

## TME 7 : Indexation visuelle - Apprentissage structuré

### Travaux sur Machines Encadrés

À toutes fins utiles, se reporter à l'url <http://webia.lip6.fr/~thomen/Teaching/RI.html>. On va s'intéresser à la recherche d'informations visuelles sur des images issues de la base ImageNet<sup>1</sup>. ImageNet est l'une des bases publiques contenant actuellement le plus gros volume d'images annotées : plus d'une dizaine de millions d'images. Les catégories d'ImageNet sont construites à partir de la hiérarchie wordnet<sup>2</sup>. Une compétition pour comparer les algorithmes de reconnaissance visuelle est organisée chaque année depuis 2010 : ImageNet Large Scale Visual Recognition Challenge (ILSVRC), avec  $\sim 10^6$  images d'apprentissage,  $\sim 10^5$  images de test, et 1000 catégories.

On va s'intéresser à une sous-base d'ImageNet constituée de 9 catégories, contenant au total 13196 images, et illustrée à la figure 1. Les images correspondent à trois noeuds différents : "Amphibian" (n01627424) pour Tree-frog, Wood-frog et European fire salamander, "car, auto, automobile, machine, motorcar" (n02958343) pour Ambulance, Taxi et Minivan, et "stringed instrument" (n04338517) pour Harp, Electric Guitar et Acoustic Guitar. Au sein de chacun de ses trois noeuds, la reconnaissance visuelle s'apparente à un problème de recherche d'information à "grain fin" : les images sont visuellement proches et les différences reposent sur des détails précis et localisés (forme et couleur des parties, *etc*).

Le TME 7 vise à générer des indexs visuels selon le modèle "Bag of Words" (BoW) pour la base, puis de mettre en place un algorithme générique pour résoudre un problème d'apprentissage structuré. Les TME 8 et TME 9 proposent d'instancier l'apprentissage structuré à des cas particuliers : classification multi-classe, classification multi-classe hiérarchique (en utilisant l'information disponible dans l'ontologie wordnet), et ordonnancement (ranking) au TME 9.

