Conversion de données vers MNIST

Vivien Kraus

29 mars 2017

Table des matières

1 Introduction

Le but de ce programme est de convertir des bases de données de *deep learning* vers le format MNIST, tel que défini sur le site de Yann LeCun [1].

Ici, nous allons avoir pour les données d'entrée des blocs de matrices, donc le « nombre magique » IDX est 4.

Nous allons charger des données sous la forme suivante : la base de données d'images est séparée entre **tests** et **apprentissage**, et chaque dossier contient des images dont le nom est `target_identifiant_vue.jpg`, avec `target` le nom de la conclusion en toutes lettres, `identifiant` un nombre qui définit un individu, et `vue` le numéro de la vue. Les identifiants sont discontinus.

Les noms des cibles sont transformés en numéro, pour retrouver les noms, nous créons aussi un fichier d'indice avec une conclusion par ligne.

2 Compilation

Vous pouvez télécharger l'archive source ici. Décompressez-la. Vous devez avoir sur votre ordinateur :

1. `http://yann.lecun.com/exdb/mnist/`

— ocamlfind ;
— ocamlbuild ;
— camlimages (avec le module jpeg).

Le meilleur moyen est d'utiliser OPAM, plus de détails ici.

Une fois l'archive téléchargée et dépaquetée, il faut compiler. Si vous avez enregistré dans la variable `PATH_TO_SHREC_DATABASE` le chemin d'accès vers la base SHREC (un dossier contenant deux sous-dossiers, **test** et **train**), et dans `MNIST_OUTPUT_DIR` le chemin où enregistrer les fichiers idx, vous pouvez utiliser la commande suivante :

```
cd idx
ocamlbuild -use-ocamlfind \
           src/main.native -- \
           $PATH_TO_SHREC_DATABASE $MNIST_OUTPUT_DIR
```

Listing 1: Commande pour exécuter le programme

3 Implémentation

Nous allons utiliser Objective CAML.

3.1 Composition du fichier de sortie

Le but est de calculer la composition du fichier de sortie. On obtient l'interface suivante :

```
(** Compute the file content from the input data. *)

(** [write_idx dimensions write f] will call [write] in turn to all
bytes.  The input is given by [f coordinates], where [coordinates] are
the coordinates, such that the i-th coordinate is between 0 and the
i-th dimension. *)
val write_idx: int list -> (int -> unit) -> ((int list -> int) -> unit)
```

Et voici l'implémentation :

```
let rec to_net acc = function
  | n when n <= 0 -> acc
  | n -> to_net ((n mod 256) :: acc) (n / 256)

let int32_to_net x =
  match to_net [] (Int32.to_int x) with
  | [] -> [|0; 0; 0; 0|]
  | [x] -> [|0; 0; 0; x|]
  | [x; y] -> [|0; 0; x; y|]
  | [x; y; z] -> [|0; x; y; z|]
```

```
  | [x; y; z; t] -> [|x; y; z; t|]
  | _ -> failwith "Impossible"

let write_idx dimensions write_char =
  let write_uint n =
    let n = Int32.of_int n in
    let n = int32_to_net n in
    Array.iter write_char n
  in
  let rec range_aux read_char coordinates = function
    | [] -> write_char (read_char coordinates)
    | n :: tl ->
       for i=0 to n-1 do
         range_aux read_char (i :: coordinates) tl
       done
  in
  let range dimensions read_char = range_aux read_char [] dimensions in
  write_char 0;
  write_char 0;
  write_char 8;
  write_char (List.length dimensions);
  List.iter write_uint dimensions;
  range dimensions
```

3.2 Indexation

Ici, nous parcourons la base de données SHREC pour enregistrer les cibles. Nous obtenons une interface un peu plus complexe :

```
(** Search both train and test databases *)

(** Keep the association between target names and values *)
type database =
    {
      (** Database directory name *)
      dirname: Bytes.t;
      (** list of targets *)
      targets: Bytes.t array;
      (** Number of views *)
      n_views: int;
      (** List of train items - unsorted, conclusion * item ID *)
      train: (int * Bytes.t) array;
      (** List of test items *)
      test: (int * Bytes.t) array;
    }
```

```
17
18 (** Read the database under given directory *)
19 val read_database: Bytes.t -> database
```

