainable = False
106 optimizer = Adam(1e-5)
107 siamese_nets.compile(optimizer=optimizer, loss='binary_crossentropy')
108 siamese_nets.fit_generator(
109     train_sequence,
110     validation_data=val_sequence,
111     callbacks=callbacks,
112     initial_epoch=3,
113     epochs=10,
114     use_multiprocessing=True,
115     workers=2,
116 )
117 siamese_nets = load_model(output_folder / 'best_model.h5')
```

pairs generators

training with transfer learning

43:1 LF UTF-8 4 spaces Git: master KerasFSL @pct

SOMMAIRE

1. INTRODUCTION

- a. Concept
- b. Dans la littérature
- c. En pratique ?

2. L'EXEMPLE DES SIAMESE NETS

- a. Entraînement
- b. Visualisations

3. CODE

- a. keras_fsl
- b. notebooks

4. CONCLUSION

Few-shot learning

CONCLUSION

Meta learning with retraining

- Best performers on benchmarks, but do benchmarks matter?
- Requires retraining whenever the support set changes

Metric learning

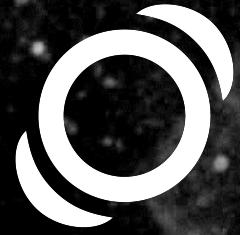
- One single training
- Siamese net with simple classifier (nearest neighbors) more suitable to production diversity

Siamese nets as usual classifier with specific loss

- #pairs = (batch_size)²
- Output usual classifier with internal support set for inference

Ressources

- kaggle : <https://www.kaggle.com/c/humpback-whale-identification/notebooks>
- nanonets.com
- <https://github.com/ClementWalter/Keras-FewShotLearning>



SICARA

Merci