---

1. <http://www.image-net.org/>  
2. <http://wordnet.princeton.edu/>

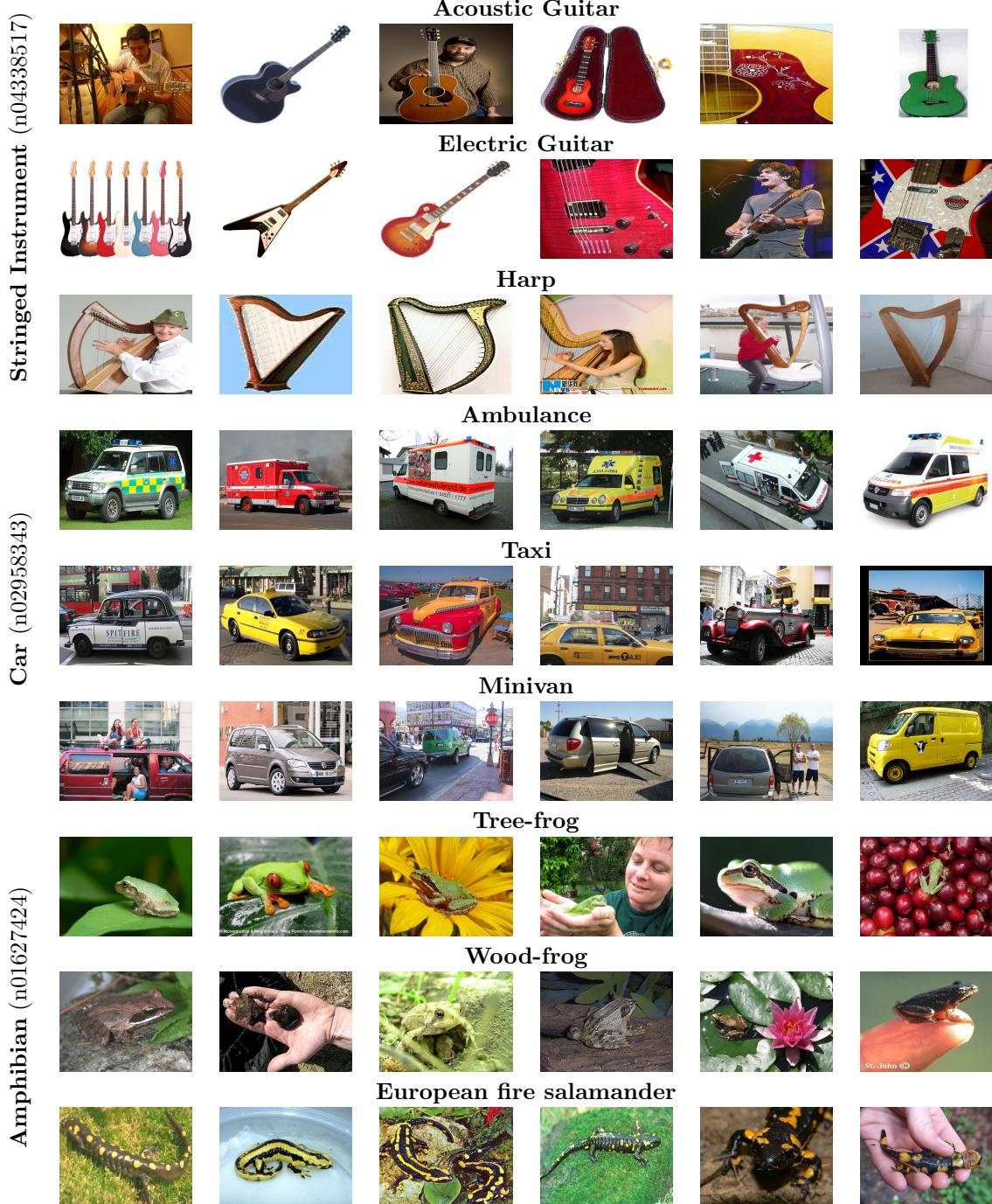


FIGURE 1 – Base d’images utilisée. Au sein de chacun des trois sous-nœuds (Stringed Instrument, Car, Amphibian), la recherche visuelle correspond à un problème de reconnaissance à grain fin.

## Exercice 1 Génération des indexs visuels

Les descripteurs visuels bas niveau, similaires à ceux fournis sur le projet imageNet<sup>3</sup>, sont donnés dans des fichiers texte, un par chaque catégorie : par exemple "taxi.txt". Chaque fichier contient l'ensemble des descripteurs locaux pour chaque image de la catégorie. Plus précisément, la représentation de chaque image correspond à ce qu'on appelle le sac de descripteurs ("Bag of Features", BoF) : des patchs d'images sont échantillonnnés de manière uniforme - figure 2a) - et un descripteur SIFT est calculé pour chaque patch. La Classe `ImageFeatures` du package `upmc.ri.index` contient la représentation visuelle de chaque image fournie :

- Un identifiant de chaque image (son nom, *e.g.* n01644373).
- Une liste content le mot visuel associé à chaque point (issu d'une étape de clustering sur la base, dans l'espace des descripteurs SIFT) - figure 2b).
- Une liste de coordonnées pixels (x,y) donnant la position de chaque patch.

La classe `ImageNetParser` du package `upmc.ri.io` contient la méthode `List<ImageFeatures> getWords(String filename)` qui permet de parser chaque fichier et de retourner la description (BoF) de toutes les images d'une catégorie. **L'objectif de ce premier exercice est de générer le sac de mots ("Bag Of Words", BoW) à partir du BoF.** On effectuera ici un **codage dur** (l'assignement au mot le plus proche est déjà fourni dans les données), une **agrégation (pooling) somme**. Le **descripteur final sera normalisé pour que sa norme euclidienne ( $\ell_2$ ) soit de 1**.

1. Dans le package `upmc.ri.index`, mettre en place une classe `VIndexFactory` contenant une méthode `static double[] computeBow(ImageFeatures ib)` permettant de calculer le BoW à partir du BoF.
2. Dans le package `upmc.ri.bin`, mettre en place une classe `VisualIndexes` avec une méthode principale permettant de calculer créer (et serialiser) un objet de la classe `DataSet` (package `upmc.ri.struct`) contenant l'ensemble des données d'apprentissage. Pour cela il faudra :
  - Appeler la méthode précédente pour calculer le BoW pour chaque des images. Vérifier le BoW obtenu à la figure 2d). Montrer les résultats de BoW pour d'autres images.
  - Créer des exemples d'apprentissage d'entraînement et de test. On prendra les 800 premiers exemples de chaque classe pour apprendre, les autres pour tester.
3. Finalement, on effectuera une PCA sur les données afin de réduire leur dimension. On conservera ici 250 composantes. Utiliser la méthode `computePCA` fournie pour cela (classe `PCA` du package `upmc.ri.utils`).