Et une implémentation qui utilise le module `Sys` (`https://caml.inria.fr/pub/docs/manual-ocaml/libref/Sys.html`) :

```
1  type database =
2      {
3        dirname: Bytes.t;
4        targets: Bytes.t array;
5        n_views: int;
6        train: (int * Bytes.t) array;
7        test: (int * Bytes.t) array;
8      }
9
10 let randomize t =
11   let swap i j =
12     let x = t.(i) in
13     let () = t.(i) <- t.(j) in
14     t.(j) <- x
15   in
16   for i=1 to Array.length t - 1 do
17     swap i (Random.int i)
18   done
19
20 let read_database path =
21   let n_views = ref 0 in
22   let n_targets = ref 0 in
23   let targets = Hashtbl.create 100 in
24   let rec get_target name =
25     try
26       Hashtbl.find targets name
27     with Not_found ->
28          let () = Hashtbl.add targets name !n_targets in
29          let () = incr n_targets in
30          get_target name
31   in
32   let analyze_path path =
33     let b = Filename.basename path in
34     let triple = Filename.chop_extension b in
35     try
36       let sep_1 = Bytes.index triple '_' in
37       let sep_2 = Bytes.index_from triple (sep_1 + 1) '_' in
38       let start_target = 0 in
```

```
      let end_target = sep_1 in
      let len_target = end_target - start_target in
      let start_id = sep_1 + 1 in
      let end_id = sep_2 in
      let len_id = end_id - start_id in
      let start_view = sep_2 + 1 in
      let end_view = Bytes.length triple in
      let len_view = end_view - start_view in
      let (t, i, v) =
        (Bytes.sub triple start_target len_target,
         Bytes.sub triple start_id len_id,
         Bytes.sub triple start_view len_view) in
      let v = int_of_string v in
      let t = get_target t in
      let () = if v > !n_views then n_views := v in
      if v = 1 then
        [(t, i)]
      else []
    with _ -> []
  in
  let analyze_dir path =
    let files = Sys.readdir path in
    let () = randomize files in
    let files = Array.to_list files in
    let files = List.map (fun f -> Filename.concat path f) files in
    let items = List.map analyze_path files in
    List.concat items
  in
  let train_items = analyze_dir (Filename.concat path "train") in
  let test_items = analyze_dir (Filename.concat path "test") in
  let all_targets = Array.make !n_targets "" in
  let () = Hashtbl.iter (fun name i -> all_targets.(i) <- name) targets in
  {
    dirname = path;
    targets = all_targets;
    n_views = !n_views;
    train = Array.of_list train_items;
    test = Array.of_list test_items;
  }
```

3.3 Lecture des images

Ici, on utilise la bibliothèque Camlimages (`gallium.inria.fr/camlimages/`, documentation en ligne `file:///usr/share/doc/libcamlimages-ocaml-doc/html/api/index.html`).

L'interface est très simple, on récupère la matrice des pixels :

```
(** Read images *)

(** Read the image under the given filename *)
val read_image: Bytes.t -> int array array
```

L'implémentation est extrêmement simple :

```
let read_image filename =
  let image = Images.load filename [] in
  let (n, p, get_rgb) =
    match image with
    | Images.Index8 i8t ->
       (i8t.Index8.height, i8t.Index8.width,
        fun i j -> Index8.get_rgb i8t j i)
    | Images.Rgb24 rgb24t ->
       (rgb24t.Rgb24.height, rgb24t.Rgb24.width,
        fun i j -> Rgb24.get rgb24t j i)
    | Images.Index16 i16t ->
       (i16t.Index16.height, i16t.Index16.width,
        fun i j -> Index16.get_rgb i16t j i)
    | Images.Rgba32 rgba32t ->
       (rgba32t.Rgba32.height, rgba32t.Rgba32.width,
        fun i j ->
        let rgba = Rgba32.get rgba32t j i in
        rgba.Color.color)
    | Images.Cmyk32 cmyk32t ->
       failwith "Error: this is a CMYK image, I can't handle it"
  in
  let get i j =
    let rgb = get_rgb i j in
    (rgb.Color.r + rgb.Color.g + rgb.Color.b) / 3
  in
  Array.init n (fun i -> Array.init p (get i))
```

3.4 Fabrication du fichier à partir de la base de données

On effectue ici la conversion à proprement parler. L'interface fournit une fonction qui prend en entrée l'indexation et le dossier où écrire les tables idx :

```
(** Convert image database to idx *)

(** Convert the database to four files in the given directory. *)
val convert: Index.database -> Bytes.t -> unit
```