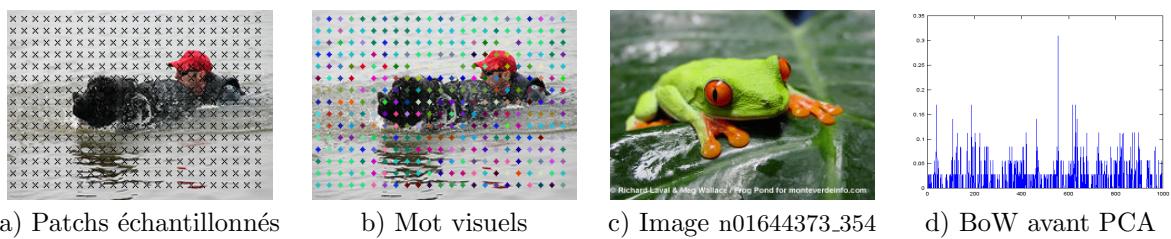


FIGURE 2 – Extraction de sac de mots (BoW) des images de la base

3. <http://image-net.org/download-features>

## Exercice 2 Algorithme d'apprentissage structuré générique

On rappelle tout d'abord les éléments principaux qu'il est nécessaire de définir en apprentissage structuré, afin de définir une architecture de code adaptée et générique.

Un modèle d'apprentissage structuré prend en entrée un élément  $\mathbf{x} \in \mathcal{X}$  (l'espace  $\mathcal{X}$  étant quelconque), et renvoie un élément de sortie  $\mathbf{y} \in \mathcal{Y}$  (l'espace  $\mathcal{Y}$  étant lui aussi quelconque). Deux éléments principaux du modèle doivent être définis pour résoudre un type particulier de problème d'apprentissage structuré :

- Une "joint feature map"  $\Psi(\mathbf{x}, \mathbf{y})$  entre chaque couple entrée/sortie  $(\mathbf{x}, \mathbf{y})$ .  $\Psi(\mathbf{x}, \mathbf{y})$ , qui définit la relation entre l'entrée  $\mathbf{x}$  et la sortie  $\mathbf{y}$ , est un vecteur de  $\mathbb{R}^d$ .
- Une fonction de coût  $\Delta(\mathbf{y}, \mathbf{y}')$  mesurant la dissimilarité entre deux sorties  $\mathbf{y}$  et  $\mathbf{y}'$  (permettant d'introduire une expertise du domaine pour mesurer l'erreur de prédiction).

On s'intéressera dans ces TME à des modèles structurés linéaires, *i.e.* des modèles dont les paramètres  $\mathbf{w} \in \mathbb{R}^d$  la permettent de calculer la sortie prédictive  $\hat{\mathbf{y}}$  de la manière suivante :  $\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}} \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle$ , étape dite d'inférence.

L'apprentissage d'un modèle structuré ainsi défini consiste à minimiser, sur un ensemble d'apprentissage, la fonction de coût  $\Delta(\mathbf{y}_i, \hat{\mathbf{y}}_i)$ , où  $\mathbf{y}_i$  est la sortie connue (pendant l'apprentissage) pour l'entrée  $\mathbf{x}_i$ . Le loss  $\Delta(\mathbf{y}_i, \hat{\mathbf{y}}_i)$  étant généralement complexe à optimiser, la formulation "margin rescaling" définit une borne supérieure convexe, donnant lieu à la fonction objectif suivante :

$$\mathcal{P}(\mathbf{w}) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^n \left[ \max_{\mathbf{y} \in \mathcal{Y}} [\Delta(\mathbf{y}_i, \mathbf{y}) + \langle \Psi(\mathbf{x}_i, \mathbf{y}), \mathbf{w} \rangle] - \langle \Psi(\mathbf{x}_i, \mathbf{y}_i), \mathbf{w} \rangle \right] \quad (1)$$

La fonction de coût  $\mathcal{P}(w)$  est convexe et sous-différentiable, il est donc possible de la minimiser par une méthode (sous-)gradient stochastique, comme illustré à l'algorithme 1. Afin de calculer un sous-gradient de la fonction objectif, il est nécessaire de résoudre le problème suivant, dit de "Loss-Augmented Inference" (LAI) :  $\hat{\mathbf{y}} = \arg \max_{\mathbf{y} \in \mathcal{Y}} [\Delta(\mathbf{y}, \mathbf{y}_i) + \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle]$ .

---

### Algorithme 1 Algorithme d'apprentissage structuré

**Entrées :** Paires d'apprentissage  $\{(\mathbf{x}_i; \mathbf{y}_i) \in \mathcal{X} \times \mathcal{Y}\}_{i \in \{1; N\}}$ , joint feature map  $\Psi(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^d$ , loss  $\Delta(\mathbf{y}, \mathbf{y}')$ , régulariseur  $\lambda$ , nombre d'itérations  $T$ , pas de gradient  $\eta_t$ .