L'implémentation est un peu plus longue ; il faut charger les images une par une :

```
let convert db dirname =
  let open Index in
  let open Compute in
  let load_view dirname (target, item_id) view =
    let basename =
      Printf.sprintf
        "%s_%s_%03d.jpg"
        (db.targets.(target))
        item_id
        (view + 1) in
    let name = Filename.concat
                 (Filename.concat db.dirname dirname)
                 basename in
    Read_images.read_image name
  in
  let do_database write dir items =
    let i_item = ref 0 in
    let i_view = ref 0 in
    let i = ref 0 in
    let j = ref 0 in
    let view = ref (load_view dir items.(0) !i_view) in
    let get () =
      let v = !view in
      let i = !i in
      let j = !j in
      v.(i).(j)
    in
    let next () =
      let () = incr j in
      let v = !view in
      if !j >= Array.length (v.(!i)) then
        let () = incr i in
        let () = j := 0 in
        if !i >= Array.length v then
          let () = incr i_view in
          let () = i := 0 in
          let () = j := 0 in
          let () =
            if !i_view >= db.n_views then
              let () = incr i_item in
              let () = i_view := 0 in
              ()
          in
          if !i_item < Array.length items then
```

```
45              view := load_view dir items.(!i_item) !i_view
46      in
47      let get_next coord =
48        let x = get () in
49        let () = next () in
50        x
51      in
52      let dimensions =
53        [Array.length items;
54         db.n_views;
55         Array.length !view;
56         Array.length !view.(0)] in
57      write_idx dimensions write get_next
58    in
59    let do_labels write items =
60      let i = ref 0 in
61      let get_next _ =
62        let (x, _) = items.(!i) in
63        let () = incr i in
64        x
65      in
66      write_idx [Array.length items] write get_next
67    in
68    let train_images = open_out (Filename.concat
69                                   dirname "train-images-idx4.ubyte") in
70    let train_labels = open_out (Filename.concat
71                                   dirname "train-labels-idx1.ubyte") in
72    let test_images = open_out (Filename.concat
73                                  dirname "test-images-idx4.ubyte") in
74    let test_labels = open_out (Filename.concat
75                                  dirname "test-labels-idx1.ubyte") in
76    let dict = open_out (Filename.concat dirname "targets.txt") in
77    let () =
78      for i=0 to Array.length db.targets - 1 do
79        output_string dict (db.targets.(i) ^ "\n")
80      done
81    in
82    let write stream c =
83      let c = char_of_int c in
84      output_char stream c
85    in
86    do_database (write train_images) "train" db.train;
87    do_database (write test_images) "test" db.test;
88    do_labels (write train_labels) db.train;
89    do_labels (write test_labels) db.test
```

3.5 Application principale

On utilise les deux arguments du programme comme chemin d'accès aux dossiers de la base SHREC et MNIST :

```
match Sys.argv with
| [|_; input; output|] ->
   let db = Index.read_database input in
   Convert.convert db output
| _ ->
   Printf.printf "Usage: shrec-to-mnist [SHREC database] [MNIST output dir]\n%!"
```

3.6 Compilateur

Pour compiler le programme, on utilise `ocamlbuild`. Voir le manuel en ligne.

Nous voulons simplement compiler et linker les paquets Ocamlfind `camlimages` et `camlimages.jpeg`. Nous avons donc besoin d'un fichier `_tags` pour instruire ces dépendances externes.

```
<src/read_images.ml>: package(camlimages), package(camlimages.jpeg)
<src/main.{byte,native}>: package(camlimages), package(camlimages.jpeg)
```

Nous utilisons les tags `package()`, ce qui signifie qu'il faut lier le plugin `ocamlfind` au plugin de compilation ; pour ce faire on passe `-with-ocamlfind` à la compilation.

4 Tangle

Ici, nous extrayons le code source.

```
(org-babel-tangle)
```

Listing 2: Extraire le code source

```
tar cf shrec-to-mnist.tar.gz \
    idx/src/compute.ml \
    idx/src/compute.mli \
    idx/src/index.ml \
    idx/src/index.mli \
    idx/src/read_images.ml \
    idx/src/read_images.mli \
    idx/src/convert.ml \
    idx/src/convert.mli \
    idx/src/main.ml \
    idx/_tags
```

Listing 3: Empaquetage

L'archive est donc disponible ici.