**Sortie :**  $\mathbf{w} \Rightarrow$  fonction de prédiction  $f_w(x) = \arg \max_{\mathbf{y} \in \mathcal{Y}} \langle \mathbf{w}, \Psi(x, \mathbf{y}) \rangle$ .

```

1: w  $\leftarrow \vec{0}$  // Initialisation
2: pour  $t = 1, \dots, T$  faire
3:   pour  $i = 1, \dots, N$  faire
4:      $(x_i; y_i) \leftarrow$  Sélection aléatoire d'une paire d'apprentissage
5:      $\hat{y} \leftarrow \arg \max_{\mathbf{y} \in \mathcal{Y}} [\Delta(\mathbf{y}, \mathbf{y}_i) + \langle \Psi(x_i, \mathbf{y}), \mathbf{w} \rangle]$  // "Loss-augmented inference"
6:      $g_i = \Psi(x_i, \hat{y}) - \Psi(x_i, y_i)$  // Calcul du gradient
7:      $\mathbf{w} \leftarrow \mathbf{w} - \eta_t (\lambda \mathbf{w} + g_i)$  // Mise à jour
8:   fin pour
9: fin pour
10: retourner  $\mathbf{w}$ 

```

---

L'objectif de ce premier TME est de mettre au point un algorithme d'apprentissage qui soit générique, *i.e.* applicable pour n'importe quelle instantiation du modèle structuré.

**Architecture du code** On s'appuiera sur la classe `STrainingSample<X,Y>` dans le package `upmc.ri.struct`, qui fournit la structure de base pour représenter les exemples d'apprentissage. Afin d'avoir un code générique, on propose d'architecturer le code autour des packages et interface suivants, qui sont fournis :

- Dans le package `upmc.ri.struct.model`, une interface `IStructModel<X,Y>` qui définit les méthodes suivantes : `predict(STrainingSample<X,Y>)`, `lai(STrainingSample<X,Y> ts)`, `IStructInstantiation<X,Y> instantiation()` et `getParameters()`.
- Dans le package `upmc.ri.struct.instantiation`, une interface `IStructInstantiation<X,Y>` qui définit les méthodes suivantes : `double[] psi(X x, Y y)`, `double delta(Y y1, Y y2)`, `Set<Y> enumerateY()`.
- Dans le package `upmc.ri.struct.training`, une interface `ITrainer<X,Y>` qui définit la méthode suivante : `train(List<STrainingSample<X, Y>> lts, IStructModel<X,Y> model)`.

Afin de pouvoir implémenter l'apprentissage du modèle, on demande de mettre en place le code suivant :

1. Dans le package `upmc.ri.struct.training`, définir une classe `SGDTrainer<X, Y>` implémentant `ITrainer<X,Y>`, en définissant notamment le code de la méthode `train`. Cette méthode doit être générique et n'utilisera que des appels aux méthodes des interfaces fournies.
  - Dans la classe `SGDTrainer<X, Y>` définir aussi une fonction `convex_loss` qui calculera la fonction objectif définie à l'Eq (1). Munir également la classe d'une variable de type `Evaluator<X, Y>` (fournie dans le package `upmc.ri.struct`), permettant de pouvoir calculer l'évolution de la fonction de coût  $\Delta(\mathbf{y}_i, \hat{\mathbf{y}}_i)$  au cours de l'apprentissage.
2. Dans le package `upmc.ri.struct.model`, définir une classe abstraite `LinearStructModel<X,Y>` implémentant `IStructModel<X,Y>`, contenant :
  - Une variable d'instance de type `IStructInstantiation<X,Y>` pour stocker le type particulier de modèle structuré, et implémentera la méthode `instantiation()` en conséquence.
  - Une variable d'instance de type `double[]` pour stocker les paramètres du modèle structuré linéaire, et implémentera la méthode `getParameters()` en conséquence.
  - On définira aussi le constructeur `LinearStructModel(int dimpsi)` prenant en paramètres la dimension de l'espace des paramètres.
3. Dans le package `upmc.ri.struct.model`, définir une classe `LinearStructModel_Ex<X,Y>` héritant de `LinearStructModel<X,Y>`, qui implémentera les méthodes `predict(X x)` et `lai(STrainingSample<X,Y> ts)`. Pour cela, cette classe sera dédiée aux modèles pour lesquels la prédiction et la résolution du problème de "Loss-Augmented Inference" (LAI) seront effectués de manière exhaustive, c'est à dire en parcourant de manière naïve l'espace  $\mathcal{Y}$  pour effectuer la maximisation.